Master MCSC 1

Calculabilité

YOUSSEF SBAYTRI

Plan

Calculabilité

I- Les Automates Cellulaires

1. La naissance des automates cellulaires

1.1 L'idée de départ de Von Neumann et d'Ulam

Les automates cellulaires ont été inventés par **Stanislaw Ulam** (1909-1984) **et John Von Neumann** (1903-1957) à la fin des années 40 au **Los Alamos National Laboratory**.

Von Neumann est indiscutablement un grand génie du XX^{e} siècle, bien que ses travaux ne soient pas très connus du grand public. Le nombre de domaines auxquels il apporta des contributions décisives ne peut que laisser admiratif. Il fut un des pionniers dans la conception des ordinateurs, sa « Théorie des jeux » est un outil toujours utilisé par les décideurs dans les domaines économiques et militaires.
En 1948, Von Neumann a proposé un article intitulé « Théorie générale et logique des automates » dans une conférence tenue à Pasadena, en Californie. En 1949, il donna une série de cours sur le thème : « Théorie et organisation des automates complexes ». **Une des questions centrales de ce cours était de savoir s'il était possible de concevoir une machine capable de s'auto-reproduire.** En effet, il est clair que les objets fabriqués par une machine sont généralement plus simples que la machine elle-même. Prenons l'exemple d'une usine de fabrication de bouteilles de soda, on ne contestera pas dans ce cas que la bouteille est plus simple que la machine qui l'a fabriquée. Même dans le cas d'une usine de fabrication d'ordinateurs, l'outillage utilisé est bien plus complexe que le produit fabriqué.
Von Neumann émit l'idée qu'une machine capable de manipuler des composants de machine élémentaires pourrait résoudre ce problème. Dans sa première conception, l'automate devait puiser dans un réservoir de composants de machine et construire un automate similaire à lui-même, à la façon dont on construit un automate par un jeu de « Meccano ». Mais cela nécessitait que l'automate soit doté d'un système de vision et de reconnaissance suffisamment élaboré pour pouvoir distinguer les différents éléments de machine. Cet automate aurait dû être en outre doté de capacités exceptionnelles pour pouvoir souder et manipuler des tubes à vide sans les abîmer. Ces exigences étaient tout simplement inconcevables étant donné l'état des techniques dans les années 50.
La solution à ce problème vint d'un collègue de Von Neumann au laboratoire de Los Alamos : Stanislaw Ulam. Elève de Banach (1892-1945), influencé par les lectures de Sierpinsky (1882-1969) ; Ulam est l'auteur d'un problème simple non encore résolu à ce jour : « prenez un nombre, si ce nombre est pair divisez le par 2, s'il est impair multipliez par trois et ajoutez un, vous obtenez une suite de nombres, toute suite finit-elle par converger vers le cycle 4/2/1 quel que soit le terme initial choisi ? » Il est notamment connu pour être l'inventeur de la méthode Monte-Carlo et pour ses travaux sur la fusion nucléaire. Ulam s'intéressait aux « objets géométriques définis de façon récursive » qu'il étudiait après les heures réglementaires de travail en utilisant les ordinateurs du laboratoire de Los Alamos. Ces objets provenaient de jeux aux règles simples dans lesquels on pouvait voir des figures se développer, se reproduire, combattre pour une portion de territoire et mourir. Ces jeux se déroulaient dans un univers « cellulaire », composé d'une matrice infinie où les cellules, régulièrement réparties, peuvent être dans un état passif ou un état actif. Les figures de cet univers étaient composées des cellules actives, et à tout moment, le devenir de chaque cellule était dicté par l'état des cellules avoisinantes. Ulam s'aperçut que l'analyse de ces figures défiait le bon sens : elles semblaient évoluer dans un monde qui leur était propre avec des lois bien spécifiques. Il suggéra alors à Von Neumann d'utiliser un tel monde abstrait pour pallier les difficultés pratiques qui se posaient pour la construction de l'automate autoreproducteur. Ce monde serait suffisamment complexe pour pouvoir simuler les opérations élémentaires des machines et en même temps construit de façon à ce que les lois de la physique qui gouvernent ce monde se réduisent à quelques règles simples. L'idée plut à Von Neumann qui était habitué à voir les machines comme des **circuits logiques**, il adopta donc l'univers d'Ulam pour commencer à réfléchir à son automate

2. Les automates cellulaires élémentaires

2.1 Définition des automates cellulaires

Un automate cellulaire peut être vu comme une infinité de petites machines identiques : les cellules, réparties sur un graphe régulier (espace discret). Les cellules n'ont qu'une mémoire finie, c'est-à-dire qu'elles ne peuvent être que dans un nombre fini d'états possibles et qu'elles ne portent pas d'autre information que leur état. À chaque étape de temps, chaque cellule change son état courant en fonction de son état et de celui de ses voisines uniquement. Formellement, on a la définition suivante :
Un automate cellulaire (**AC**) est un 4-uplet **A = (S, d, N, f)** où :
• **d** est la dimension de l'automate, son réseau est alors $\boldsymbol{Z^d}$, l'espace discret de dimension d.
• **N** est l'ensemble des voisinages.
• **S** est l'ensemble des états.
• **f** est la fonction de transition **: f :** $\boldsymbol{S^{|N|}}$ → **S,** avec |N| le cardinal de N.

2.2 La fonction de transition

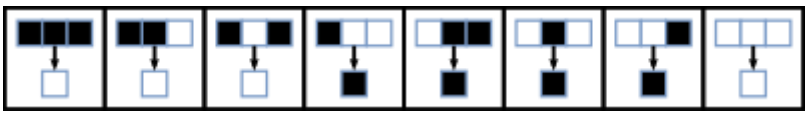
La fonction de transition locale **f :** $\boldsymbol{S^{|N|}}$ → **S,** $\boldsymbol{S^{|N|}}$ représente l'ensemble de tous les états possibles que le voisinage peut avoir, Chaque valeur de $\boldsymbol{S^{|N|}}$ a la forme $(s_0, s_1, \dots \dots s_{|N|-1})$ avec $s_i \in S$.
Exemple : Automate cellulaire simple

Il s'agit d'un automate **A =(S,*1*,*N*,f)** où :

- **S = {0,1}.**
- **N** : Les voisins d'une cellule i sont les cellules : **i, i-1 et i+1** Pour simplifier ce voisinage peut être décrit par le triplet N = {-1,0,1}.
- **f** : La fonction de transition qui est définie par le tableau suivant :

$\boldsymbol{(q_1, q_2, q_3)}$ Voisinage possible de la cellule centrale	111	110	101	100	011	010	001	000
$\boldsymbol{f(q_1, q_2, q_3)}$ Valeur suivante de la cellule centrale	0	0	0	1	1	1	1	0

On peut aussi représenter la fonction de transition par le graphe suivant :



Dans ce cas la fonction de transition peut être représenté sous forme d'une valeur binaire **00011110** qui est **30,** ou on peut la nommer aussi « la règle 30 ».
Cela signifie que si par exemple, à un temps *t* donné, une cellule est à l'état « 1 », sa voisine de gauche à l'état « 1 » et sa voisine de droite à l'état « 0 », au temps *t+1* elle sera à l'état « 0 ».

Il s'agit d'une grille unidimensionnelle de cellules ne pouvant prendre que deux états (« 0 » ou « 1 »), avec un voisinage constitué, pour chaque cellule, d'elle-même et des deux cellules qui lui sont adjacentes.
Chacune des cellules pouvant prendre deux états, il existe $\boldsymbol{2^3 = 8}$ configurations possibles d'un tel voisinage. Pour que l'automate cellulaire fonctionne, il faut définir quel doit être l'état, à la génération suivante, d'une cellule pour chacun de ces motifs. Il y a $\boldsymbol{2^8 = 256}$ façons différentes de s'y prendre, soit donc 256 automates cellulaires différents de ce type.

Exemple :

Simuler un automate cellulaire consiste à tracer sur un écran graphique l'état de l'automate à des dates successives.

Considérons l'automate à une dimension, formé de 11 cellules, dans l'état initial suivant :

0	0	0	0	1	0	0	0	0

Supposons que sa fonction de transitions soit celle donnée plus haut. Alors, nous pouvons simuler l'automate en représentant sont état initial sur la première ligne, l'état suivant sur la seconde ligne etc… Voici les 5 premiers états de l'automate.

t=0	0	0	0	0	1	0	0	0	0
t=1	0	0	0	1	1	1	0	0	0
t=2	0	0	1	1	0	0	1	0	0
t=3	0	1	1	0	1	1	1	1	0
t=4	1	1	0	0	1	0	0	0	1
t=5	0	0	1	1	1	1	0	1	1

2.2 La Classification des AC

Pour étudier la classification des ACs Wolfram a proposé dans son article « Statistical Mechanics of Cellular Automata » une première classification des automates cellulaires selon leur comportement dynamique. Cette classification comporte quatre classes. Les trois premières sont inspirées des catégories qui apparaissent dans l'étude des systèmes dynamiques, la quatrième étant la plus intéressante puisque spécifique au domaine des AC.

classe	**attracteur**	**dynamique**
classe I	Points limites	Après un certain nombre de cycles, l'automate tend à atteindre un état unique partant de configurations initiales différentes.
classe II	Cycles limites	L'automate aboutit à une phase de répétition périodique des états.
classe III	Attracteur étrange et comportement chaotique	A partir de la majorité des états initiaux, ce type d'automate mène à des figures chaotiques, non périodiques. Pour des automates 1D on reconnaît clairement des figures fractales telles que le triangle de Sierpinski.
classe IV	comportement plus complexe	Après un certain nombre de cycles, ce type d'automates se place dans un état « mort ». Néanmoins, un petit nombre de figures stables peuvent subsister comme dans le Jeu de la Vie.

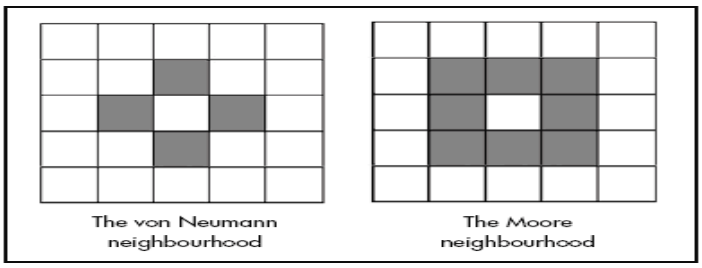
Le problème de cette classification est que la classe d'un automate ne peut être déterminée qu'a posteriori, c'est-à-dire par simulation et observation du comportement. Ceci implique que la classification n'apportera d'information que dans la mesure où elle peut être reliée aux caractéristiques fonctionnelles de l'automate cellulaire.

Les frontières entre les différentes classes sont définies de façon qualitative. Comment, par exemple, savoir si l'on est en présence d'un automate de classe IV ou de classe III ? Jusqu'à présent, c'est généralement en observant le comportement des automates et en décidant « à vue d'œil » que les chercheurs ont classé les automates. Il n'existe pas à ce jour d'algorithme qui tranche entre les classes. Wolfram a émis l'hypothèse qu'une propriété des automates de la classe IV est de réaliser la calculabilité universelle. Dans ce cas, des résultats d'indécidabilité ont été obtenus : il serait alors impossible d'arriver à trouver une méthode systématique pour décider de la classe d'un AC. La pertinence de cette classification a donc été largement discutée dans la communauté des chercheurs qui travaillent sur les AC. De nombreuses autres propositions ont depuis été émises, néanmoins aucune d'entre elles ne s'est imposée de façon définitive et le problème de la classification des automates cellulaires reste posé à ce jour.

3. Les automates cellulaires à deux dimensions

3.1 Définition

Les automates cellulaires à deux dimensions (bidimensionnels) contiennent plusieurs caractéristiques des automates élémentaires. Il existe deux types fondamentaux de voisinage. D'abord il y a le voisinage de Von Neumann qui est composé de quatre cellules adjacentes aux côtés de la cellule centrale, et le voisinage Moore dont huit cellules adjacentes à la cellule centrale.



3.2 Jeu de la vie

Le Jeu de la Vie est une bonne illustration d'un automate cellulaire simple. Créé par John Horton Conway en 1970, ce jeu a connu un grand succès du fait de sa simplicité, sa mise en œuvre informatique aisée et les développements variés auxquels il a donné lieu.

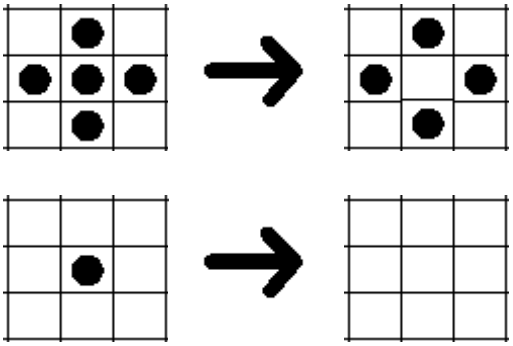
Les règles sont les suivantes :

Il se joue sur un damier infini, sur les cases duquel sont disposés des pions représentant les cellules d'un organisme. Ces cellules naissent, meurent ou survivent à chaque génération selon des règles bien précises. Conway a choisi ces règles après de nombreuses expériences, de sorte que l'évolution de ces organismes soit assez imprévisible, notamment en ce qui concerne leur disparition, leur caractère périodique ou leur croissance infinie.

- Survie
Toute cellule ayant exactement deux ou trois cellules voisines survit à la génération suivante.

- Mort

Toute cellule ayant quatre cellules voisines ou plus meurt par étouffement à la génération suivante. Une cellule isolée ou n'ayant qu'un seule cellule voisine meurt d'isolement à la génération suivante.



- Naissance

Sur une case vide comportant exactement trois cellules voisines, il naît une cellule à la génération suivante.

On peut faire deux remarques importantes :

- les cases peuvent donc avoir deux états : vivants ou mort.
- Une case tient compte de ses huit voisines pour déterminer son état à la génération suivante

3.3 Les voisinages de Margolus

L'un des voisinages les plus simples et les plus utilisées dans les automates cellulaires après voisinage de Von Neumann et Moore est les voisinages de Margolus, du nom de Norman Margolus, et il a étudié dans un livre qu'il a co-écrit, « Cellular Automata Machines (CAM). »

L'idée consiste à diviser une grille de cellules en groupes de bloc de dimension (2x2), auxquels la règle est appliquée complètement localement, alors l'information serait incapable de se propager au-delà des limites d'une partition individuelle - et la dynamique du système global serait stérile.

On note une règle de Margolus de cette forme : {n1;n2;n3;..;n16}
avec n1,n2…n16 représentent les transition de toutes les configurations de voisinage possibles, Il existe 16 configuration de voisinage Margolus numérotées de 0 à 15 :

0:  , 1:  , 2:  , 3:  , 4:  , 5:  , 6:  , 7:  , 8:  , 9:  , 10:  , 11:  , 12:  , 13:  , 14:  , 15: 

Example : Règle {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}

Si on fait la permutation de différents éléments de ces règles on aura un nombre de règle égale à **16 !** = 2.092279e+13 règles.

La propriété importante des règles de Margolus est qu'elles sont réversibles et facile à implémenter.

Example de certaines règles de Margolus et leurs descriptions :

Name	**Rule**	**Description**
BounceGas	{0;8;4;3;2;5;9;14; 1;6;10;13;12;11;7;15}	A uniform "gas". A rule by Tim Tyler.
Critters	{15;14;13;3;11;5; 6;1;7;9;10;2;12;4;8;0}	This rule supports "gliders" - as described in Cellular Automata Machines. A rule by Margolus/Toffoli.
HPP_Gas	{0;8;4;12;2;10;9; 14;1;6;5;13;3;11;7;15}	HPP (Hardy/Pazzis/Pomeau) lattice gas - as described in Cellular Automata Machines. A rule by Hardy, Pazzis, and Pomeau.
Rotations	{0;2;8;12;1;10;9; 11;4;6;5;14;3;7;13;15}	Limited diffusion. A rule by Tim Tyler.
StringThing	{0;1;2;12;4;10;9;7;8; 6;5;11;3;13;14;15}	String shaped patterns. A rule by Tim Tyler.
SwapOnDiag	{0;8;4;12;2;10;6;14; 1;9;5;13;3;11;7;15}	A gas with no particle interactions - as described in Cellular Automata Machines. A rule by Margolus/Toffoli.
Tron	{15;1;2;3;4;5;6;7;8; 9;10;11;12;13;14;0}	A "trip-a-tron" - from the pages of Cellular Automata Machines. A rule by Margolus/Toffoli.

4. Réversibilité des automates cellulaires

4.1 automates cellulaires réversibles

Un AC et dit réversible si l'AC reviendra toujours à son état initial, La propriété intéressante d'être réversible signifie que l'AC peut s'évoluer en avant et en arrière, En utilisant une règle réversible, il est possible de revenir à un état antérieur à n'importe quel moment, Ce type de l'AC et utilisé en cryptographie.

Dans le cas des AC élémentaires, parmi les 256 règles seul six règles qui sont réversibles.

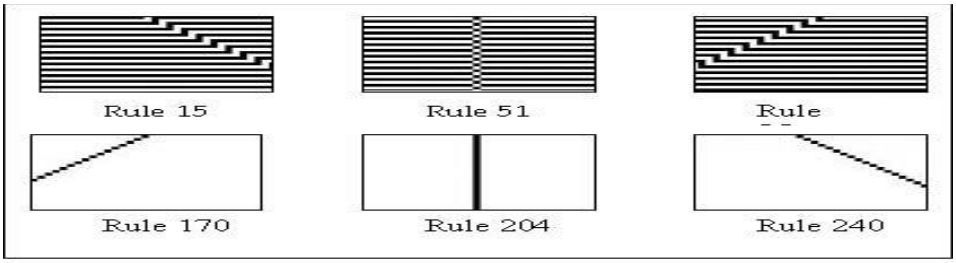


Figure 1 Rule 15, 51, 85, 170, 204, and 240

Exemple de règle réversible.

Soit **la règle 15** quand la représente par la chaine binaire **00001111**, et on a la chaine binaire **11010011,** En appliquant une transition avec la règle 15 on aura le résultat suivant :

t=0	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
t=1	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**
t=2	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**0**
t=3	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
t=4	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**
t=5	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**
t=6	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**
t=7	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**
t=8	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**

NB : A l'instant t=8 l'AC reviendra à son état initial, dans ce cas la règle 15 est réversible.

5 Application des automates cellulaires

Les applications pratiques des automates cellulaires sont nombreuses et diverses. Fondamentalement ils constituent des univers dont on fixe les lois. Notre Univers est soumis aux lois de la Physique. Ces lois ne sont que partiellement connues et apparaissent hautement complexes. Dans un automate cellulaire, les lois sont simples et complètement connues. On peut ainsi tester et analyser le comportement global d'un univers simplifié. Voici quelques exemples d'application :

- En cryptographie : Exploitation des propriétés de réversibilité des automates cellulaires à la conception des nouveaux crypto-systèmes, génération des nombres/chaines aléatoires, création des boites de substitution S-BOX, …etc.

- Machine Learning : Utilisation des automates cellulaires pour définir des nouveaux méthodes de classification (exemple : Petra Povalej et al « Machine-Learning with Cellular Automata », https://doi.org/10.1007/11552253_28)
- Deep Learning : utilisation des automates cellulaires comme réseaux de neurones convolutifs (William Gilpin « Cellular automata as convolutional neural networks », https://10.1103/PhysRevE.100.032402), exploitation des automates cellulaires comme un réservoir de systèmes dynamiques (Stefano Nichele et Andreas Molund «Deep Learning with Cellular AutomatonBased Reservoir Computing », https://doi.org/10.25088/ComplexSystems.26.4.319).
- Conception d'ordinateurs massivement parallèles.
- Simulation et étude du développement urbain.
- Simulation de la propagation des feux de forêt.
- Simulation du comportement d'un gaz. Un gaz est composé d'un ensemble de molécules dont le comportement est fonction de celui des molécules voisines.
- Étude des matériaux magnétiques selon le modèle d'Ising.

II- La machine de Turing

1. Introduction

L'objectif de ce chapitre est de définir la notion de calculable et de montrer que les réponses à certaines questions ne sont pas calculables. Cette notion va être introduite par l'intermédiaire des machines de Turing qui fournissent un modèle de calcul abstrait pas trop éloigne du fonctionnement d'un ordinateur. En 1936, le célèbre mathématicien britannique Alan Turing (1912-1954) publie l'article fondateur de la science informatique. Il démontre dans cet article qu'il n'existe pas de méthode de calcul générale qui permette de décider si une proposition mathématique est démontrable. Pour établir sa démonstration il conçoit sur le papier les plans d'une machine qu'on appellera désormais la machine de Turing. Il appelle alors méthode de calcul une méthode qui peut être mise en œuvre sur sa machine et il démontre que sa machine est incapable de décider si une proposition mathématique est démontrable.

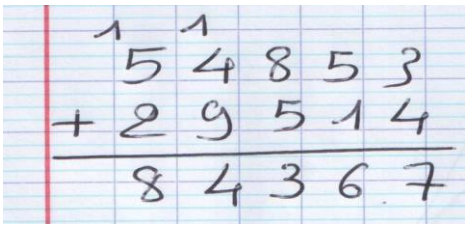
Une question se pose immédiatement : les capacités de la machine de Turing permettent-elles de mettre en œuvre toutes les méthodes de calcul susceptibles d'être imaginées ? Si la réponse est non cela diminue considérablement la portée du résultat de Turing. Mais en fait :

- Tout ce qui peut être calculé sur un ordinateur actuel peut aussi être calculé sur une machine de Turing (et réciproquement).
- Personne n'a pu encore proposer une méthode de calcul qui ne puisse pas être mis en œuvre sur une machine de Turing.

Alonzo Church a proposé la thèse selon laquelle la machine de Turing capture complètement toute la notion de calculabilité. D'autres chercheurs ont proposés d'autres définitions de la calculabilité. Ces autres définitions ont été démontrées équivalentes à la calculabilité par la machine de Turing. Remarquez bien que la thèse de Church n'est pas une proposition mathématique. Elle n'est donc pas démontrable et ne sera donc jamais établie.

La machine de Turing est extrêmement rudimentaire et il est tout à fait remarquable qu'elle ait exactement les mêmes capacités qu'un ordinateur moderne. D'une certaine façon cette machine est dépouillée de tout ce qui n'est pas nécessaire et met en lumière l'essence même de ce qu'est un calcul.

Calculez 54853+ 29514 en appliquant la méthode de calcul que vous avez apprise à l'école primaire.
C'est en s'observant lui-même, effectuant ce genre de calcul, qu'Alan Turing a inventé sa machine.



Les objets sur lesquels on veut effectuer un calcul (ici deux nombres) doivent tout d'abord être représentés par des suites de symboles (ici des chiffres).

• Les symboles sont disposés sur une feuille selon une règle bien précise.

Remarquez que la feuille est en deux dimensions.

• Calculer consiste alors à écrire de nouveaux symboles sur la feuille en appliquant de façon systématique les règles précises que l'on a apprises.

Si on étudie plus attentivement le processus on peut observer que :

• Le processus peut être découpé en un nombre fini d'étapes élémentaires.

• On ne peut observer à chaque instant qu'un seul symbole à la fois.

• L'action que l'on effectue dépend uniquement du symbole que l'on observe actuellement et de l'état mental dans lequel on se trouve. Notre état mental est certe complexe mais si on n'en conserve que ce qui est nécessaire pour effectuer une addition il se simplifie considérablement.
Pour effectuer l'addition de (54853 + 29514), lorsqu'on lit le chiffre des unités de 29514 la seule chose qui importe dans notre état mental est le souvenir du chiffre des unités de 54853. Un nombre fini d'états mentaux différents est suffisant pour mener à bien le calcul.

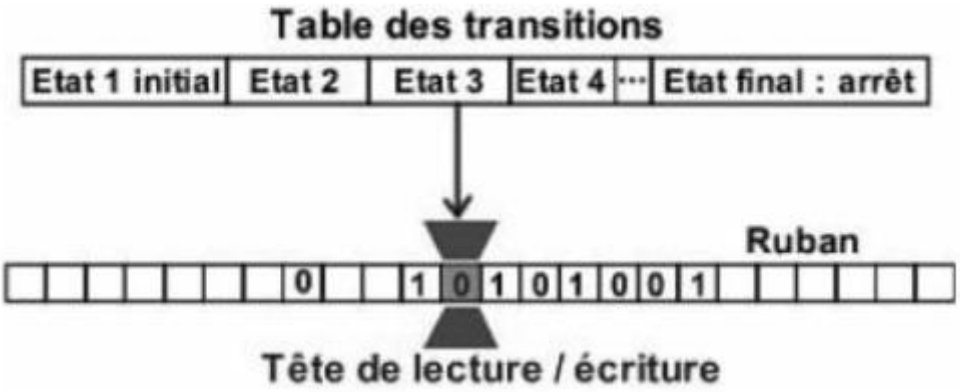
2. Description théorique de la machine

La machine d'Alan Turing est conçue sur ces principes, en les simplifiant au maximum :

- La feuille de papier en deux dimensions est remplacée par un ruban (à une dimension). Ce ruban est découpé en cases.
- Le ruban contient initialement une suite finie de symboles binaires 0 ou 1. Cette suite est le codage de l'objet sur lequel va porter le calcul.
- La machine dispose d'une tête de lecture/écriture qui est initialement placée juste à gauche de la suite de symboles.
- À chaque instant la machine ne peut lire que la case située sous sa tête de lecture.
- Le nombre d'états possibles pour la machine est fini. Initialement elle se trouve dans l'état 1. Les autres états sont numérotés 2, 3, 4, etc. Le dernier état est appelé FINAL.
- Lorsque la machine arrive dans cet état elle s'arrête.
- À chaque étape la machine lit le symbole situé sous sa tête de lecture. Selon le symbole lu et selon l'état dans lequel elle se trouve elle accomplit.

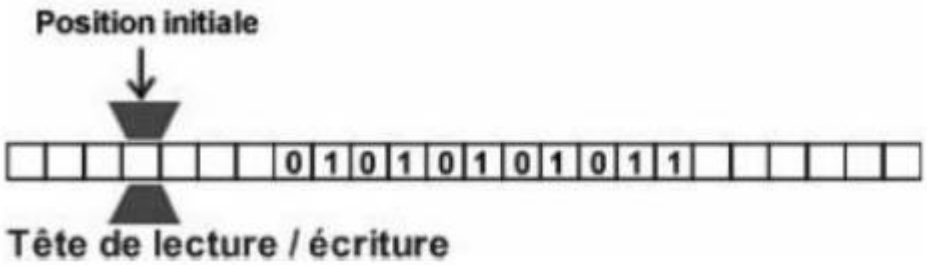
Mathématiquement on définit la machine de Turing sous la forme d'une **septuplet (Q,Γ,Σ,δ,q0,B,F)** où :

- Q : ensemble fini d'états
- Γ : alphabet fini des symboles de ruban
- Σ ⊆ Γ : alphabet fini des symboles d'entrée
- B ∈ Γ\ Σ : symbole particulier dit « blanc »
- q0 ∈ Q : état initial
- F ⊆ Q : ensemble des états d'acceptation
- δ : relation de transition



2.1 Exemple

Trouver une séquence et en remplacer les 0 par des 1, Une séquence de 0 et de 1 est écrite sur le ruban et la tête est à gauche de cette séquence.



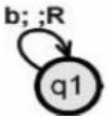
Pour cet algorithme, on utilise deux états :

- À l'état 1 : la tête devra se déplacer à droite tant qu'elle lira un blanc, dès qu'elle trouve un 0 ou un 1 elle passe à l'état 2.
- À l'état 2 : si elle lit un 0, elle écrit 1 et se déplace à droite ;
  - ✓ si elle lit un 1, elle n'écrit rien et se déplace à droite ;
  - ✓ enfin, si elle lit un blanc, c'est qu'elle a fini de parcourir la séquence, elle passe à l'état final et s'arrête.

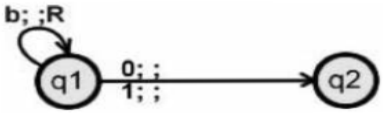
On construit le diagramme de la machine de la façon suivante :
Les états sont écrits sous la forme q1, q2, etc. et qF représente l'état final.

Les déplacements à gauche et à droite sont représentés par les lettres L et R (Left et Right), Les triplets indiqués représentent (lecture ; écriture ; déplacement) et la flèche indique le prochain état.

Dans l'état 1 (q1) :
Le triplet (**b ; ; R**) avec la flèche signifie que si la tête lit un blanc, elle n'écrit rien, elle se déplace d'une case à droite et elle reste à l'état 1.



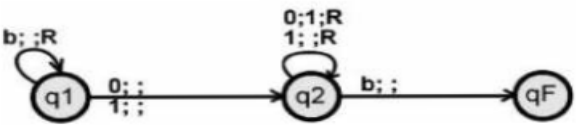
Les triplets (**0 ; ;**) et (**1 ; ;**) signifient que si la tête lit un **0** ou un **1**, elle n'écrit rien, elle ne se déplace pas et elle passe à l'état **2.**



Dans l'état 2 (q2) :

- Le triplet (**0 ; 1 ; R**) signifie que si la tête lit un 0, elle écrit 1, elle se déplace à droite et elle reste à l'état 2.
- Le triplet (**1 ; ; R**) signifie que si la tête lit un 1, elle n'écrit rien, elle se déplace à droite et elle reste à l'état 2.
- Le triplet (**b ; ;**) signifie que si la tête lit un blanc, elle n'écrit rien, elle ne se déplace pas et elle passe à l'état final **qF**.

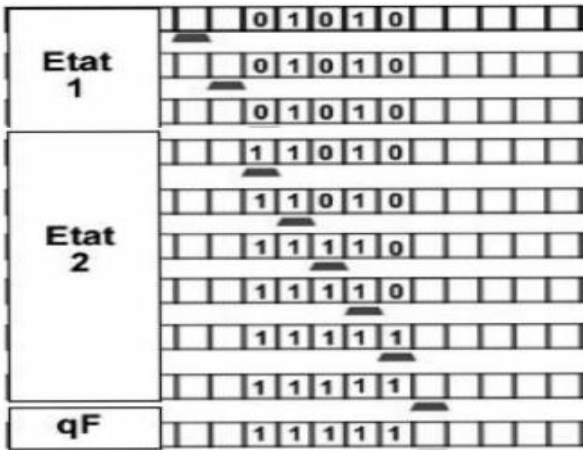
Le diagramme complet est le suivant :



Et voilà le résultat :

2.2 Représentation de la machine en fonctionnement.

Le trapèze représente la tête de lecture/écriture. On voit clairement ici que la tête de lecture ne fait que se déplacer de la gauche vers la droite (ce n'est pas le cas général) et que dans l'état 2 elle remplace les 0 par des 1. L'état final qF provoque l'arrêt de la machine.

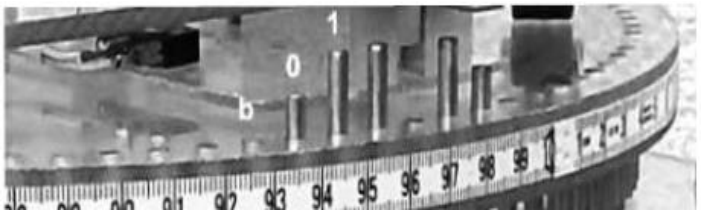


3. Description du prototype

Le ruban

Il comporte 100 petits cylindres avec 3 positions, alphabet {**b,0,1**} :

- **b** (blanc) : le cylindre est enfoncé totalement
- **0** : le cylindre est à mi-hauteur (7mm)
- **1** : le cylindre est relevé à pleine hauteur (14mm)



À noter : Le ruban est ici « replié sur lui-même » sous la forme d'un disque. Si le nombre de cylindre est fini (100), le ruban est quand à lui illimité.

Les états

Le commutateur des états comprend 12 états, de l'état initial 1 à l'état final 12 qui provoque l'arrêt de la machine.

Ce mécanisme retourne, pour chaque état, le résultat de la lecture à la table des transitions.

Dans la machine théorique, le nombre d'états est fini.

Mécanisme de lecture

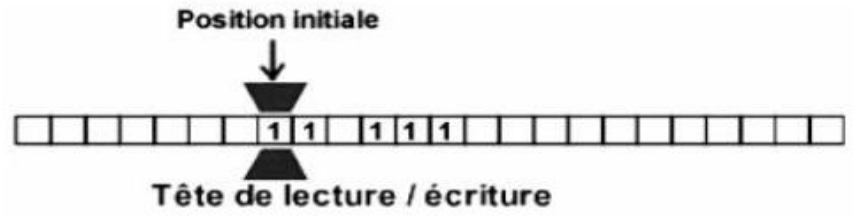
Un petit moteur pousse une tige contre les cylindres, la position d'arrêt indique la hauteur des cylindres.

Les relais permettent de garder en mémoire le résultat d'une lecture jusqu'à la suivante.

4. Quelques Exemples à étudier

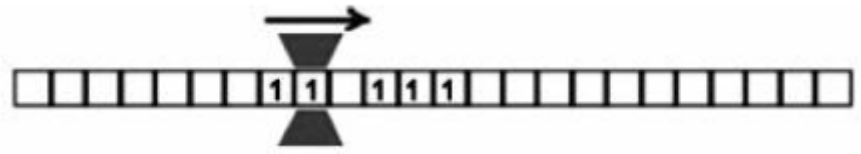
4.1 Faire une addition en unaire

Écrire un nombre en unaire revient à écrire autant de 1 que la valeur du nombre, exemple 5 s'écrit 11111. Il faut additionner deux nombres représentés par des 1 (dans cet exemple 2 et 3), et séparés par un espace blanc.

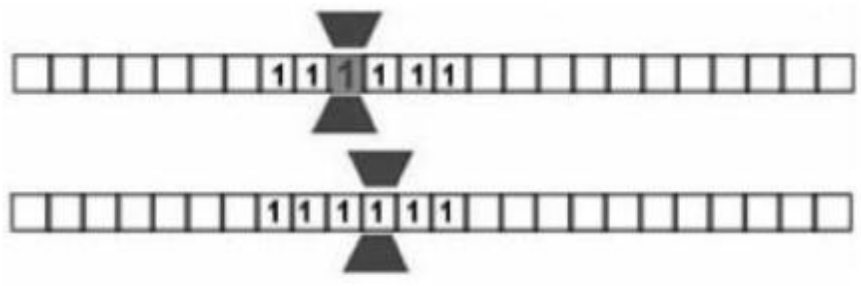


Pour cela la tête de lecture va parcourir le premier nombre, arrivée sur le blanc, elle va écrire un 1, puis elle va parcourir le deuxième nombre, arrivée au blanc elle va reculer d'une case et va terminer en mettant à blanc le 1 le plus à droite.

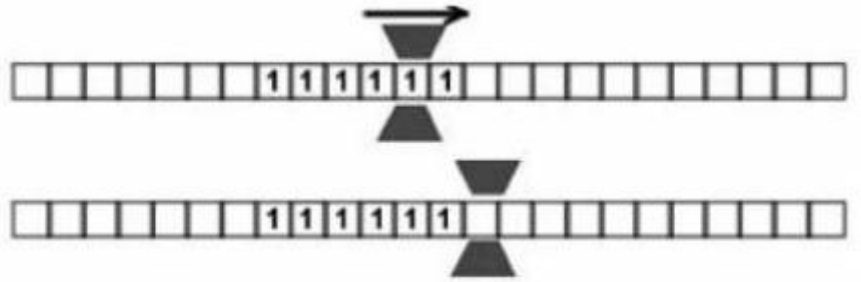
État 1 : la tête se déplace vers la droite jusqu'à trouver un blanc.



Quand elle trouve un blanc, elle écrit 1, puis se déplace à droite et passe à l'état 2



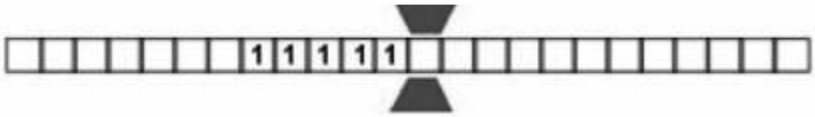
État 2 : la tête se déplace de nouveau vers la droite jusqu'à trouver un blanc



Quand elle a trouvé un blanc, elle recule d'une case pour se mettre sur le **1** le plus à droite

.

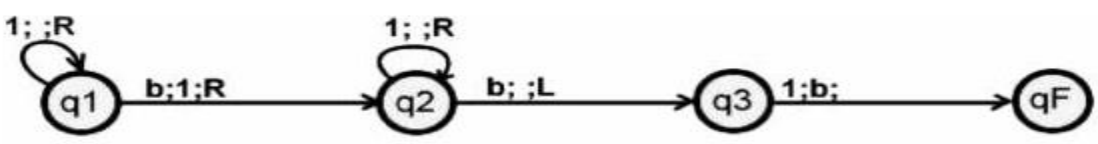
État 3 : Elle écrit un blanc à la place du dernier 1 et passe à l'état final pour s'arrêter.



La table des transitions est la suivante :

Etat	Lecture	Ecriture	Déplacement	Nouvel état
q1	b 0 1	1	R R	q2
q2	b 0 1		L R	q3
q3	b 0 1	b		Etat final qF

Et le diagramme correspondant



4.2 Adition de deux nombres

Soit Q = {q_0, q_1, q_2, q_F } et F = {q_F }. La figure 2 illustre le diagramme à états finis de la MT d'addition unaire. L'idée est de concaténer deux séquences de 1 en entrée. Pour cela, on remplace le premier chiffre du premier nombre de l'entrée par un b et remplace le b entre les deux nombres par un 1.

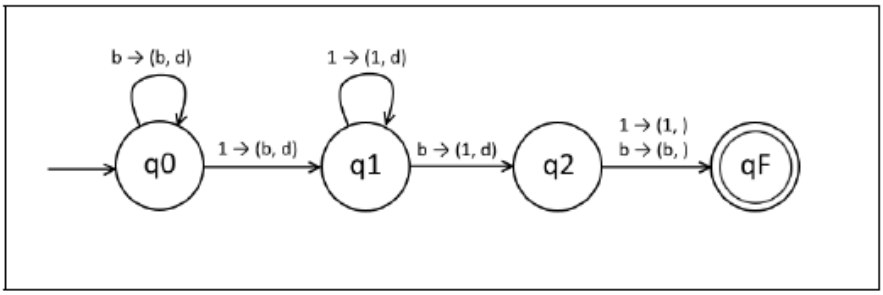


FIGURE 2 – Machine de Turing : Addition Unaire

Exemple d'exécution de la machine en appliquant les règles de transition de la table ci-dessus :
Supposons que :
1. L'entrée est 11 et 111 qui sont séparés par un b sur le ruban
2. La tête de lecture pointe sur le symbole b juste avant le premier nombre à gauche
3. La tête de lecture est représentée par un triangle

b	1	1	b	1	1	1	b
▲							

Nous sommes à l'état $q0$: On lit b donc on écrit b et on va à droite.

b	1	1	b	1	1	1	b
	▲						

Nous sommes à l'état $q0$: On lit 1 donc on passe à l'état $q1$ et on écrit b puis on va à droite.

b	b	1	b	1	1	1	b
		▲					

Nous sommes à l'état $q1$: On lit 1 donc on écrit 1 et on va à droite.

b	b	1	b	1	1	1	b
			▲				

Nous sommes à l'état $q1$: On lit b donc passe à l'état $q2$, on écrit 1 puis on va à droite.

b	b	1	1	1	1	1	b
				▲			

Nous sommes à l'état $q2$: On lit 1 donc on passe à l'état qF, écrit 1 et on s'arrête.

b	b	**1**	**1**	**1**	**1**	**1**	b

Terminaison de la MT.

Résultat : Nous avons bien 11 + 111 = 11111, conclusion : la machine a bel et bien réalisé une addition unaire. Pour exécuter une machine de turing on peut soit lire les règles de transitions ou lire le graphe puis écrire successivement ses configurations. Une configuration correspond à une représentation d'un état de la machine qui comprend le ruban, des symboles écrits dessus et la position de la tête de lecture.