monoff-Levin – $\log_2 SL(s)$ est qu'elle donne ce que nous attendons du classement des séquences courtes : ainsi, les trois séquences 0000000, 0001000 et 1001101 sont classées dans cet ordre. Pour les petites séquences, cette mesure probabiliste est stable et conforme à notre idée de la complexité ; pour les grandes, elle est, d'après le théorème mentionné, conforme à la meilleure mesure de complexité unanimement admise, la complexité de Kolmogorov. Que demander de plus ?

Le calcul de $SL(s)$ s'obtient en faisant fonctionner un très grand nombre de programmes, qui seront soit tirés au hasard, soit pris dans un ensemble de programmes énumérés systématiquement (ce qui revient au même). En cumulant les résultats, on obtient expérimentalement un calcul approché de la distribution $SL(s)$, donc des nombres $-\log_2 SL(s)$ donnant la complexité des petites séquences. Le procédé est le même que celui qu'on appliquerait pour évaluer si une loterie est truquée ou non : on lancerait la loterie un très grand nombre de fois, et, à partir des résultats obtenus, on observerait, par exemple, que le 13 sort plus souvent que les autres numéros. À l'aide d'un grand nombre de tirages, on évaluerait même la probabilité de sortie de chaque numéro et bien sûr, plus le nombre de tirages expérimentaux serait grand, meilleure serait la précision obtenue pour les probabilités de chaque numéro.

On comprend cependant les difficultés de la méthode. Pour évaluer expérimentalement les probabilités d'une loterie à 20 numéros, il faut faire des milliers de tirages. Pour évaluer le nombre $SL(s)$ pour les 128 séquences binaires de sept chiffres ou les 1024 séquences binaires de longueur dix, il faut faire un nombre de tirages encore plus massif, et la limite de ce qu'on peut envisager pratiquement est rapidement atteinte, quelle que soit la puissance informatique dont on dispose.

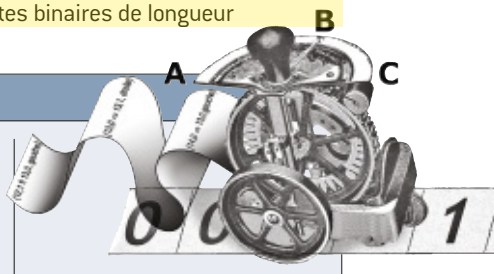
Les machines de Turing à la rescousse

Comment H. Zenil a-t-il procédé ? Il a utilisé une machine à calculer aussi primitive que possible, mais suffisamment puissante pour que tout programme puisse y être exécuté. Ces « machines de Turing », introduites par Alan Turing en 1936, jouent depuis un rôle fondamental en informatique théorique et en logique mathématique, car on n'a pas su faire plus simple et plus général pour étudier la notion d'algorithme. Ces machines constituent un langage de programmation, parce que décrire une machine de Turing est équivalent à écrire un programme informatique. Passer par ces machines est intéressant pour le calcul de $SL(s)$, car elles permettent une programmation d'une redoutable efficacité : des programmes courts mènent des calculs non triviaux *(voir la description d'une machine de Turing sur la figure 3)*.

Le nombre d'états d'une machine de Turing détermine son pouvoir de calcul. Les machines à un état ne peuvent faire que des calculs simples, par exemple inverser les 0 en 1 et les 1 en 0 d'une séquence qu'on leur soumet.

Les machines à deux états sont déjà plus intéressantes. Leur nombre est 10 000 (les formules compliquées de dénombrements de ces machines donnent ce résultat et c'est un hasard qu'il soit un chiffre aussi rond), ce qui bien sûr ne peut donner accès qu'à une vue extrêmement sommaire de $SL(s)$. En faisant fonctionner toutes les machines de Turing à trois états, les choses deviennent plus sérieuses. Ces 7 529 526 machines produisent une approximation de $SL(s)$ qui n'est pas absurde et qui montre qu'on est sur la bonne voie. Par exemple, parmi les suites binaires de longueur

3. La machine de Turing



Une machine de Turing est un mécanisme de calcul qui peut se trouver dans une série d'états internes (ici trois états notés A, B, C). La machine possède une tête de lecture-écriture qui se déplace et modifie une à une les cases d'un ruban infini (une sorte de mémoire externe). En fonction de l'état interne et de ce que la tête de lecture-écriture observe sur le ruban, la machine décide (en suivant des règles constituant le programme de la machine et fixées une fois pour toutes) de changer ce qui est écrit dans la case sous sa tête de lecture-écriture et de se déplacer vers la droite ou vers la gauche d'une unité en passant dans un autre état interne. Voici une machine à trois états A, B, C possédant un programme de quatre instructions :

(A, 0 → B, 1, droite)
(B, 0 → C, 1, gauche)
(C, 1 → B, 1, droite)
(B, 1 → arrêt)

L'instruction (A, 0 → B, 1, droite) signifie que lorsque la machine est dans l'état A et qu'elle lit un 0, alors elle passe dans l'état B en remplaçant le 0 par un 1 et se déplace d'une case vers la droite. Lorsque l'état dans lequel la machine passe est l'état *arrêt*, elle cesse toute action. La machine donnée en exemple opère un calcul élémentaire : placée sur un ruban infini couvert de 0, elle mène quelques opérations d'écriture et de déplacements et s'arrête en laissant ...001100... Cette machine produit la séquence $s = 11$. Voyons le détail des calculs.

A
.....0 0 0 0 0 0 0 0 ...

La machine, dans l'état A, lit un 0. L'instruction (A, 0 → B, 1, droite) lui indique qu'elle doit changer le 0 en 1, passer dans l'état interne B et se déplacer vers la droite :

B
... 0 0 0 1 0 0 0 0 ...

L'instruction (B, 0 → C, 1, gauche) lui fait écrire un second 1 et la conduit dans l'état C :

C
... 0 0 0 1 1 0 0 0 ...

L'instruction (C, 1 → B, 1, droite) donne maintenant :

B
... 0 0 0 1 1 0 0 0 ...

L'instruction (B, 1 → arrêt) fait cesser toute activité.

La machine a produit la séquence $s = 11$ sur les cases qu'elle a modifiées. Dans l'évaluation, que l'on fait avec toutes les machines de Turing à trois états, des complexités des suites à deux éléments, cette machine augmentera la probabilité de la séquence binaire 11.