Langages formels (11X003) - Automne 2026

10. Machines de Turing (déterministes)

Enseignant: Arnaud Casteigts

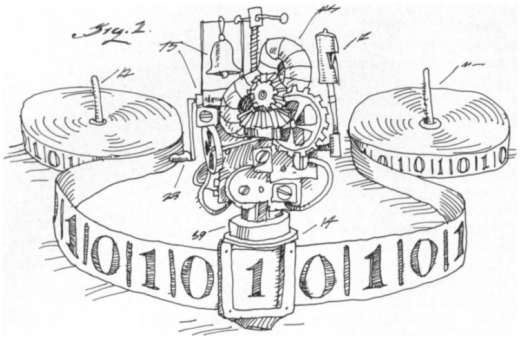
Assistants: E. Arm & M. Marseloo
Moniteurs: A. Hugener & O. Oternaud

Nous avons vu jusqu'à présent plusieurs modèles de machines. Nous avons commencé par les automate finis déterministes et non-déterministes, les deux permettant de reconnaître la même famille de langages (langages réguliers). Puis, nous avons vu les automates à piles, qui utilisent une mémoire infinie, mais limitée dans son usage (la pile). Ces derniers sont plus puissants et correspondent aux langage hors-contextes (pouvant être engendrés par des grammaires hors-contexte). Nous allons maintenant parler du modèle de machine le plus puissant : les *machines de Turing*.

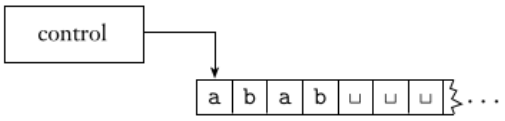
Introduites par Alan Turing en 1936, ces machines peuvent être vues comme un modèle de nos ordinateurs. Elles sont en effet capables de calculer tout ce que nos ordinateurs peuvent calculer.

10.1 Description informelle

Une machine de Turing a un fonctionnement proche de celui d'un automate à pile. La différence principale est que sa mémoire est une bande qui peut être lue ou écrite à n'importe quel endroit. En voici deux représentations :



(Tom Dunne, 2002)

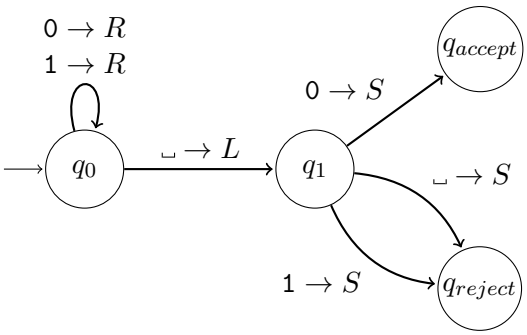


La partie "contrôle" fonctionne comme celle d'un automate (états & transitions), elle commande en outre une **tête de lecture** qui pointe sur une certaine case mémoire de la bande. Les transitions de l'automate dépendent, comme d'habitude, de l'état courant et du

symbole pointé sur la bande. La nouveauté est qu'une transition peut causer l'écriture d'un symbole sur cette bande ainsi qu'un déplacement à gauche ou à droite de la tête de lecture (si l'on essaie d'aller plus à gauche que le début, la tête de lecture ne bouge pas). L'automate possède un état initial et deux autres états spéciaux : q_{accept} et q_{reject}, qui terminent immédiatement l'exécution lorsqu'on s'y trouve. Ce fonctionnement est différent de celui d'un automate car l'exécution s'arrête quelle que soit la position de la tête de lecture.

Au début de l'exécution, le mot d'entrée se trouve au début de la bande, suivi d'une infinité de symboles ␣ (espaces), qui est un symbole réservé (interdit dans l'alphabet d'entrée Σ mais présent dans l'alphabet de bande Γ). La tête de lecture se trouve sur le premier symbole. La machine peut lire les symboles, les modifier, ainsi que lire et écrire arbitrairement loin sur la bande en remplaçant les espaces. Les possibilités sont nombreuses.

Prenons un exemple très simple, par exemple, la reconnaissance des mots qui représentent des nombres pairs sur l'alphabet binaire, décrits par l'expression régulière $L = (\mathtt{0} \cup \mathtt{1})^*\mathtt{0}$. Intuitivement, on souhaiterait que la machine se déplace pour aller lire le dernier symbole, puis accepte le mot si ce symbole est un `0`. On peut faire cela avec 4 états $q_0, q_1, q_{accept}, q_{reject}$ en utilisant la machine ci-dessous. On peut commencer par avoir une transition qui boucle sur l'état initial q_0, déclenchée si le symbole courant est différent de ␣ et qui déplace la tête de lecture vers la droite. Cette transition va se répéter jusqu'à ce que le symbole courant soit ␣. On sait alors que le dernier symbole vient d'être dépassé, on retourne donc un cran à gauche pour se positionner dessus. Enfin, selon le symbole courant, on transitionne vers l'état q_{accept} ou q_{reject} selon qu'il vaut `0` ou `1` (et la tête de lecture reste là où elle est).



10.2 Définition formelle

Il existe plusieurs définitions équivalentes, voici la plus simple. Une **machine de Turing** est un 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ où :

- Q est un ensemble fini d'états,
- Σ est l'alphabet d'entrée (ne contenant pas ␣),
- Γ est l'alphabet de bande (contenant au moins Σ et ␣),
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ est la fonction de transition,

- $q_0 \in Q$ est l'état initial,
- $q_{accept} \in Q$ et $q_{reject} \in Q$ sont deux états distincts qui terminent l'exécution (en acceptant ou rejetant, respectivement).

La fonction de transition δ se lit comme suit : en fonction de l'état actuel et du symbole lu sur la bande, la machine va vers un certain état, écrit un certain symbole au même endroit, puis déplace la tête de lecture vers la gauche (L pour left), vers la droite (R pour right), ou reste sur place (S pour stay). Si le symbole reste inchangé, on peut l'omettre de la représentation, comme dans la machine précédente. Autrement, on l'indique en séparant par une virgule le symbole écrit et le mouvement à effectuer, p.ex. $\mathtt{a} \to \mathtt{b}, R$ représente une transition qui s'active si le symbole lu est `a`, écrit `b` à la place et se déplace vers la droite.

La machine précédente se définit donc avec $Q = \{q_0, q_1, q_{accept}, q_{reject}\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$ et $\delta =$

État de départ	Symbole lu	État d'arrivée	Symbole écrit	Mouvement
q_0	0	q_0	0	R
q_0	1	q_0	1	R
q_0	$\sqcup$	q_1	$\sqcup$	L
q_1	0	q_{accept}	0	S
q_1	1	q_{reject}	1	S

Comme pour les automates, on peut aussi donner δ par une suite de tuples : $(q_0, 0, q_0, 0, R)$, $(q_0, 1, q_0, 1, R), \ldots$, ou encore : $\delta(q_0, 0) = (q_0, 0, R)$, $\delta(q_0, 1) = (q_0, 1, R), \ldots$, tout cela est équivalent. Enfin, lorsque la machine est dessinée, par convention, toutes les transitions non-spécifiées mènent à l'état q_{reject}.

10.2.1 Langages reconnus

À tout moment, l'état global de la machine correspond à trois éléments : 1) l'état courant dans Q ; 2) le contenu de la bande ; 3) la position de la tête de lecture. Le tout est appelé une **configuration**. Il existe une notation compacte pour représenter les configurations. Par exemple, la configuration uqv signifie que la machine est dans l'état q et que le contenu de la bande est le mot uv, avec la tête de lecture qui pointe sur le premier symbole de v.

Pour une machine de Turing M, on dit qu'une configuration C **donne** une configuration C' si le passage de l'une à l'autre est compatible avec la fonction de transition de M. Une exécution pour un mot d'entrée w donné correspond alors à une suite de configurations $C_1, C_2, \ldots$ telle que :

- $C_1 = q_0 w$
- C_i donne C_{i+1} pour tout $i < k$

De plus, si la dernière configuration a pour état q_{accept}, alors on dit que la machine M **accepte** le mot w. Enfin, le **langage reconnu** par M est l'ensemble des mots que M accepte, noté $L(M)$.

Un langage est dit **Turing-reconnaissable** s'il existe une machine de Turing qui le reconnaît. La famille des langages Turing-reconnaissables est au moins aussi grande que celle des langages hors-contextes (car les machines de Turing peuvent simuler les automates à pile).

Pour finir cette section, voici un exemple d'exécution de la machine précédente si le mot `0110` est initialement présent sur la bande :

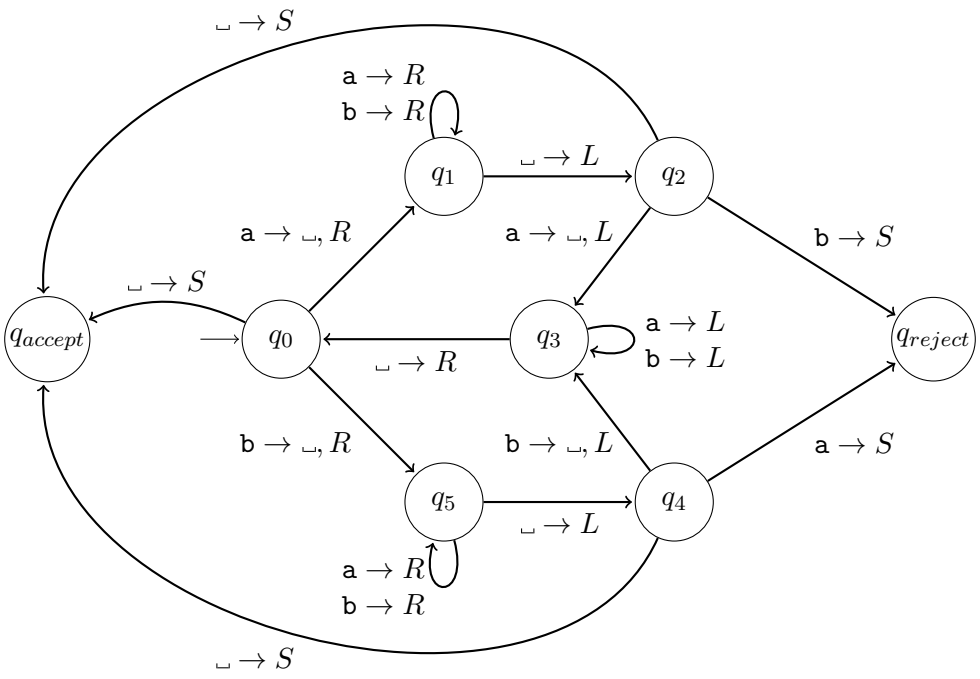
$q_0$0110␣
0$q_0$110␣
01$q_0$10␣
011$q_0$0␣
0110q_0␣
011$q_1$0␣
011q_{accept}0␣

10.3 Palindromes sur l'alphabet $\{\mathtt{a}, \mathtt{b}\}$

Le principal atout des machines de Turing (par rapport aux piles, notamment) est d'être capable d'aller et venir sur la bande sans en détruire le contenu, on parle alors de *zigzags*. Intuitivement, reconnaître un palindrome est assez facile si l'on peut faire des zigzags : on regarde le premier symbole, on l'efface, puis on va vérifier si le dernier est identique et on l'efface à son tour. Puis on se remet au début du mot, et on recommence. Voyons comment faire cela plus précisément. Supposons que le mot d'entrée est `ababa`. On commence par lire et effacer le premier `a`, puis on se déplace vers la droite (en se souvenant, via un état dédié de l'automate, que c'est bien un `a` qu'on a lu). Quand on tombe sur un espace, on revient d'un cran vers la gauche et on vérifie que le symbole correspondant est bien un `a` et on l'efface (sinon, on rejette). On revient alors à gauche jusqu'à rencontrer un espace, on fait un cran vers la droite, on est maintenant prêts à refaire un nouveau zigzag pour vérifier le second symbole, etc. Pour détecter que le travail est terminé, plusieurs options :

- Soit le mot est de longueur impaire, et il sera vide après avoir lu la première lettre lors d'un zigzag : on détectera alors qu'après l'arrivée à droite (en l'occurrence immédiate, mais on a pas besoin de le savoir) et le petit cran à gauche, la case est toujours vide.
- Soit le mot est de longueur paire, et il sera vide après avoir lu la deuxième lettre lors d'un zigzag : on détectera alors qu'après le retour à gauche (idem) et le petit cran à droite, la case est toujours vide.

Dans les deux cas, on peut le détecter et partir immédiatement sur l'état q_{accept}. La machine correspondante est la suivante :



Voici une trace d'exécution pour le mot `ababa` :

Zigzag 1	Zigzag 2	Zigzag 3
$q_0$ababa␣	$q_0$bab␣	q_0a␣
␣$q_1$baba␣	␣$q_5$ab␣	␣q_1␣
␣b$q_1$aba␣	␣aq_5b␣	q_2␣
␣ba$q_1$ba␣	␣abq_5␣	q_{accept}␣
␣babq_1a␣	␣aq_4b␣	
␣babaq_1␣	␣q_3a␣	
␣babq_2a␣	q_3␣a␣	
␣baq_3b␣		
␣b$q_3$ab␣		
␣$q_3$bab␣		
q_3␣bab␣		

Spécifier des machines de Turing en détail peut rapidement être compliqué. Nous ne le ferons que rarement pour des exemples simples. Le reste du temps, nous tâcherons d'expliquer leur fonctionnement par une description à haut niveau, comme celle donnée ci-dessus.

10.4 Thèse de Church-Turing

La notion d'algorithme existe depuis l'antiquité (au minimum), on peut citer par exemple l'algorithme d'Euclide pour trouver le PGCD de deux nombres. Cependant, c'est seulement au vingtième siècle que la notion d'algorithme a été définie de manière précise, par Alonzo Church et Alan Turing. Le premier a défini le λ-calcul et le second les machines de Turing. Bien que très différents, il s'avère que ces deux modèles sont équivalents en termes d'expressivité (chacun des deux peut simuler l'autre). Church et Turing ont alors conjecturé que ces modèles peuvent exprimer n'importe quel traitement physiquement réalisable. C'est ce qu'on appelle la **thèse de Church-Turing**, que personne n'a jamais réussi à contredire jusqu'à aujourd'hui (malgré de nombreuses tentatives). Autrement dit, ces modèles semblent capturer exactement ce qui est physiquement calculable, y compris par nos ordinateurs actuels, avec le grand avantage qu'ils peuvent être définis de manière mathématique et donc étudiés rigoureusement.

En fait, l'écrasante majorité des modèles de calcul imaginés depuis se sont avérés équivalents aux machines de Turing. Autrement dit, à partir d'un certain niveau d'expressivité, tous les modèles sont universels et peuvent se simuler les uns les autres. Cela vaut pour les langages de programmation également : tous les langages de programmations classiques tels que le C, Python, Java, Typescript, *etc.* sont équivalents aux machines de Turing et sont capables de se simuler les uns les autres. On dit qu'ils sont **Turing-complets**.

10.5 Hiérarchie de Chomsky (reloaded)

Un petit retour sur la hiérarchie de Chomsky pour la compléter. Pour rappel, cette hiérarchie regroupe les équivalences entre familles de langages (et grammaires correspondantes) et modèles de calculs capables de les reconnaître.

Important	**Langage**	**Grammaire**	**Machine**
$\rightarrow$	Turing-reconnaissable	Générale	Machine de Turing
	Contextuel	Contextuelle	MT non-dét. linéairement bornée
$\rightarrow$	Hors-contexte	Hors-contexte	Automate à pile non-déterministe
$\rightarrow$	Régulier	Régulière	Automate fini

Nous n'avons quasiment pas parlé des langages contextuels, qui ne sont pas à l'examen. Leur étude est surtout pertinente dans le cadre des grammaires. Quant au modèle de machine qui les reconnaissent, il s'agit de machines de Turing non-déterministes dont la taille de la bande utilisée est proportionnelle à la taille du mot d'entrée ("linéairement bornée"). Ne perdez pas votre énergie là-dessus, à moins que vous ne maîtrisiez déjà le reste.

En revanche, les langages Turing-reconnaissables sont important et nous en reparlerons. Au passage, notez que ces derniers sont aussi appelés langages *récursivement énumérable*.