CHAPITRE 4

Les machines de Turing

Jean-Marc Alliot

4.1 Introduction

Dans les années 1930-1936, la communauté mathématique travaillait à définir la notion de calculabilité et plus particulièrement de calculabilité intuitive. Grossièrement, les mathématiciens cherchaient à savoir ce que l'homme estimait être une fonction mathématique que l'on pouvait calculer, au sens pratique du terme.

Certains résultats avaient déjà été obtenus par Kleene et Church en 1932-1935 : ils avaient défini une classe de fonctions, les fonctions *λ-calculables*, qui, d'après eux, représentaient la classe des fonctions intuitivement calculables. Ces travaux allaient donner naissance au *λ-calcul* qui reste une base de l'informatique théorique (cf. chapitre 12).

En 1934, Kurt Gödel présenta une autre classe de fonctions, les fonctions *récursives générales*. Cette classe représentait également à ses yeux la classe des fonctions intuitivement calculables. Il fut démontré par Church et Kleene en 1936 que les fonctions λ-calculables et les fonctions récursives générales recouvraient la même classe de fonctions calculables *i.e.*, toute fonction récursive générale était λ-calculable et réciproquement.

L'inconvénient de ces deux présentations étaient leur manque de signification intuitive. C'est en 1936 que le mathématicien anglais Turing[1] publia un article décrivant la classe des fonctions appelées ultérieurement *Turing-calculables*. Comme nous allons le voir une fonction *Turing-calculable* est une fonction pour laquelle on est capable de construire une description algorithmique de calcul à destination de *machines de Turing*. En ce sens, les machines de Turing peuvent être considérées comme les précurseurs des

1 Alan Mathison Turing (1912-1954) fut un des grands mathématiciens de son époque. Il se consacra aux problèmes de métamathématique et d'informatique théorique, au même titre que Gödel, Church, Kleene, Herbrand, Skolem, Post, Von Neumann... Il participa pendant la seconde guerre mondiale à l'opération ENIGMA qui avait pour but de décoder les messages secrets allemands (la théorie de la cryptographie a d'ailleurs des rapports importants avec l'informatique théorique). C'est à cette occasion (1943) que fut réalisé le premier calculateur électronique, le COLOSSUS (quoique certains estiment en fait que le premier modèle de calculateur électronique a été développé en 1935-1940 par le physicien américain John Atanassof (Mackintosh 1988). La question de savoir quel fut le premier « ordinateur » est complexe. On peut lire pour plus d'informations le commentaire fait par René Moreau de l'article sus-nommé ou pour une vision plus large des choses (Breton 1987)). Turing fut probablement un des mathématiciens qui entrevit le premier ce que pourrait devenir l'informatique. Dans un de ses articles, il énonça tout un ensemble de principes pour construire des machines qui jouent aux échecs ; dans un autre il définit le *test de Turing* dont nous avons parlé dans le chapitre 1. Personnalité étrange, peu adapté à l'Angleterre encore puritaine dans laquelle il vivait, il se suicida à l'âge de 42 ans.

ordinateurs modernes, et les programmes de *machines de Turing* sont les précurseurs des langages procéduraux[2] de programmation classique[3].

En 1937, Turing démontra l'équivalence des fonctions Turing-calculables et récursives générales. Il énonça alors avec Church, la thèse suivante, dite *thèse de Church-Turing* :

Définition 4.1 – Thèse de Church-Turing – *Toute fonction intuitivement calculable est Turing-calculable.*

Il nous faut bien garder en mémoire que ce principe n'est qu'une hypothèse qu'il est impossible de démontrer, la notion de calculabilité intuitive étant bien difficile à déterminer.

Cette thèse serait évidemment infirmée s'il était possible de présenter une fonction calculable « intuitivement » qui ne soit pas Turing-calculable.

Le fait que 50 ans après, on n'ait jamais pu trouver de semblables fonctions incite à penser que cette hypothèse a une bonne chance d'être exacte. D'autre part, certains résultats théoriques (théorème de récursion de Kleene), montrent qu'une fonction contredisant la thèse de Church-Turing devrait être construite de telle façon qu'elle ne ferait appel à aucune fonction dont nous savons qu'elle est Turing-calculable, c'est-à-dire à aucune fonction que nous connaissons aujourd'hui. Il semble difficile de construire intuitivement une telle fonction.

Pourquoi étudions-nous les machines de Turing ? Tout d'abord parce qu'elles représentent une des bases fondamentales de l'informatique théorique, mais aussi pratique. Ensuite parce qu'elles vont nous permettre de donner des définitions simples de la calculabilité et sont indispensables pour étudier les problèmes de complexité. Enfin parce que la démonstration du théorème d'incomplétude de Gödel par les machines de Turing (donnée par Kleene en 1943) est particulièrement élégante et montre les rapports étroits entre métamathématique, logique, informatique théorique et intelligence artificielle.

4.2 Machine de Turing déterministe

Comme nous allons le voir dans ce paragraphe, la définition d'une machine de Turing[4] est celle d'un calculateur numérique idéal.

Définition 4.2 – Structure de stockage – *Une machine de Turing opère sur une bande linéaire divisée en une infinité de cases. Chacune de ses cases pourra, à un instant donné, contenir un symbole pris dans un ensemble fini de symboles* $\Gamma = \{s_0, s_1, \ldots, s_n\}$ *(appelé alphabet). L'ensemble des mots*[5] *sur* Γ *sera noté* Γ^*. *Le symbole* s_0 *est traditionnellement le* blanc, *que nous noterons aussi* B. *À l'instant initial, toutes les cases, sauf éventuellement un*

2 Même s'il n'est historiquement pas évident que les machines de Turing aient réellement influencé le développement des langages classiques.

3 Le λ-calcul fut à la base des langages fonctionnels (LISP, cf. chapitre 17). La logique mathématique engendra la programmation logique (PROLOG, cf. chapitre 18).

4 Pour plus de détails, on peut se reporter à (Minsky 1967; ven Emde Boas 1990).

5 Un mot est une suite finie de symboles de Γ. Nous reviendrons de façon plus précise sur les alphabets et les mots dans le chapitre consacré aux langages formels.

nombre fini, contiennent s_0. Une machine de Turing à un instant donné n'a accès qu'à une seule de ces cases à travers une fenêtre de lecture.

Définition 4.3 – États – *Une machine de Turing se trouve à un instant donné dans un état z_j pris dans un ensemble $Z = \{z_0, z_1, \ldots, z_m, z_h\}$ fini. On distinguera en particulier l'état z_0, dit état de démarrage (ou état initial) et l'état z_h dit état d'arrêt.*

Intuitivement les états correspondent à l'état de l'unité centrale d'un calculateur, alors que la bande et l'alphabet correspondent à la structure de la mémoire.

Définition 4.4 – Fonction de transition – *La fonction de transition d'une machine de Turing est une application δ définie par :*

$$\delta : ((Z - \{z_h\}) \times \Gamma) \rightarrow (Z \times \Gamma \times \{G, D, I\})$$

Ainsi $\delta(z, s) = (z', s', G)$ signifie que si la machine de Turing est dans l'état z et lit une case contenant le symbole s elle passera dans l'état z', écrira dans la case le symbole s' à la place de s et ira sur la case de gauche (D signifie que la machine va à droite et I qu'elle reste immobile).

Si la machine aboutit à l'état z_h, cela signifie que la machine a terminé son calcul. On introduit parfois, sans que cela affecte la puissance de la machine, plusieurs états d'arrêt pour représenter, indépendamment du contenu de la bande lors de l'arrêt, plusieurs résultats possibles[6].

Intuitivement, la fonction de transition correspond à la notion de programme ou d'algorithme dans un calculateur traditionnel. On appelle parfois le graphe de la fonction de transition le programme de la machine de Turing.

Définition 4.5 – Machine de Turing – *Une machine de Turing est définie par le n-uplet $(\Gamma, Z, \delta, z_0, z_h)$.*

Définition 4.6 – Configuration d'une machine de Turing – *La configuration d'une machine de Turing est un élément de $\Gamma^* \times (Z \times \Gamma) \times \Gamma^*$, donc un triplet $(u, (z, s), w)$. u représente le mot à gauche de la tête de la machine de Turing, w le mot à droite de la tête de la machine de Turing, z est l'état de la machine de Turing et s le symbole inscrit dans la case lue par la tête de lecture.*

Une configuration d'arrêt *est un triplet $(u, (z, s), w)$ tel que $z = z_h$.*

Le comportement d'une machine de Turing est entièrement déterminé dès que l'on connaît $(\Gamma, Z, \delta, z_0, z_h)$ et la configuration C courante.

Définition 4.7 – Configuration initiale définie par un argument – *Soit le mot $m = (a_1 a_2 \ldots a_n) \in \Gamma^*$. On appelle configuration initiale définie par m la configuration $(B, (z_1, a_1), a_2 a_3 \ldots a_n B)$. ($B$ signifie que l'on ne trouve plus que des blancs.)*

Définition 4.8 – Séquence de calcul – *On dit qu'une configuration C' succède à une configuration C et l'on note $C \rightsquigarrow C'$ si la machine de Turing dans la configuration C passe*

6 En particulier, l'introduction de deux états d'arrêt z_y et z_n est fréquente lors du calcul d'une fonction à valeur dans $\{t, f\}$ par exemple. Ce sera en particulier le cas lors de l'étude de la complexité des problèmes de décision dans le chapitre 11.

dans la configuration C' en appliquant sa fonction de transition. Une séquence de calcul est une succession $(C_0, C_1, \ldots, C_k)$ de configurations telle que $\forall i, 0 \leq i \leq k-1$, $C_i \rightsquigarrow C_{i+1}$.

Définition 4.9 – Mot accepté par une machine de Turing – *Soit un mot m et la configuration initiale définie par m : $C(m)$. On dit que le mot m est accepté par la machine de Turing $\mathcal{M}$ s'il existe un $k \in \mathbb{N}$ tel que*

$$C(m) \rightsquigarrow C_1 \rightsquigarrow \ldots \rightsquigarrow C_k$$

et C_k est une configuration d'arrêt. Un tel calcul est appelé calcul d'acceptation *de m.*

Définition 4.10 – Langage accepté par une machine de Turing – *L'ensemble des mots m de Γ^* tel que m est accepté par la machine de Turing $\mathcal{M}$ est appelé langage accepté par $\mathcal{M}$. On notera ce langage $L(\mathcal{M})$.*

Une machine de Turing est donc munie d'une capacité mémoire infinie. Il faut d'autre part remarquer que l'ensemble des symboles $\{s_0, s_1, \ldots, s_n\}$ et l'ensemble des états $\{z_0, z_1, \ldots, z_m, z_h\}$ étant l'un et l'autre finis, la définition d'un programme de machine de Turing est nécessairement finie.

Les machines de Turing que nous venons de définir sont appelées *machines de Turing déterministes*. En effet, dans une configuration donnée, la machine de Turing $\mathcal{M}$ peut effectuer au plus une transition (donnée par δ). En partant d'une configuration donnée, elle effectuera donc toujours la même séquence de calcul (séquence qui peut d'ailleurs ne jamais se terminer).

Il existe bien d'autres définitions des machines de Turing déterministes (ou d'autres modèles) que l'on peut trouver chez quantité d'auteurs (Kleene 1987; Salomaa 1989; Johnson 1990; van Emde Boas 1990; Garey and Johnson 1979). Il peut s'agir de machines à plusieurs bandes, ou une bande de longueur finie d'un côté, avec un alphabet de deux symboles, ou une définition des états de démarrage et d'arrêt différente de celle que nous donnons. Il faut savoir que toutes ces machines de Turing sont équivalentes au sens de la calculabilité.

4.3 Notion de calcul

4.3.1 Calcul d'un argument et fonction calculable

Définition 4.11 – Calcul d'un argument – *On dit qu'une machine de Turing calcule* la valeur $b \in \Gamma^*$ *pour l'*argument $a \in \Gamma^*$ *si :*

- *a est accepté par $\mathcal{M}$*
- *lorsque la machine de Turing atteint son état d'arrêt, le mot représenté par la bande est b.*

Le temps d'acceptation étant fini, et une écriture ayant lieu par transition, b est nécessairement fini. La notion de calcul d'un argument par une machine de Turing est bien adéquate à l'idée intuitive que avons de la notion de calcul. Remarquons que la notion de finitude du temps de calcul (du nombre fini d'étapes dans le calcul d'acceptation) est

	B	0	1
z_0	(z_1, B, G)	$(z_0, 0, D)$	$(z_0, 1, D)$
z_1	$(z_h, 1, I)$	$(z_h, 1, I)$	$(z_1, 0, G)$

Table 4.1 – Programme de la machine de Turing effectuant $f(x) = x + 1$

fondamentale dans la définition. Si nous construisons un programme de machine de Turing dont nous démontrons qu'il calculera un résultat pour un argument a sans pouvoir démontrer que ce résultat sera obtenu en un temps fini nous ne dirons pas que la machine de Turing est capable de calculer un résultat lorsqu'elle est appliquée à a.

Définition 4.12 – Fonction calculable – *On dit qu'une fonction f définie sur un domaine $\mathcal{D}$ est Turing-calculable*[7] *si nous pouvons construire une machine M telle que M appliquée à tout élément a de $\mathcal{D}$ calcule $f(a)$.*

Notons tout de suite que la définition impose à la machine de calculer la valeur $f(a)$ pour toute valeur de a de l'ensemble $\mathcal{D}$ sur lequel f est définie, et ce, même si $\mathcal{D}$ est infini.

Remarquons également qu'en raison de la thèse de Church-Turing, les notions de Turing-calculabilité et de calculabilité intuitive sont considérées comme étant équivalentes. On parlera donc de *fonction calculable* au lieu de *fonction Turing-calculable*.

4.3.2 Exemple : $f(x) = x + 1$

Nous allons rapidement montrer que $f(x) = x+1$, définie sur l'ensemble des entiers strictement positifs, est calculable. Pour ce faire, nous allons construire une machine de Turing calculant la fonction f.

Le premier point auquel nous devons nous arrêter est l'alphabet de notre machine. Il s'agit ici du système de représentation numérique que nous allons décider d'adopter. De nombreux auteurs choisissent l'alphabet minimal $\{B, |\}$. Dans cet alphabet, un nombre x est représenté par x bâtons sur la bande. Cette représentation est appelée *représentation unaire*. Ce type de représentation n'est pas admissible en théorie de la complexité (nous y reviendrons dans le chapitre 11), en revanche elle est très utilisée par les logiciens. D'autres adoptent le système de numération décimale, ce qui les amène à considérer l'alphabet $\{B, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

En ce qui nous concerne, nous adopterons, pour cet exemple, le système de numération binaire ($\Gamma = \{0, 1, B\}$) qui représente un intéressant moyen terme entre les systèmes précédents (une machine de Turing opérant sur un alphabet minimal a souvent un grand nombre d'états, alors que le système de numération décimale comporte un nombre de symboles important). Un entier sera codé par sa représentation en base 2.

Le programme de notre machine est représenté table 4.1.

Considérons par exemple la ligne correspondant à l'état z_1. Elle signifie que si la machine est dans l'état z_1 et que le symbole courant est :

7 Cette définition présuppose l'existence d'une fonction de codage qui permet de transformer tout élément de $\mathcal{D}$ en un mot de Γ^*, et d'une fonction de décodage qui effectue le travail inverse.

Un blanc La machine écrit un 1, ne bouge pas et passe dans l'état z_h.

Un 0 La machine écrit un 1, ne bouge pas et passe dans l'état z_h.

Un 1 La machine écrit un 0, se déplace à gauche et reste dans l'état z_1.

Quel est le principe de fonctionnement de cette machine ? Partant de la gauche (premier symbole non blanc) de son argument, elle va vers la droite en restant dans l'état z_0 jusqu'au moment où elle rencontre un symbole blanc. Elle passe alors dans l'état z_1 et revient à gauche. La machine ne sortira de cet état z_1 que lorsqu'elle rencontrera une case 0 ou B lui permettant de terminer son addition, en inscrivant un 1 et en passant dans l'état d'arrêt z_h. Tant qu'elle lira des 1, elle les remplacera par des 0 et passera à gauche.

On constate donc que cette machine calcule la représentation binaire de $x+1$ à partir de la représentation binaire de n'importe quel $x \in \mathbb{N}$. Donc $f(x) = x + 1$ est calculable.

4.4 Réalisation pratique

L'exemple précédent montre qu'il n'est déjà pas immédiat de construire une machine de Turing calculant la fonction $x + 1$.

On peut alors se demander si notre notion de calculabilité intuitive est bien valable, car comment prouver qu'une fonction est calculable si l'on est en pratique incapable de construire la machine de Turing associée ?

Il existe deux réponses à cet argument.

La première est simplement de dire que nous ne demandons pas, pour dire qu'une fonction est calculable, de construire explicitement une machine de Turing, mais simplement de démontrer qu'il existe une machine de Turing calculant la fonction.

Pour ceux qui, membres de l'école constructiviste, refusent ce genre de réponse, signalons qu'il existe des méthodes permettant de construire automatiquement des machines de Turing à partir des fonctions récursives générales, et qu'il est actuellement possible de donner des méthodes permettant de construire de façon automatique une machine de Turing pour toutes les fonctions intuitivement calculables connues.

En fait, dans toute la suite de cet ouvrage, nous ne chercherons jamais à calculer explicitement une machine de Turing et nous nous contenterons de montrer par des arguments intuitifs qu'une telle fonction existe.

4.5 Les machines de Turing non déterministes

Comme nous l'avons vu, une machine de Turing dans un état donné lisant un symbole donné, peut effectuer au plus une action : celle qui figure dans son programme. En revanche une machine de Turing non déterministe peut, dans un état donné et face à un symbole donné, choisir une action parmi plusieurs. Chaque case dans le tableau d'une machine de Turing est remplacée par une liste d'actions possibles. Pour explorer complètement l'ensemble des exécutions possibles, il faut donc effectuer, l'un après l'autre, l'ensemble des choix possibles. Au lieu d'être séquentiel, le schéma d'exécution devient arborescent. Chaque nœud de l'arbre correspond à une configuration et est un point de

choix pour la machine de Turing, chaque segment correspond à une transition, chaque feuille à une position terminale de la résolution (une configuration d'arrêt) et chaque branche à une suite de choix.

Formellement, la différence se situe au niveau de la fonction δ dont l'espace d'arrivée n'est plus $\Delta = (Z \times \Gamma \times \{G, D, I\})$, mais l'ensemble des n-uplets (n *fini*) d'éléments de Δ.

Les machines de Turing non déterministes sont peu utilisées par les logiciens. En revanche, elles ont une grande importance en théorie de la complexité, c'est la raison pour laquelle nous les avons brièvement définies ici.

4.6 Remarques

Notons tout de suite que, de par la définition de la calculabilité, une fonction $f(a)$ n'est calculable que s'il existe une machine de Turing pouvant calculer f pour *toute* valeur de a. En particulier, une machine de Turing doit être capable d'effectuer des calculs sur des nombres arbitrairement grands. C'est parce que les fonctions calculables doivent être totalement calculables, que nous pourrons bientôt construire une fonction non-calculable.

Remarquons également la similitude qui existe entre une machine de Turing et un humain effectuant un calcul *numérique*. Comme le fait remarquer Kleene (Kleene 1987), on peut considérer la bande de la machine de Turing comme une feuille de papier, les symboles utilisés sont les symboles de calcul et les états de la machine sont les « états mentaux » du calculateur humain, états qui lui permettent de se rappeler à quel point du calcul il se trouve à un instant donné. Le nombre d'états du calculateur (humain ou mécanique) est supposé fini. Citons la phrase de Turing :

> « Le nombre d'états mentaux qu'il faut prendre en considération est fini. Si nous admettions une infinité de ces états, certains d'entre eux seraient aussi voisins que l'on voudrait et il y aurait des confusions. »

Il s'agit là, bien entendu, d'un postulat qui prétend qu'un être humain effectue des séries de calculs de façon numérique et non analogique. Si cette hypothèse est peu critiquable quand elle s'applique à des nombres, elle est loin d'être évidente si l'on se réfère à des domaines plus complexes comme la compréhension du langage ou la perception des formes. Les textes de Turing ayant été une des bases du développement de l'IA, on comprend mieux les thèses et postulats mécanistes développés quelques années plus tard en s'appuyant précisément sur ces paroles.