Algorithmes et puzzles : une ultime approche de Turing

Lény Oumraou

Docteur agrégé de philosophie - Lycée Charles Péguy (Orléans) et Université Paris I

Paru l'année même de la mort de Turing (1954), l'article « *Solvable and unsolvable problems* » présente, de façon souvent esquissée et élémentaire, un certain nombre de résultats concernant la théorie de la calculabilité. Bien qu'il soit plutôt issu d'une entreprise de vulgarisation, il n'en comporte pas moins une démonstration rigoureuse de l'indécidabilité du problème de l'arrêt. Mais l'approche est très différente de celle des travaux fondateurs de 1936, et les inflexions que Turing fait subir à ces travaux antérieurs sont particulièrement instructives.

Le titre de l'article fait référence à des « problèmes ». Mais le terme désigne ici exclusivement des « problèmes de décision ». Turing distingue en effet scrupuleusement ces derniers des « puzzles » pour lesquelles sont recherchées des procédures de décision. Le même souci nous conduit à traduire ' puzzle ' par ' énigme ' (parfois ' casse-tête '), plutôt que ' problème '.

Un problème est donc une demande de la forme : « Existe-t-il une procédure de décision pour résoudre cette énigme ? ». Et un problème insoluble est une énigme pour laquelle le problème de décision n'est pas soluble (*i.e.* il n'existe pas de procédure de décision pour la résoudre). Dès lors, l'ensemble du propos de Turing repose sur deux notions, qu'il conviendra de clarifier avant toute autre chose : les notions d'énigme et de procédure effective. Or ces dernières seront elles-mêmes définies comme des énigmes qui remplissent un certain nombre de conditions. C'est donc la notion d'énigme (*puzzle*) qui joue ici un rôle central, et qui doit être précisée en premier lieu.

1. Des exemples d'énigmes.

Dans un premier temps, c'est en donnant des exemples d'énigmes que Turing entend familiariser le lecteur à cette notion. En un sens, il s'agit même de montrer au lecteur qu'elle lui est déjà familière. Car ce n'est pas un hasard si Turing ouvre son inventaire avec un jeu qui, certes, n'est commercialisé que depuis peu, lorsqu'il écrit, mais qui n'en est pas moins déjà populaire – à savoir le taquin. Comme nous le verrons, ce casse-tête, bien connu des enfants, joue un rôle quasi paradigmatique dans cette première étape de la démarche de Turing.

Sous sa forme commercialisée, le taquin se présente comme un carré divisé en seize cases (tantôt moins, tantôt plus, dans des variantes auxquelles nous ne nous intéresserons pas ici), quinze d'entre elles étant occupées par des carrés mobiles numérotés de 1 à 15, la seizième étant vide. Le jeu consiste à placer les carrés dans une configuration donnée, étant entendu que l'on ne peut déplacer les carrés qu'un par un, en coulissant dans la case vide un carré qui lui est adjacent.(voir Figure 1)

Figure 1

Configuration standard

Configuration accessible à partir de la configuration standard

Configuration inaccessible à partir de la configuration standard

Il convient de s'arrêter sur ce premier exemple, qui présente de façon particulièrement claire les caractéristiques essentielles des énigmes :

1. On dispose de configurations, qui sont autant d'arrangements d'un nombre déterminé de pièces – dans le cas du taquin, on a en principe 16! configurations possibles.
2. Ces pièces peuvent être déplacées conformément à des règles spécifiées, qui autorisent certains coups, et en interdisent d'autres – par exemple, pour le taquin, on ne peut « démonter » le jeu !
3. En déplaçant les pièces, on passe d'une configuration à une autre. Le but est d'obtenir une certaine configuration finale à partir d'un configuration initiale.
4. A chaque étape, on a le choix entre un nombre fini de coups possibles. Ainsi, dans le taquin, il n'y a que deux, trois ou quatre coups possibles, à chaque étape, en fonction de la position de la case vide.

Ce dernier point est important, car le caractère proprement « énigmatique » de l'énigme consiste précisément dans cette nécessité de faire le bon choix. Ce que veut savoir celui qui entreprend de résoudre l'énigme du taquin, c'est la nature du mouvement à effectuer à chaque étape. Il souhaiterait disposer d'un *plan d'exécution* qui élimine toutes les hésitations qui, à chaque étape, risquent de l'écarter du but recherché.

Mais Turing n'adopte pas tout à fait le point de vue de celui qui tente de résoudre l'énigme. Comme il le remarque dès le début de l'article, lorsque l'on peine à résoudre une énigme de ce type, on en vient à se demander si cela est possible. Et c'est essentiellement à cette nouvelle interrogation qu'il s'intéresse. Il faut remarquer qu'il s'agit bien là d'une modification profonde de la question de départ : on passe, en effet, de la question du comment (comment obtenir la configuration finale ?) à celle de la possibilité (est-il possible d'obtenir la configuration finale ?). Nous reviendrons plus loin sur les relations qui subsistent, en général, entre ces deux questions. Nous nous bornons pour le moment au cas du taquin. Or il est clair que, dans ce cas, la question de la possibilité renvoie à un tout autre jeu que celui que l'on trouve dans le commerce. Car, pour celui-ci, l'acquéreur fait généralement confiance au fabricant quant au fait qu'il *est* possible d'obtenir la configuration finale. Peut-être même le jeu lui a-t-il été vendu avec cette configuration, qu'il a défaite par la suite ; si les pièces n'ont jamais été sorties de leur cadre, le joueur peut ainsi être sûr qu'il est possible de retrouver cette configuration.

La question de la possibilité serait donc plus appropriée pour une autre version du jeu, d'ailleurs plus proche de celle de son inventeur, Sam Loyd, où ce sont des jetons qui sont déplacés sur un damier 4×4, en respectant la règle qui veut que l'on ne puisse les déplacer qu'en les glissant dans l'unique case vide. Dans ce cas, on peut imaginer que le joueur choisit lui-même une

configuration initiale et une configuration finale, et la première question qu'il lui faudra se poser sera bien celle de la possibilité.

Si l'on n'est toujours pas convaincu qu'il s'agit en fait d'une modification du jeu initial, on s'en convaincra en remarquant qu'il est, en principe, possible de répondre à la question de la possibilité sans répondre à celle du comment. Turing indique deux manières possibles de traiter la première question dans le cas du taquin. On peut montrer qu'une configuration peut être obtenue par les règles du taquin si et seulement si elle peut être obtenue par une permutation paire (ou un nombre pair de transpositions) – pourvu que la case vide se trouve dans la même position dans les deux configurations[1]. Ce critère repose entièrement sur le fait algébrique que l'on ne peut retrouver une configuration donnée en procédant à un nombre impair de transpositions ; pour cette raison, nous appellerons ce procédé la *méthode algébrique*. Clairement, on peut parfaitement imaginer vérifier, par ce procédé, qu'une configuration est accessible à partir de la configuration initiale, sans que l'on dispose d'une marche à suivre pour l'obtenir effectivement. Mais la question de la possibilité est en revanche complètement traitée par ce moyen ; car celui-ci permet aussi de montrer qu'une configuration n'est pas accessible, ce que l'échec de nos tentatives pour l'obtenir ne suffirait pas à prouver.

Le deuxième procédé traite, il est vrai, la question de possibilité et celle du comment de façon simultanée. Mais le « comment » se trouve traité d'une manière si peu praticable, si peu réaliste, pourrait-on dire, qu'il est clair que ce n'est qu'en vue de montrer la possibilité de principe que Turing le présente. Ce procédé consiste, en effet, à réaliser une partition de l'ensemble des configurations possibles, en définissant, sur cet ensemble, une relation d'équivalence : la relation d'accessibilité entre deux configurations. Formellement, on peut représenter les choses de la façon suivante : appelons C l'ensemble des configurations possibles, et notons R la relation d'accessibilité directe, *i.e.* la relation définie, pour $\alpha, \beta \in C$, par :

$\alpha \, R \, \beta$ si $\alpha = \beta$ ou β peut être obtenue à partir de α en glissant un carré dans la case vide.

R est clairement une relation réflexive et symétrique. Et sa clôture transitive R^*, définie par :

$\alpha \, R^* \, \beta$ s'il existe une suite $\alpha_1, \ldots, \alpha_n$ telle que $\alpha_1 = \alpha$, $\alpha_n = \beta$ et $\alpha_r \, R \, \alpha_{r+1}$ pour tout $r \in \{0,\ldots,n-1\}$

est une relation d'équivalence. L'ensemble quotient C/R^* constitué des R^*-classes d'équivalence est la partition cherchée. Ainsi, pour tout $\alpha \in C$, $[\alpha]_{R^*}$ est l'ensemble des configurations accessibles (directement ou indirectement) à partir de α. Et pour savoir si une configuration β est accessible à partir de la configuration α, il suffit de regarder si α et β appartiennent à la même classe, ce qui permet de fournir aussi bien une réponse positive que négative.

On voit qu'en pratique cette solution théorique du problème est de peu d'intérêt. Par exemple, comment pourrions-nous constituer la R^*-classes d'équivalence $[\alpha]$ sinon en progressant, pas à pas, de la configuration α à toutes celles qui peuvent être atteintes ? Cette manière de construire $[\alpha]$ peut être représentée par l'exploration d'un arbre dont la racine est (étiquetée) α, et où chaque nœud a pour descendants les configurations accessibles à partir de celle qui correspond à ce nœud. Comme il s'agit de dresser une liste exhaustive de toutes les configurations accessibles, l'exploration a lieu « en largeur d'abord », c'est-à-dire en développant tous les nœuds d'un même niveau, avant de passer au niveau suivant. Constituer une seule classe exigerait un travail d'une extrême complexité, et il est aussi vain de s'y engager que de tenter de construire toutes les classes ! Le but de Turing est seulement de souligner que, même si la *méthode de partition* (comme nous l'appellerons) implique l'examen d'un nombre rédhibitoire de cas, ce nombre reste fini, ce qui signifie, ici, que la méthode est au moins applicable...en principe. Autrement dit, par cette seconde

[1]Voir Annexe 1.

méthode, Turing s'éloigne d' un pas de plus de la question du comment : il cherche moins à montrer comment répondre à la question de la possibilité (ce que la méthode algébrique réalisait plus efficacement) qu'à montrer qu'il est possible d'y répondre.

Cette indifférence aux questions de « faisabilité » autorise Turing, lorsqu'il aborde d'autres exemples d'énigmes, à prendre la méthode de partition pour modèle. Il est vrai que si elle est moins efficace que la méthode algébrique, elle est cependant plus aisément généralisable. Et Turing semble choisir les exemples ultérieurs avec le seul souci de montrer comment contourner les obstacles qui pourraient s'opposer à cette généralisation de la méthode. Que faire, par exemple, lorsque les règles du jeu ne sont pas réversibles, c'est-à-dire lorsque l'on ne peut revenir d'une configuration atteinte à une configuration antérieure ? C'est le cas, remarque Turing, lorsque l'on fait une patience ; à vrai dire, les types de patience sont si variés, qu'il est difficile de donner une description de la situation suffisamment générale pour couvrir tous les cas. Mais les cas correspondant au propos de Turing sont assez nombreux pour être exemplaires. Typiquement, on part d'un certain ordre des cartes dans le paquet, on tire des cartes du paquet pour constituer des combinaisons de cartes, combinaisons qui peuvent avoir un nombre fixé d'éléments (comme dans « les Onze », « les Quatre Couleurs », *etc*.), ou un nombre variable (comme dans « les Alliances ») ; les règles du jeu permettent d'éliminer des cartes de cette combinaison, ou de lui en ajouter, et il faut souvent compter, comme « configuration », l'ensemble du dispositif constitué du paquet, de la combinaison et des cartes éliminées. Et il est vrai qu'en général, on ne revient pas en arrière : les cartes de la combinaison peuvent être éliminées, mais non remises dans le paquet, celles que l'on élimine ne sont remises ni dans le paquet ni dans la combinaison, *etc*. Le passage d'une configuration à l'autre est orienté dans une seule direction : on ne remonte pas l'arbre. La relation d'accessibilité directe n'est pas symétrique, et sa clôture transitive n'est pas une relation d'équivalence. Mais on peut encore la définir. De même qu'à défaut d'une classe d'équivalence, on peut définir l'ensemble des configurations accessibles à partir d'une configuration donnée, en procédant dans un seul sens, et jusqu'à saturation (*i.e.* jusqu'à ce qu'il ne soit plus possible d'ajouter une nouvelle configuration).

Mais l'absence de réversibilité des règles n'est pas le seul problème posé par les patiences, lorsque l'on tente de leur appliquer la méthode de partition. Comme le fait remarquer Turing, lorsque l'on pose les cartes que l'on sort du paquet, il y a, en principe, une infinité de positions qu'elles peuvent occuper. Or la méthode de partition, dans le cas du taquin, reposait évidemment sur le fait que le nombre de positions que peuvent occuper les pièces y était fini. Le problème peut paraître dérisoire, lorsqu'il est soulevé à propos des patiences. Mais il l'est beaucoup moins si l'on considère deux autres énigmes envisagées par Turing ; le première est un casse-tête que l'on trouve également dans le commerce, et qui consiste en un certain nombre de barres métalliques entrelacées qu'il est demandé de séparer ; la seconde a une expression mathématique bien connue : il s'agit de l'énigme consistant à dénouer un nœud, ou à transformer un nœud en un autre ; dans ce cas, la question de possibilité revient clairement à se demander si le nœud de départ est équivalent (homotopique) au nœud auquel on veut aboutir. On voit que, dans ces deux cas, on passe de façon continue d'une configuration à une autre, de sorte qu'entre deux configurations données, on peut en intercaler une infinité d'autres, ce qui rend considérablement plus difficile la définition d'une relation d'accessibilité directe.

Figure 2

Barres métalliques entrelacées

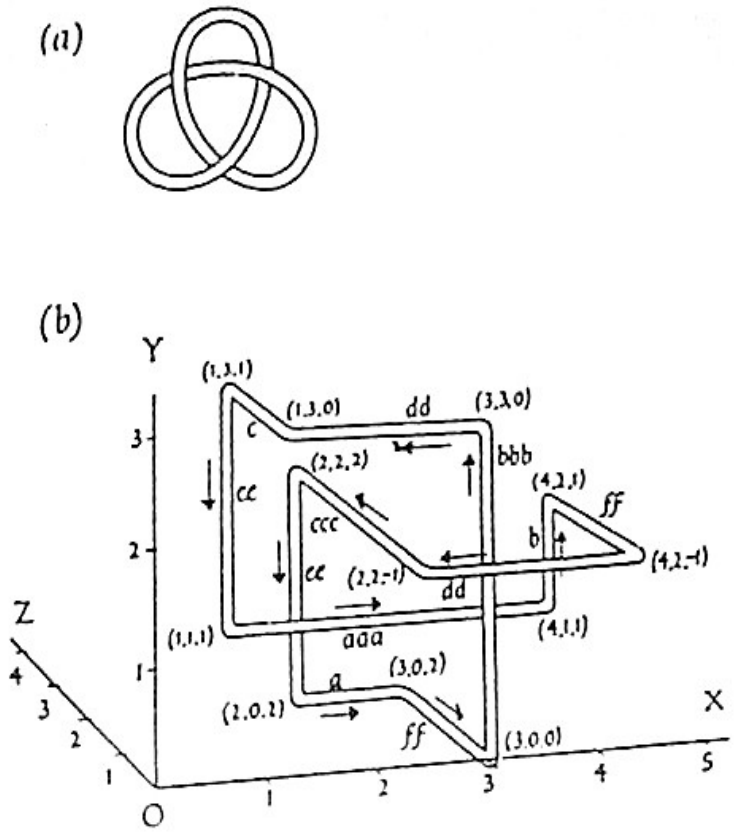
Pour assurer l'applicabilité de la méthode de partition, Turing soutient que, malgré tout, dans ces trois dernières énigmes, même s'il y a une infinité de positions, seul un nombre fini d'entre elles s'avèrent être « essentiellement différentes ». Turing ne cherche pas à justifier cette affirmation, qui est sans doute moins évidente dans le cas des nœuds que dans celui des patiences. Tout au plus souligne-t-il que, pour les deux dernières énigmes évoquées, une description mathématique du dispositif sera nécessaire pour étudier les positions essentiellement différentes. Et c'est à la nature de cette description qu'il va s'intéresser dans un second temps.

2. Des énigmes en général aux énigmes de substitution (et retour).

La manière dont Turing formalise, et esquisse (car il ne va pas plus loin) le traitement de l'énigme des nœuds est importante. Elle conduit, en effet, directement aux énigmes de substitution, qui fourniront à leur tour une formalisation générale pour toutes les énigmes. La démarche de Turing est ici stratégique : en introduisant la formalisation des énigmes par celle qui paraît le plus retors au traitement de celles-ci, il donne un puissant argument en faveur de la généralité de cette formalisation.

Rappelons qu'un nœud, en mathématique, est une courbe fermée dans l'espace tridimensionnel, et qui ne se croise jamais elle-même, c'est-à-dire un plongement du cercle dans l'espace tridimensionnel. Pour venir à bout du cas des nœuds, Turing convient de les décomposer en segments, la version segmentée que l'on obtient du nœud d'origine étant clairement homéomorphe à ce nœud. Mais ce n'est là qu'une première étape. Pour traiter le problème mathématiquement, estime Turing, il reste à représenter cet arrangement de segments par un arrangement linéaire de symboles, c'est-à-dire une expression, ou un mot. Turing y parvient en se donnant un système de coordonnées cartésiennes, et en représentant, par des lettres, les translations selon les vecteurs unité e_1, e_2, e_3, ainsi que leurs opposés. Il en résulte un alphabet de six lettres : *a*, *b*, et *c*, pour les translations selon, respectivement e_1, e_2, et e_3 ; *d*, *e*, et *f* pour les translations selon les vecteurs opposés (respectivement). C'est ainsi que le noeud de trèfle peut être décrit, par exemple, par l'expression *aaabffddcccceeaffbbbcee* (ce qu'il faut lire : une translation de $3e_1$, suivie d'une translation de e_2, *etc*.). Bien entendu, une infinité d'autres expressions peuvent remplir la même fonction, pour autant qu'elles représentent des courbes topologiquement équivalentes.

Figure 3



(a) Nœud de trèfle (b) Codage de Turing [Turing 54, p. 13]

On a ainsi réduit un nœud à un mot ; pour poursuivre le traitement formel de cette énigme, il resterait encore à exprimer les transformations que l'on est autorisé à réaliser sur les nœuds par des opérations permises sur les mots correspondants. Or, ces opérations prennent la forme de règles de réécriture, de sorte que la formalisation cherchée de l'énigme des nœuds revient à définir ce que Turing appelle une énigme de substitution.

Les énigmes de substitution sont initialement décrites comme des casse-têtes dans lesquels on dispose de jetons alignés ; le jeu consiste à passer d'une configuration initiale à une configuration finale, en appliquant un ensemble donné de substitutions de suites de jetons à d'autres suites de jetons. On a un nombre fini de type de jetons (par exemple des jetons blancs et et des jetons noirs), et pour chaque type, il est supposé que l'on dispose d'une infinité de jetons de ce type (on aura ainsi une infinité de jetons blancs, et une infinité de jetons noirs). Cette dernière stipulation rend le jeu physiquement irréalisable (on ne le trouvera pas dans le commerce...), mais on perçoit d'emblée l'analogie avec les langages formels : la relation entre les types de jetons et les jetons est la même qu'entre les symboles (comme types) et leurs occurrences. Les arrangements linéaires de jetons sont donc assimilables à des expressions, ou mots, dans un alphabet fini (par exemple ne comportant que les symboles *W*, pour « White », et *B*, pour « Black »).

Dans cette transcription symbolique, les substitutions autorisées sont représentées par ce que Turing appelle des « paires de substitution ». Il s'agit plus précisément de paires ordonnées, des

couples de mots, signifiant qu'une occurrence du premier peut être remplacée par une occurrence du second. Un tel couple peut s'écrire sous la forme :

$$A \rightarrow B$$

On peut appeler *A* le *mot source*, et *B* le *mot cible* de la paire de substitution. Par exemple, les paires de substitution :

$$WW \rightarrow W$$
$$BB \rightarrow B$$

permettent de remplacer deux « *W* », ou deux « *B* » successifs par un unique « *W* », pour la première, ou une un unique « *B* », pour la seconde. On voit alors immédiatement que du mot :

$$WWWBBWBBBBWWWWW$$

on pourra obtenir le mot :

$$WBWBW$$

par ces règles. Par exemple, via la suite de substitutions :

$$WWWBBWBBBBWWWWW \rightarrow WWBBWBBBBWWWWW \rightarrow WBBWBBBBWWWWW \rightarrow WBWBBBBWWWWW$$
$$\rightarrow WBWBBBWWWWW \rightarrow WBWBBWWWWW \rightarrow WBWBWWWWW \rightarrow WBWBWWWW \rightarrow WBWBWWW$$
$$\rightarrow WBWBWW \rightarrow WBWBW$$

Ici, la stratégie a consisté à appliquer les règles sur les occurrences susceptibles d'être traitées et se trouvant les plus à gauche ; mais il est clair que ces règles pouvaient être appliquées dans des ordres différents. Il est clair aussi que toute alternance de suites de « *W* » et de « *B* » peut être réduite, par ces règles, à une simple alternance de « *W* » et de « *B* », à laquelle on ne pourra plus appliquer ces règles.

Une énigme de substitution consiste donc à donner une liste de paires de substitutions, ainsi qu'un mot initial et un mot final, la question étant de déterminer si le mot final peut être obtenu à partir du mot initial, en appliquant uniquement des paires de substitution de la liste.

On voit nettement comment la version symbolique de l'énigme traduit directement la version initiale, les symboles se substituant aux jetons. Mais Turing franchit un pas supplémentaire, en soutenant que *toutes* les énigmes peuvent être ainsi traduites. Cela implique que toutes les configurations susceptibles d'intervenir dans les énigmes peuvent être représentées par des configurations linéaires de symboles (des mots) ; et tous les « coups » autorisés dans une énigme peuvent être traduits par des paires de substitution.

On remarquera que l'assertion de Turing ne saurait être prouvée, car si la définition des énigmes de substitution est précise, celle d'énigme en général ne l'est pas ; Turing n'en a donné que quelques illustrations, dont nous avons pu, certes, abstraire les grands traits, mais ceux-ci restent flous (comme les expressions « pièces », « configurations », « coups », *etc*). On comprend dès lors que le choix de ces exemples n'était pas anodin. Les configurations du taquin étaient bidimensionnelles, les nœuds se déploient dans l'espace tridimensionnel. Tout se passe comme si Turing avait voulu dire : si ces énigmes peuvent être ramenées à des énigmes de substitution, dont les configurations sont pourtant linéaires, nous aurons de bonnes raisons de penser que toutes le pourront.

Turing remarque qu'il s'agit là moins de preuve que de propagande. On sait bien qu'il en est de même de la thèse de Church-Turing (voir Annexe 2). Mais n'allons pas trop vite : à ce stade, Turing n'a défini que les énigmes, non les procédures systématiques. Qu'en est-il de ces dernières ?

3. Des énigmes aux « procédures systématiques ».

Selon Turing une procédure systématique est une énigme de substitution « dans laquelle il n'y a jamais plus d'un coup possible dans l'une quelconque des positions qui surgissent, et dans laquelle quelque signification est attachée au résultat final ». Commençons par quelques commentaires de cette définition.

En premier lieu, il s'agit de nouveau d'une thèse. Et même, cette fois, nous sommes très proches de ce qu'il est convenu d'appeler la Thèse de Church-Turing[2]. Ce qui nous en sépare, c'est d'abord que, comme nous le verrons, la nature des énigmes de substitution impliquées n'est pas déterminée de façon unique par les caractéristiques qu'en donne Turing. Ensuite, une fois précisée la nature de ces énigmes, il resterait encore à montrer qu'elles définissent les mêmes classes de fonctions calculables (disons) que les machines de Turing – puisque ce sont-elles qui servent de *definiens* dans la formulation « classique » de la thèse de Church-Turing. Mais l'essentiel réside en ceci que Turing part d'une idée informelle, supposée commune, de « procédure systématique », et affirme qu'il est possible d'en donner une traduction formelle par des énigmes de substitution satisfaisant certaines conditions.

De quelles conditions s'agit-il ? Essentiellement d'une *condition de détermination* et d'une *condition de terminaison*. La première consiste au fond à remplacer le point 4 énoncé dans la section 1 par le point :

4' . A chaque étape, un seul coup est possible.

Dans le prolongement de ce que nous disions plus haut, une procédure systématique se présente donc, paradoxalement, comme une énigme qui n'a plus rien d'énigmatique. Il faut comprendre qu'en toute circonstance, au cours du jeu, on n'est jamais indécis quant à ce qu'il faut faire car, *d'une manière ou d'une autre*, les règles du jeu indiquent la paire de substitution qui doit être appliquée. Ou, pour le dire autrement, une procédure systématique définit une unique branche de l'arbre que nous devions explorer pour construire les classes d'équivalence de la méthode de partition.

On peut ainsi comprendre en quoi consistera la condition de terminaison : la procédure systématique définira une unique branche, qui devra de surcroît être finie. De ce point de vue, l'expression que Turing utilise - « quelque signification est attachée au résultat final » - est quelque peu sibylline ; pour commencer, elle suggère qu'il y a toujours un résultat final, ce qui veut dire qu'une procédure systématique s'arrête toujours. Mais en disant qu'une signification est attachée au résultat final, Turing ajoute qu'elle ne s'arrête pas « au hasard » ; il faut probablement comprendre qu'elle s'arrête sur une configuration, ou un nombre fini donné de configurations possibles, dont la nature est prédéfinie, et qui donnent une réponse à une question posée. Par exemple, une procédure systématique qui constitue un test de primalité s'appliquera à des mots initiaux désignant des nombres ; et s'arrêtera sur un mot final signifiant « oui » ou « non », en réponse à la question : « le nombre représenté par le mot initial est il premier ? ».

Mais Turing reste très évasif quant à la manière dont les deux conditions qu'il pose peuvent être satisfaites par une énigme (de substitution). Afin d'y apporter plus de précisions, on peut remarquer que les « paires de substitution » sont finalement très proches des « prescriptions » des algorithmes introduits par Markov [1951]. Les procédures systématiques de Turing peuvent être formalisées par les « algorithmes normaux » de Markov. Or, pour qu'un ensemble de prescriptions (dans un alphabet donné) définissent un algorithme normal, il doit remplir les conditions suivantes :

a) Ou bien l'ensemble des prescriptions est ordonné, *i.e.* un ordre de préséance doit être respecté dans leur application, de sorte qu'elles ne constituent pas seulement un ensemble, mais une suite; ou bien deux prescriptions distinctes n'ont pas les mêmes expressions à gauche de la flèche.

[2] Voir Annexe 2.

b) Au cours d'une application des prescriptions, on applique la première prescription de la liste qui se trouve être applicable.

c) Chaque prescription est appliquée sur la première occurrence du mot auquel elle est applicable, et qui se trouve à l'intérieur du mot auquel elle est appliquée.

En remplaçant l'expression « prescription » par « paire de substitution », dans ces stipulations, on s'approche sans doute de la manière dont Turing entendait procéder pour faire de ses énigmes des « énigmes à coups univoques », c'est-à-dire des procédures effectives.

4. L'usage de symboles auxiliaires.

Tout comme Turing, Markov affirmait la nécessité de recourir, pour certains calculs, à des symboles auxiliaires, qui ne figureront pas dans les résultats, mais qui doivent néanmoins apparaître dans la formulation de certaines paires de substitution. Par exemple, pour définir un algorithme permettant de doubler chaque symbole d'une expression E, dans l'alphabet $\{W, B\}$, on procède en transformant l'expression E en αE, où α est un symbole auxiliaire. On applique ensuite, selon la nature du symbole qui suit α, une des prescriptions :

$$\alpha W \rightarrow W\beta W\alpha \qquad \alpha B \rightarrow B\beta B\alpha$$

permettant de recopier la première lettre de E, tout en marquant cette copie par un nouveau symbole auxiliaire β ; le α de droite permet, à son tour, de réitérer cette procédure pour la deuxième lettre de E, et ainsi de suite. Si, par exemple, $E = WBW$, on se retrouve, à la fin de ces étapes, avec l'expression :

$$W\beta WB\beta BW\beta W\alpha$$

Il reste à faire disparaître les symboles auxiliaires, ce qui peut être réalisé par les prescriptions $\alpha\rightarrow$ et $\beta\rightarrow$. L'algorithme complet s'écrirait donc :

$$\begin{aligned} &\alpha W \rightarrow W\beta W\alpha \\ &\alpha B \rightarrow B\beta B\alpha \\ &\alpha \rightarrow \\ &\beta \rightarrow \\ &\rightarrow \alpha \end{aligned}$$

Ainsi, au début de la procédure, il n'y a aucun symbole auxiliaire dans E, de sorte que la seule paire de substitution applicable est la dernière de la liste, dont l'effet est d'introduire un α au début de l'expression (elle s'applique en effet à la première occurrence du mot vide). De même, la troisième prescription, celle qui supprime α, ne sera pas appliquée tant que les précédentes pourront l'être, ce qui assure que α ne sera pas supprimé trop tôt. On voit ainsi l'importance de l'ordre des prescriptions.

5. Énigmes de substitution et machines de Turing.

Ainsi, Turing énonce ici la thèse de Church-Turing[3] en recourant à une formalisation proche de celle de Markov, qui prolongeait lui-même des travaux de Post [1943]. L'équivalence de la thèse de Markov et de la thèse de Church a été démontrée par Detlovs [1953]. Mais il vaut la peine de comparer les travaux de Turing [1936] à l'approche du présent article.

Nous avons remarqué, dans la Section 2, qu'en réduisant toutes les énigmes aux énigmes de substitution, Turing affirmait que toute configuration susceptible d'apparaître dans une énigme peut être représentée par une suite linéaire de symboles. En appliquant cette assertion au cas particulier

[3]Voir Annexe 2.

des procédures systématiques (qui ne sont que des énigmes d'un certain type), on voit que Turing réaffirme ainsi la suffisance de la linéarité, qu'il avait déjà soulignée en 1936, à partir d'une analyse de la pratique du calcul d'écolier. Cette analyse l'avait, en effet, conduit de la feuille quadrillée usuelle, au ruban caractéristique de la « machine de Turing » :

> Le calcul se fait en écrivant certains symboles sur le papier. Nous pouvons supposer que ce papier est divisé en carrés comme sur le carnet d'arithmétique d'enfant. En arithmétique élémentaire, le caractère bidimensionnel du papier est parfois utilisé. Mais un tel usage peut toujours être évité, et je crois que l'on accordera que le caractère bidimensionnel du papier n'est pas essentiel au calcul. Je supposerai donc que le calcul est effectué sur un papier à une dimension, *i.e.* sur un ruban divisé en carrés [1936, p135]

Mais il existe une différence de taille entre la manière dont une machine classique de Turing « gère » son ruban, et celle dont se pratiquent les substitutions dans les arrangements de l'énigme standard de 1954. Dans les premières, les substitutions sont exécutées lettre par lettre (à chaque pas de substitution, on remplace une unique lettre par une unique lettre), tandis que les règles des énigmes de substitution autorisent le remplacement d'un mot par un mot, en un seul « coup ». Ce traitement des mots, plutôt que des lettres, rend, par ailleurs, superflue toute commande de déplacement, telle que celles qui permettent de mouvoir, case par case, la tête de lecture d'une machine de Turing, vers la gauche ou vers la droite. On peut dire, avec W. Sieg [1996], que Turing fait ici appel à des machines à mots (*string machines*), en contraste avec les « machines à lettres » des travaux antérieurs.

Il est vrai que dans l'article de 1936, Turing avait nettement marqué sa volonté de procéder à une analyse *complète* des opérations « complexes » qui peuvent intervenir dans un calcul ; il s'agit alors d'en extraire des opérations non seulement plus simples, mais aussi *atomiques*, en poussant la décomposition jusqu'à sa limite :

> Imaginons que les opérations effectuées par le calculateur soient décomposées [split up] en « opérations simples » qui soient si élémentaires qu'il n'est pas facile de les imaginer divisées davantage (...) [*Ibid.*]

On serait alors tenté d'en conclure que si le présent article se contente des machines à mots, c'est parce qu'étant destiné à la vulgarisation, il se soucie moins de donner une analyse complète des calculs. La situation est cependant moins simple qu'il n'y paraît. A la suite de Gödel, W. Sieg soutient que la démarche de Turing 1936 comportait deux étapes : il s'agissait, *dans un premier temps*, d'analyser la notion informelle de procédure mécanique, puis, *dans un second temps* seulement, de la réduire à des opérations élémentaires. Or, si la deuxième étape conduit aux machines à lettres auxquelles Turing a donné son nom, la première aboutit aux machines à mots. L'importance de ces dernières serait, dès lors, nettement soulignée par ceci que *a)* elles sont plus générales que les machines à lettres ; *b)* elles assument la totalité de l'élément non prouvable de la thèse de Church-Turing. Celle-ci se décomposerait en effet comme suit :

α) toute procédure mécanique peut être exécutée par une machine à mots, et

β) tout calcul exécuté par une machine à mots peut être exécutée par une machine à lettres.

Il est clair que *β)* est un énoncé que l'on peut rigoureusement prouver : un théorème, non une thèse. Ce n'est que pour autant qu'elle renferme *α)* que la thèse de Church-Turing est une thèse[4]. D'après cette interprétation, l'article de 1954 ne retient de la démarche originelle de Turing que sa partie la moins triviale, ou la plus problématique. Elle n'est pas moins « formelle », elle est seulement plus générale ; le caractère informel du propos étant peut-être bien, ici, au service de sa généralité.

6. Des énigmes aux problèmes.

D'après ce que nous avons dit précédemment, une énigme est un triplet (P, m_I, m_F), où P est un ensemble de « paires de substitution », m_I et m_F des mots, les uns et les autres étant définis sur un

[4] Pour cette raison, Sieg appelle *α)* la *thèse centrale*, tandis que *β)* est appelé *thèse de Turing*.

alphabet donné ; l'énigme de base consiste à demander d'obtenir m_F en partant de m_I, et en n'utilisant que des paires de substitution de P. Résoudre l'énigme, c'est donc donner un sous-ensemble P_0 de P tel que P_0 constitue une procédure systématique, et le résultat de l'application de P_0 à m_I est m_F. On répond alors à la question :

(1) « Comment résoudre l'énigme ? »

Mais selon Turing, lorsque l'on peine à la résoudre, il est naturel de se demander :

(2) « Est-il possible de résoudre cette énigme ? »

On passe ainsi de la *recherche d'une solution* à l'*examen de la résolubilité*, de la réalisation à la réalisabilité (de principe). C'est à ce dernier type de question que Turing réserve le nom de *problème*. Un pas de plus, et nous en venons à nous demander, plus généralement :

(3) « Est-il possible de savoir à l'avance (*i.e.* avant même d'en chercher une solution) si une énigme est résoluble ou non ? »

Maintenant, étant donné la nature des énigmes, (2) peut être interprété par :

(2)' « Existe-t-il une procédure systématique pour résoudre cette énigme ? »

Et le passage de (2) à (2)' suggère d'interpréter (3) par :

(3)' « Existe-t-il une procédure systématique permettant de savoir à l'avance si une énigme quelconque est soluble ou non ? »

C'est la réponse négative à cette question qui est annoncée dès le début de l'article, et que Turing établit rigoureusement dans la suite. Or il le fait en transformant le problème en énigme.

7. L'indécidabilité de la résolubilité des énigmes et le problème de l'arrêt.

Dès le début de l'article, Turing présente comme l'objectif premier de son étude la preuve de cette affirmation :

(...) il n'existe aucune méthode systématique permettant de tester les énigmes, pour voir si elles sont solubles ou non.

Cette assertion est essentiellement liée au problème de l'arrêt. Ce que va montrer Turing, en effet, c'est qu'il n'existe pas de procédure systématique permettant de déterminer si les énigmes terminent ou non. Pour l'établir, il montre, en fait, qu'une solution de ce problème fournirait celle d'un autre problème, dont il démontre pourtant l'insolubilité (procédé classique de la théorie de la calculabilité).

Turing commence par établir que l'ensemble des paires de substitution d'une énigme peut être représenté par un seul mot *R* : il suffit de mettre les paires de substitution l'une à la suite des autres (l'ordre n'ayant pas toujours d'importance), en les séparant par quelque symbole de ponctuation. Mais pour les besoins de la preuve, aucun symbole ne figurant pas dans l'alphabet de base ne doit figurer dans *R*. Turing propose une méthode pour éliminer les symboles de ponctuation, ainsi que les flèches figurant dans les paires de substitution. Il conviendrait de l'appliquer aussi au cas des symboles auxiliaires, dont curieusement, Turing ne parle pas. La méthode consiste à coder les symboles que l'on veut éliminer par une répétition, un nombre approprié de fois, de symboles « autorisés » ; par exemple, les flèches sont codées par un symbole

de l'alphabet de base, qui doit être différent des symboles qui se trouvent à ses côtés (ce qui suppose que l'alphabet contienne au moins trois lettres, *W*, *B*, *C*); chaque lettre apparaissant dans une paire de substitution est elle-même doublée, tandis que les symboles de ponctuation sont remplacés par une suite de trois lettres.

Maintenant, on peut encore définir $P(R,S)$, désignant une énigme dont les règles sont décrites par le mot *R*, et dont la position initiale est décrite par le mot *S*. On peut alors donner un sens à l'expression $P(R, R)$. Soulignons que, dans cette expression, *R* ne désigne pas nécessairement une procédure systématique. Nous pouvons parler de « procédure » en général, plutôt que de procédure systématique. La démonstration de Turing consiste alors à établir une bipartition de la classe des procédures *R* :

$R \in C_1$ si *R* est une procédure systématique, et $r[P(R, R)] = W$.
$R \in C_2$ sinon.

(Ici, $r[P(R, S)]$ est le résultat, supposé exister, de l'application de la procédure systématique *R* au mot *S*.)

Ainsi *R*, partant de la configuration *R*, ou bien ne s'arrête jamais (et alors est dans C_2), ou bien s'arrête ; dans ce dernier cas, le mot final est soit *W* (auquel cas *R* est dans C_1), soit distincte de *W* (et *R* est dans C_2). Supposons, maintenant, qu'il existe une procédure systématique *K* permettant de décider si une procédure systématique appartient à C_1 ou à C_2. Plus exactement, on a :

$$r[P(K,R)] = \begin{cases} B & \text{si } R \in C_1 \\ W & \text{si } R \in C_2 \end{cases}$$

Dès lors, si *K* est dans C_1, alors $r[P(K, K)]=W$, d'après la définition de C_1, ce qui contredit le fait que $r[P(K, K)] = B$ si $K \in C_1$, d'après la définition de *K*. Si, en revanche, *K* est dans C_2, on a $r[P(K, K)] = W$, d'après la définition de *K*, mais alors *K* est dans C_1, d'après la définition de C_1. En somme, $K \in C_1$ ssi $K \in C_2$, ce qui est contradictoire.

On a ainsi montré qu'il n'existe pas de procédure systématique permettant de dire si une procédure (au sens large) appartient à C_1 ou C_2. Comment ce résultat se rattache-t-il à celui que Turing voulait établir ? Simplement, si le problème de l'arrêt était soluble, autrement dit, s'il existait une procédure systématique pour déterminer si les énigmes aboutissent ou non, nous pourrions nous servir de cette procédure pour déterminer si les énigmes appartiennent à C_1 ou à C_2 (tous les autres éléments de la définition de ces classes étant décidables), ce qui contredirait le résultat qui vient d'être établi.

8. Des jeux sérieux.

Dans le choix de ses exemples, Turing mêle habilement les considérations ludiques aux aspects mathématiques des énigmes. Ce sont parfois des problèmes mathématiques d'une grande complexité, ou d'une grande importance théorique, qui se cachent derrière le divertissement que nous procurent les « casse-têtes » par lesquels il illustre la notion d'énigme. Les exemples les plus attendus de résultats théoriquement importants concernent les questions de décidabilité des théories. Turing remarque que « la tâche de démontrer un théorème mathématique donné, à l'intérieur d'un système axiomatique, est un très bon exemple d'énigme ». Les règles de formation, et de transformation, dans un système axiomatique formel sont, en effet, assimilables aux règles d'une énigme. Étant donnés des énoncés initiaux (dans un certain langage), et des règles permettant d'obtenir des énoncés à partir d'énoncés donnés (les règles de transformation au sens de Carnap), l'énigme consiste à savoir si un énoncé du langage peut être obtenu à partir des axiomes. Le caractère syntaxique de ces règles d'inférence permet d'assimiler la démonstration d'un énoncé à un

calcul, opérant mécaniquement à partir des axiomes (tout comme on obtenait la configuration finale par une suite de substitutions, dans les précédentes énigmes).

En prolongeant cette comparaison, on obtient le problème classique de décision pour les théories formelles : pour une théorie *T* donnée, existe-t-il une procédure effective permettant de décider si un énoncé donné (du langage de la théorie) est, ou non, un théorème de *T* ? On sait que c'est notamment le cas des théories du premier ordre qui sont à la fois récursivement énumérables et complètes[5]. Avec ces théories, en effet, on se trouve dans la situation décrite par Turing à propos du taquin : on est muni d'un test effectif aussi bien pour les énoncés démontrables que pour ceux qui ne le sont pas. Mais le premier théorème d'incomplétude de Gödel montre que la complétude est un phénomène relativement exceptionnel[6].

On ne s'étonnera guère de ces applications classiques de l'étude des « énigmes » à la logique mathématique ; après tout, le lien entre le problème de l'arrêt et la décidabilité de l'arithmétique est bien connu (si l'arithmétique était décidable, le problème de l'arrêt serait soluble) et cette application de la théorie de la calculabilité était celle développée par Turing en 1936. Il se montre, ici, plus soucieux d'évoquer des applications hors de la logique. C'est en particulier le cas des considérations sur les nœuds, dont la classification occupe encore l'attention des mathématiciens. La recherche d'invariants permettant de déterminer si un nœud peut être obtenu à partir d'un autre (en somme, s'ils sont le même nœud), recherche qui s'est particulièrement développée depuis les années 80, s'inscrit parfaitement dans les recherches de Turing : lorsque l'on remarque que les deux nœuds diffèrent pour quelque invariant, on peut répondre négativement à la question de principe sur la possibilité de la transformation. Ces questions sont loin d'être résolues, et l'on ne dispose pas encore d'invariants « complets », dont la présence, dans deux nœuds donnés, suffirait à conclure que l'on peut transformer l'un en l'autre. Le résultat le plus proche des considérations de Turing est probablement celui obtenu par Hemion [1979], qui montre que la question de savoir si un nœud est identique à un autre est décidable. Turing aurait probablement jugé de peu d'intérêt le fait que l'algorithme concerné est d'une trop grande complexité pour pouvoir être appliqué, même dans des cas simples.

Du point de vue de la théorie de la calculabilité elle-même, l'approche de Turing 1953 reste assez inhabituelle : plutôt que de partir des problèmes de décision, qui impliquent immédiatement la référence aux procédures effectives, il se base sur la notion d'énigme (*puzzle*) qui renvoie à un ensemble d'investigations plus universel, couvrant aussi bien les casse-têtes que l'on trouve dans le commerce que des recherches mathématiques abstraites. Ce n'est qu'à partir de ce point que Turing définit les algorithmes comme solutions déterminées des énigmes. Par cette approche, sa présentation appartient fondamentalement à une analyse générale des procédés d'investigation et de résolution.

BIBLIOGRAPHIE

[5] Rappelons que lorsque l'ensemble des axiomes d'une théorie *T* du premier ordre est « récursif » (c'est-à-dire décidable : il existe une procédure effective permettant de déterminer si un énoncé du langage de *T* est ou non un théorème), alors *T* est récursivement énumérable (ou semi-décidable) : il existe une procédure effective permettant de dire qu'un énoncé est un théorème, *s'il en est un* ; en général, cette procédure ne permet pas de déterminer qu'un énoncé n'est pas un théorème, s'il ne l'est pas. Ainsi, *T* peut être semi-décidable, sans être décidable. Mais si *T* est en outre complète, alors *T* est décidable. Car si φ est un énoncé du langage de *T*, φ ou $\neg\varphi$ est une théorème (c'est en cela que consiste la complétude de *T*) ; il suffit donc d'appliquer la procédure, parallèlement, à ces deux énoncés pour obtenir une réponse positive pour l'un d'eux, au bout d'un nombre fini d'étapes. Pourvu que *T* soit consistante (non-contradictoire), si l'un des énoncés est théorème de *T*, l'autre ne l'est pas (Lassaigne & Rougemont, Chap. 8, p 173).

[6] D'après ce théorème toute théorie qui inclut une arithmétique très élémentaire (arithmétique de Robinson) est incomplète (Ibid., p 175).

Curry, H. B.
[1963] *Foundation of mathematical logic*, *McGraw-Hill Book Company, Inc*.

Davis, M.
[1958] *Computability and unsolvability*, New York.

Hemion, G.
[1979] *On the classification of the homeomorphisms of 2-manifolds and the classification of 3-manifolds, Acta. Math.*

R. Lassaigne, M. de Rougemont
[1993] *Logique et fondements de l'informatique*, Hermès, Paris.

Markov, A. A.
[1951] *Teoriya algorifmov* (*Theory of algorithms*), *Trudy Mat. Inst. Steklov*, 38: 176-189. Traduction anglaise par Edwin Hewitt *in American Mathematical Society Translations*, 2, 15: 1-14, 1960.
[1954] *Teoriya algorifmov* (*Theory of algorithms*), *Trudy Mat. Inst. Steklov*, vol 42.

Post, E. L
[1943] *Formal Reduction of the General Combinatorial Decision Problem*, *Amer. J. Math.*, **65**:197-215.

[1947] *Recursive unsolvability of a problem of thue, in The Undecidable, Journal of Symbolic Logic 12 (1) ; The Colected Workx of Emil L. Post, Birkhäuser, Boston a.o., 1994, pp. 503-513.*

Sieg, W. et Byrnes, J.
[1996] *K-graph machines : generalizing Turing's machines and arguments* ; *in Gödel*'96 (Edité par P. Hayek) ; *Lectures Notes in Logic* 6, *Springer Verlag*, Berlin, 98-119.

Turing, A.
[1936] '*On computable numbers, with an apllication to the Entscheidungsproblem*', *Proc. London Math. Soc*., 1937, 230-265.
[1954] '*Solvable and Unsolvable problems'*. *Science News* 31 (1953), 7-23.

Annexe 1 : Taquin et permutations

Turing traite le cas du taquin en rappelant le résultat suivant :

Théorème : Soient α et β deux configurations du taquin dans lesquelles la case vide est placée en même position. Alors β est accessible à partir de α *ssi* β est une permutation paire de α.

On montre l'équivalence en deux étapes ; la partie triviale est :

1. Si β est accessible à partir de α, alors β est une permutation paire de α.

Preuve : en principe, ce sont les carrés que l'on déplace. Mais on peut s'intéresser aux déplacements correspondants de la case vide. Associons le nombre 1 à la position initiale de la case vide, et convenons qu'à chaque déplacement élémentaire elle change de signe. Comme elle ne peut se déplacer que verticalement, et horizontalement, dans les cases adjacentes, on peut associer à chaque case des valeurs *fixées* par le parcours de la case vide, indépendamment de l'ordre chronologique de ce parcours. Ces valeurs sont réparties en damier, c'est-à-dire que deux cases adjacentes n'ont jamais la même valeur. Par exemple, si la position initiale de la case vide est la case la plus basse, et la plus à droite (comme dans la position standard), on obtient la répartition :

1	-1	1	-1
-1	1	-1	1
1	-1	1	-1
-1	1	-1	1*

* = position initiale

Ainsi, si la case vide effectue n déplacements élémentaires successifs, la valeur correspondant à la position atteinte sera $(-1)^n$. Par suite, la valeur associée à la position atteinte est 1 *ssi n* est pair (*i.e.* la case vide a effectué un nombre pair de déplacements élémentaires). D'où l'on peut conclure que si la case vide revient à sa position initiale, ce ne peut être qu'après un nombre pair de déplacements élémentaires, puisque la valeur correspondant à cette position est 1.

Maintenant, il est facile de voir que chaque déplacement élémentaire du taquin est une permutation de deux cases, l'une des deux cases étant la case vide. Toute configuration atteinte, avec la case vide revenue à sa position initiale, est donc une composition d'un nombre pair de transpositions, c'est-à-dire une permutation paire.

La démonstration de cette première implication fait un usage caché de la notion algébrique de *signature* d'une permutation. Le recours à l'algèbre est moins discret dans la preuve de la réciproque.

2. Si β est une permutation paire de α, alors β est accessible à partir de α.

Rappelons que, par hypothèse, la permutation laisse invariante la position de la case vide. On peut donc considérer qu'il s'agit d'une permutation des quinze carrés. Dans la mesure où elle est supposée paire, on s'intéresse au groupe alterné de degré 15, noté A_{15}, constitué des permutations paires du groupe symétrique de degré 15, S_{15}.

Précisons que chaque configuration du taquin peut s'écrire dans la notation classique pour

les permutations de S_{15} ; ainsi :

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \sigma(1) & \sigma(2) & \sigma(3) & \sigma(4) & \sigma(5) & \sigma(6) & \sigma(7) & \sigma(8) & \sigma(9) & \sigma(10) & \sigma(11) & \sigma(12) & \sigma(13) & \sigma(14) & \sigma(15) \end{pmatrix}$$

représente la configuration où la première case (en circulant de gauche à droite et de haut en bas) est occupée par le carré numéroté $\sigma(1)$, et de façon générale, la $i^{\text{ème}}$ case est occupée par le $\sigma(i)^{\text{ème}}$ carré ($1 \leq i \leq 15$). Il est donc tout à fait légitime de traduire la question sur le taquin par une question d'algèbre.

Il s'agit en fait de montrer que tous les éléments du groupe A_{15} peuvent être obtenus par des déplacements autorisés par les règles du taquin. La difficulté est de lier les déplacements « physiquement » possibles du jeu aux éléments de A_{15}. Dans cette perspective, un résultat classique d'algèbre est particulièrement utile :

Lemme : le groupe alterné de degré n est engendré par les cycles de longueur 3 de S_n.

Avec ce lemme, il ne reste plus qu'à démontrer que tous les 3-cycles peuvent être obtenus à partir des règles du taquin. C'est là une étape décisive, mais tout n'est pas encore réglé. Car comment montrer que tous les cycles de la forme (i j k) peuvent être ainsi obtenus ? La généralité même de cette forme ne rend pas aisée l'entreprise. Mais l'algèbre vient encore à notre secours : on peut montrer, en effet, que les 3-cycles (1 2 i), $1 \leq i \leq n$ suffisent à engendrer le groupe An. Dans le cas qui nous occupe, cela revient à procéder en deux étapes :

- On commence par montrer que tout 3-cycles (i j k) est un produit de 3-cycles de la forme plus particulière (1 2 h), que nous qualifierons de *canonique*. Cette étape est encore purement algébrique, et ne fait pas référence aux possibilités du taquin.

- Il reste, dans un second temps, à montrer que tous les 3-cycles canoniques peuvent être obtenus à partir des règles du taquin. Pour cela, on repère des 3-cycles qu'il est particulièrement aisé d'obtenir sur le taquin, à partir de la configuration initiale, et on montre que chaque (1 2 i), pour $3 \leq i \leq 15$, peut être obtenu à partir de ces 3-cycles. Ces derniers sont les 3-cycles « *naturels* », que l'on peut associer à chaque carré constitué de quatre cases sur la grille du taquin. On compte neuf de ces carrés. Appelons les les « carrés de quatre », pour ne pas les confondre avec les carrés élémentaires dont ils sont constitués. Pour obtenir un 3-cycle « naturel », il suffit de placer la case vide à l'intérieur du carré de quatre correspondant, de déplacer les trois carrés restants, et de remettre la case vide, ainsi que les autres carrés, dans leur position initiale. Par exemple, dans la position standard,

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

les carrés 1, 2, 5, 6 constituent un carré de quatre (appelons le [1; 2; 5; 6]). Pour construire le 3-cycle (1 2 6), on « enlève » le 5 de ce carré de quatre :

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

puis on place les trois carrés restants dans la position voulue :

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

Enfin, on remet les autres cases dans leur position de départ :

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

Supposons, maintenant, que l'on veuille obtenir (à partir de la position standard) le 3-cycle canonique (1 2 11). On devra commencer par mettre le carré 11 dans le carré de quatre [1; 2; 5; 6]. Pour cela, on supprime 10 du carré de quatre central [1; 7; 10; 11], puis on effectue les déplacements appropriés, de manière à obtenir :

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

Puis on enlève 5 du carré de quatre [1; 2; 5; 11] ainsi formé, et l'on déplace les carrés restants :

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

Il faut encore placer le carré 1 dans la position initiale du carré 11. Il suffit d'enlever 9 du carré de quatre central pour réaliser ce but :

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

Il ne reste plus qu'à replacer les autres carrés dans leur position initiale :

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

On se convaincra aisément que le même type de procédé peut être utilisé pour obtenir tous les 3-cycles canoniques (il y a treize vérifications à effectuer).

Cela achève la preuve du théorème.

Annexe 2: La Thèse de Church-Turing

Bien que ses développements l'aient conduit au-delà de ces premières intentions, la théorie de la récursion a pour but d'étudier les fonctions (*mécaniquement)* calculables. Les fonctions (partielles) calculables sont, d'un point de vue informel, des fonctions définies sur une partie de $\mathbb{N}^k$ ($k \in \mathbb{N}$), et dont les valeurs (dans $\mathbb{N}$) peuvent être déterminées par un algorithme (ou une procédure effective), c'est-à-dire un ensemble fini d'instructions, dont l'application n'exige aucune inventivité, et qui permettent d'obtenir la valeur cherchée (lorsqu'elle existe) en un nombre fini d'étapes. Plusieurs classes de fonctions, qui se sont avérées identiques, ont été formellement définies de manière à être calculables en ce sens « intuitif » : les fonctions λ-calculables, récursives, Turing-calculables, *etc*. Elles fournissent ainsi les meilleurs candidats pour donner une définition formelle de la calculabilité. L'affirmation selon laquelle *toute* fonction calculable appartient à cette classe est appelée *la thèse de Church*. Celle-ci ne peut évidemment être démontrée, dans la mesure où elle renferme une notion informelle : il faudrait disposer d'une définition précise de l'expression « fonction calculable » ; or la thèse de Church vise justement à fournir cette définition précise.

C'est le logicien Alonzo Church qui, peu avant Turing, en 1936, énonça une première version de la thèse qui porte désormais son nom. Il identifiait alors les fonctions calculables aux fonctions λ-calculables, celles qui peuvent être définies par un terme du λ-calcul, dont il est l'inventeur. Mais bien que (légèrement) postérieure, la version de Turing, s'est rapidement imposée. Elle consiste à définir la classe des fonctions calculables comme la classe des fonctions dont les valeurs peuvent être calculées par une « machine de Turing ». C'est cette affirmation, équivalente à la thèse de Church proprement dite, que l'on peut appeler la thèse de Church-Turing.

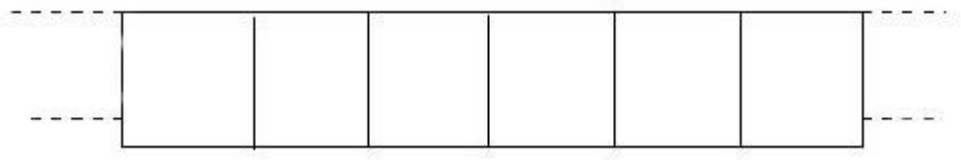
Pour expliciter cette dernière, il nous suffit donc de définir les fonctions Turing-calculables

(T-calculables), c'est-à-dire calculables par une *machine de Turing*. Il existe plusieurs manières de définir une telle machine (abstraite). N'ayant pas besoin, dans un premier temps, d'une définition plus générale, nous pouvons nous donner un alphabet constitué d'une infinité (dénombrable) de *symboles d'états internes* q_i ainsi que des symboles B, I (*les symboles d'entrée*) et R, L (*symboles de déplacement*). Un *quadruplet* est une expression de l'une des formes suivantes :

$$(a) \quad q_i S_j S_k q_l$$
$$(b) \quad q_i S_j R q_l$$
$$(c) \quad q_i S_j L q_l$$

où les S_i désignent des symboles d'entrée. Pour comprendre la signification d'un quadruplet, il faut concevoir la machine comme étant constituée d'un ruban infini divisé en une infinité de cases (Figure 1).

Figure 1



A chaque instant, une seule de ces cases est « observée », ou « lue », par la machine, et celle-ci peut y inscrire un I ou un B (ce qui revient à effacer le contenu de la case, car B fait ici office « d'espace »). Chaque quadruplet correspond, en fait, à une instruction, et la machine n'est rien d'autre qu'un ensemble de telles instructions. Au cours d'un calcul, elle peut être dans divers états. Une instruction du type (a) signifie que si la machine se trouve dans l'état q_i en observant S_j, elle doit remplacer ce symbole par S_k et entrer dans l'état q_l. L'instruction (b) commande à la machine de se déplacer d'une case à droite (« *Right* ») et d'entrer dans l'état q_l lorsqu'elle est dans l'état q_i en observant le symbole S_j. La commande (c) énonce l'instruction correspondante à gauche. Les seules actions élémentaires d'une machine de Turing ainsi définie consistent à écrire, ou à effacer, un symbole I dans une case du ruban, ou à se déplacer d'une case à gauche, ou à droite. De plus, on voit que, pour éviter qu'il y ait à choisir entre deux instructions possibles, une machine ne peut être n'importe quel ensemble de quadruplets : elle ne peut contenir deux quadruplets ayant les mêmes deux premiers symboles. C'est du moins le cas des machines dites *déterministes* ; mais il peut être utile d'introduire des algorithmes non déterministes.

Il nous faut maintenant préciser comment sont représentés les entiers pour la machine. On considérera que I représente zéro, et une suite de $n + 1$ symboles I consécutifs désignera le nombre n. Les arguments $x_0, \ldots, x_k$ d'une fonction définie sur une partie de $\mathbb{N}^k$ seront représentés sur le ruban par l'expression :

$$\overbrace{I \ldots I}^{(x_0+1)\text{ fois}} \; B \; \overbrace{I \ldots I}^{(x_1+1)\text{ fois}} \; B \ldots B \; \overbrace{I \ldots I}^{(x_k+1)\text{ fois}} \qquad (1)$$

Évidemment, toutes les autres cases sont vides, c'est-à-dire occupées par un B. On conviendra qu'au

début d'une exécution la machine est dans l'état q_0 et observe le symbole *I* le plus à gauche. On considère qu'elle a donné une réponse (*i.e.*, calculé une valeur) si elle s'arrête après un nombre fini d'étapes (aucune nouvelle instruction n'étant applicable) avec une suite finie ininterrompue de symboles *I* sur le ruban, la machine observant le plus à gauche d'entre eux. La notion d'exécution d'une machine de Turing peut être définie formellement avec précision, ce que nous ne ferons pas ici. Il nous suffira de dire qu'une exécution est une suite de descriptions du contenu du ruban, ou plutôt une description de ce contenu précisant également l'état dans lequel se trouve la machine, ainsi que le symbole observé (ce qui peut être donné en indiquant cet état devant le symbole observé) : l'exécution de *M*, lorsque celle-ci est appliquée à des arguments inscrits sur le ruban, sera la suite des descriptions du contenu du ruban après chaque application d'une instruction. Un *calcul* est une exécution finie. Ainsi, avec les conventions qui ont été données, on peut associer à chaque machine *M*, et chaque entier positif *k*, une fonction f_{Mk} qui pour (m_0, …m_k) prend la valeur *n* si l'exécution de *M* à (1) s'arrête dans les conditions décrites, et n'est pas définie sinon.

Pour illustrer de façon simple la manière dont fonctionne une machine de Turing, on peut donner l'exemple d'une machine calculant la somme de deux entiers. On se convaincra aisément que c'est le cas de :

$$M = \{ q_0IRq_0, q_0BIq_1, q_1ILq_1, q_1BRq_2, q_2IBq_3, q_3BRq_3, q_3IBq_4, q_4BRq_4 \}$$

En effet, le ruban portera initialement deux suites de *I* séparées par un *B* et la machine, dans l'état q_0, observant le plus à gauche des *I*, se déplacera vers la droite dans ce même état jusqu'à ce qu'elle rencontre le *B* séparant les deux suites. Elle le remplacera par *I*, entrera dans l'état q_1 dans lequel elle poursuivra en se déplaçant vers la gauche jusqu'à ce qu'elle rencontre un *B* à partir duquel, entrant dans l'état q_2, elle se déplacera d'une case à droite, et supprimera un *I* puis, dans l'état q_3 effacera un autre *I* et enfin, dans l'état q_4, se placera sur le premier *I* à gauche. Au total, la machine a effacé deux *I* après en avoir ajouté un (de manière à obtenir une suite ininterrompue), ce qui fournit effectivement le résultat cherché.

On peut voir la thèse de Church-Turing comme visant non seulement à définir la classe des fonctions calculables, mais aussi la notion même de procédure effective (ou algorithmes). Mais les choses sont un peu moins simples ici. Car lorsque l'on montre, par exemple, que la classe des fonctions T-calculables est exactement celle des fonctions λ-définissables, ou celle des fonctions récursives, ou celle des fonctions calculables par les algorithmes de Markov, on peut encore s'interroger sur ce que ces résultats nous enseignent sur la notion même d'algorithme ; celle-ci est-elle davantage « capturée » par les machines de Turing, les règles du λ-calcul, les opérations définissant les fonctions récursives, les « prescriptions » de Markov, *etc.* ? On voit qu'il y a donc deux manières d'interpréter la thèse. La première (définition d'une classe de fonctions) est corroborée chaque fois que l'on démontre l'identité de deux classes définies indépendamment ; la seconde ne peut être justifiée que par une analyse de la notion informelle d'algorithme de calcul – ce qu'avait précisément entrepris Turing dans l'article fondateur de 36.