Informatique Fondamentale

Amaury Pouly
CNRS

ENS Rennes

Cours FOND1 ? ?

Année 2025 – 2026

Version 0.1

Table des matières

1. Version 0.1—4409b7b modified, compiled on 7 septembre 2025

Ajouter un lien vers le cours d'Olivier Bournez

1 Préliminaires

1.1 Contexte

La notion d'algorithme a une longue histoire. Brièvement, on peut dire que le mot « algorithme » existe déjà depuis le 12ème siècle et que c'est la latinisation du nom de Muḥammad ibn Mūsā al-Khwārizmī, un mathématicien persan du 9ème siècle ayant travaillé sur l'algèbre et les systèmes de numération.

La notion de machine à calculer est encore plus longue et il est impossible ici d'en faire une présentation exhaustive. De façon intéressante, les première machines à calculer sont d'ailleurs plutôt analogiques que digitales [**survey**]. La machine analytique de Charles Babbage, imaginée en 1834, est souvent considérée comme la première machine digitale programmable, même si elle n'est pas universelle et ne sera jamais terminée. Le premier ordinateur digital vraiment universel est généralement considéré comme étant l'ENIAC, construit en 1945. On peut toutefois considérer que le premier ordinateur programmable universel est en fait le l'analyseur différentiel construit en 1931 par Vannevar Bush : celui est analogique et on sait maintenant qu'il équivalent aux machines de Turing.

En 1842, Ada Lovelace qui a beaucoup travaillé avec Babbage sur son concept de machine, écrit ce qui est généralement considéré comme les premiers algorithmes de programmation de l'histoire. Elle est surtout reconnue pour avoir perçu l'intérêt de la machine au-delà de ses aspects purement numérique, avec une perspective moderne de calculabilité. On peut considérer que Ada Lovelace est la première informaticienne de l'histoire.

En 1900, David Hilbert présente une liste de 23 problèmes majeures en mathématique. Ces problèmes n'ont pas forcément un énoncé précis, et c'est en particulier le cas du dixième problème. Celui-ci demande s'il est possible, étant donné une équation diophantienne à un nombre quelconque d'inconnues et à coefficients entiers rationnels, de « trouver une méthode par laquelle, au moyen d'un nombre fini d'opérations, on pourra distinguer si l'équation est résoluble en nombres entiers rationnels ». A l'époque, la notion de décidabilité ou d'algorithme n'est formellement définie mais il est clair que c'est bien en ces termes modernes que la question doit se comprendre. On sait maintenant depuis les résultats [1] de Martin Davis, Yuri Matiyasevich, Hilary Putnam et Julia Robinson en 1970 que ce problème est indécidable.

En 1928, avec les avancées dans la formalisation de la logique et des modèles, Hilbert et Wilhelm Ackermann posent le problème de savoir s'il existe une procédure qui décide (renvoie « oui » ou « non ») si une formule du premier ordre est vraie. Cette question est maintenant connue sous le nom du « Entscheidungsproblem ». La notion d'algorithme reste à définir formellement mais celle de problème de décision commence à émerger.

En 1936, plusieurs formalismes sont proposés indépendamment par Alan Turing avec les machines qui portent son nom, Alonzo Church avec le λ-calcul et Emil Post avec les systèmes dits de Post. Turing and Post montrent alors que le *Entscheidungsproblem* n'a pas de solution en supposant que leurs modèles capturent bien la notion d'algorithme. Cette hypothèse est maintenant connue sous le nom de thèse de Church-Turing. Il s'agit d'une question plus philosophique que mathématique. On sait maintenant que tous ces formalismes et bien d'autres sont équivalents. La thèse de Church-Turing est communément admise mais il existe d'autres variantes plus ouvertes.

1.2 Notations

On note $\mathbb{N}$ l'ensemble des entiers naturels (contenant 0) et $\mathbb{Z}$ l'ensemble des entiers relatifs. Un alphabet Σ est un ensemble fini. Une lettre est une élément d'un alphabet. On note Σ^* l'ensemble des mots sur l'alphabet Σ, ε le mot vide, et on note la concaténation de deux mots $u, v \in \Sigma^*$ par uv. Pour tout mot $w \in \Sigma^*$, on note $|w|$ sa longueur et $w_1, \ldots, w_{|w|}$ ses lettres. Ainsi $w = w_1 w_2 \cdots w_{|w|}$. On note Σ^+ l'ensemble des mots non vides sur Σ. Pour tout mot $u \in \Sigma^*$ et $k \in \mathbb{N}$, on note u^k le mot u répété k fois, c'est à dire que $u^0 = \varepsilon$ et $u^{k+1} = uu^k$. Pour tout mot $w \in \Sigma^*$, on note $\tilde{w} = w_{|w|} \cdots w_1$ son miroir.

Étant donné deux ensembles A et B, une fonction f totale de A vers B sera notée $f : A \to B$; une fonction partielle g de A vers B sera notée $g :\subset A \to B$ et on note $\mathrm{dom}(g)$ son ensemble de définition. Pour tout ensemble A, on note $\mathcal{P}(A)$ l'ensemble des sous-ensembles de A.

2 Machines de Turing

2.1 Intuition

Introduire la notion de MT, expliquer les choix

1. Le résultat est souvent appelé théorème de Matiyasevich, mais Matiyasevich est celui-ci qui a terminé la preuve.

2.2 Définition

Formellement, une machine de Turing ressemble beaucoup à un automate fini, la principale différence est la fonction de transition qui possède une information supplémentaire : la direction de déplacement. En effet, une machine de Turing se déplace sur un ruban, c'est ce qui explique sa puissance de calcul.

Définition 1 (Machine de Turing)**.** Une machine de Turing (MT) déterministe à un ruban est la donnée de

$$\mathcal{M} = (Q, \Sigma, \Gamma, \mathbf{B}, \delta, q_0, F)$$

où

— Q est un ensemble fini d'états,
— Σ est l'alphabet d'entrée,
— Γ est l'alphabet de travail : $\Sigma \subset \Gamma$,
— $\mathbf{B} \in \Gamma \setminus \Sigma$ est le symbole (ou caractère) blanc,
— δ est une fonction (partielle) de transition de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$,
— $q_0 \in Q$ est l'état initial,
— $F \subseteq Q$ est l'ensemble des états acceptants.

Dans la suite de cette section, nous allons définir plusieurs notions. Ces notions sont évidemment toutes dépendentes de la machine $\mathcal{M}$ définie plus haut mais nous ne le rappelerons pas à chaque fois.

Remarque 1. Comme nous le verrons plus loin, il existe de nombreuses variantes des machines de Turing et celles-ci sont toutes équivalentes. On peut noter dès maintenant qu'il existe des « variations syntaxiques qui sont clairement équivalentes. Par exemple, on peut avoir un ensemble d'états acceptants/refusant, ou supposer que la fonction δ est totale.

Afin de pouvoir définir la notion de calcul, il faut d'abord définir la notion de configuration, c'est à dire l'état de machine. Pour une machine déterministe, une configuration encode exactement toutes les informations dont on besoin pour poursuivre le calcul. Il existe de nombreuses façons équivalentes de présenter une configuration. Nous en choisissons une particulier qui a l'avantage d'être très simple à manipuler et qui encode bien l'idée de « localité du calcul ».

Rappelons-nous que la tête de lecture se déplace sur un ruban bi-infini (par dont toutes les cases sauf un nombre fini sont blancs). On peut donc couper le ruban en trois : une partie x strictement à gauche de la tête de lecture, la case σ où se trouve la tête de lecture et la partie y strictement à droite de la tête de lecture. Pour les parties gauches et droite, seul un nombre fini de case est non-blanc, on peut donc prendre le plus grand préfixe non totalement blanc du ruban et cela nous donne un mot sur Γ (l'alphabet de travail). On note que x et y se lisent « dans un sens différent » puisque x_1 est la case juste à gauche de la tête, x_2 la case à gauche de la gauche de la tête et ainsi de suite (on lit donc de droite à gauche) ; tandis que y_1 est la case juste à droite de la tête, y_2 à droite de la droite de la tête (on lit donc de gauche à droite).

Définition 2 (Configuration)**.** Une configuration est donnée par (q, x, σ, y) où

— $q \in Q$ est l'état courant,
— $x \in \Gamma^*$ est le contenu du ruban à gauche de la tête de lecture,
— $\sigma \in \Sigma$ est le contenu de la case sous la tête de lecture,
— $y \in \Gamma^*$ est le contenu du ruban à droite de la tête de lecture.

Lorsqu'on lance une machine de Turing sur un mot $w \in \Sigma^*$, il faut définir la configuration de départ dans laquelle se trouve la machine. Le choix que nous avons fait pour la définition d'une configuration rend la définition légèrement plus pénible car il faut distinguer deux cas selon que l'entrée est vide ou non. Pour cela, il est utile d'introduire les fonctions suivantes pour tout mot $w \in \Sigma^*$:

$$\mathrm{hd}(w) = \begin{cases} \mathbf{B} & \text{si } w = \varepsilon, \\ w_1 & \text{sinon,} \end{cases} \qquad \mathrm{tl}(w) = \begin{cases} \varepsilon & \text{si } w = \varepsilon, \\ w_2 \cdots w_{|w|} & \text{sinon,} \end{cases} \tag{1}$$

En d'autres terme, $\mathrm{hd}(w)$ est la première lettre de w sauf si w est vide auquel cas c'est $\mathbf{B}$. De même $\mathrm{tl}(w)$ est w privé de sa première lettre. On voit qu'il va vite devenir pénible d'écrire des hd et tl partout si on veut décrire des configurations. C'est pourquoi on s'autorise à simplifier la notation.

Notation 1 (Configuration simplifiée)**.** Pour tout $q \in Q$ et $x, y \in \Gamma^*$, on note $(\!|q, x, y|\!) := (q, x, \mathrm{hd}(y), \mathrm{tl}(y))$.

Définition 3 (Configuration initiale)**.** La configuration initiale sur un mot $w \in \Sigma^*$ est $C_0 = (q_0, \varepsilon, \mathrm{hd}(w), \mathrm{tl}(w))$.

On peut maintenir définir l'opération fondamentale d'une machine de Turing, à savoir un pas de calcul.

Définition 4 (Étape de calcul)**.** Pour une configuration $C = (q, x, \sigma, y)$ et une autre configuration C', on dit que C' est le successeur de C, et on note $C \rightsquigarrow C'$, si $q \notin F$ et

— si $\delta(q, \sigma) = (q', \sigma', \downarrow)$ alors $C' = (q', x, \sigma', y)$;

— si $\delta(q, \sigma) = (q', \sigma', \leftarrow)$ alors $C' = (q', \mathrm{tl}(x), \mathrm{hd}(x), \sigma' y)$,

— si $\delta(q, \sigma) = (q', \sigma', \rightarrow)$ alors $C' = (q', \sigma' x, \mathrm{hd}(y), \mathrm{tl}(y))$.

Remarque 2 (Importante !)**.** A noter que dans la définition ci-dessus, puisque la fonction δ est partielle, une configuration C peut ne pas avoir de successeur, même si $q \notin F$. Cette possibilité est extrêmement importante dans la suite afin de définir la notion de langage reconnu. Il existe d'autres conventions, comme par exemple le fait de prendre la fonction δ totale mais d'avoir deux ensembles finaux : un ensemble d'états acceptants et un ensemble refusant. Tout cela est complètement équivalent.

Définition 5 (Configuration finale)**.** Une configuration C est dite finale s'il n'existe pas de configuration C' telle que $C \rightsquigarrow C$.

Définition 6 (Configuration acceptante/refusante)**.** Une configuration $C = (q, x, \sigma, y)$ est dite acceptante si $q \in F$, et refusante si elle est finale mais non acceptante.

On note qu'une configuration acceptante est nécessairement finale puisque par définition si $q \in F$ alors C n'a pas de successeur. A partir de la configuration initiale C_0, la machine va donc passer par une suite (unique puisque la machine est déterministe) $C_0 \rightsquigarrow C_1 \rightsquigarrow C_2 \rightsquigarrow \cdots$ de configurations. Cette suite peut être **finie ou infinie**.

Définition 7 (Résultat d'un calcul)**.** Pour tout mot $w \in \Sigma^*$, le résultat du calcul de $\mathcal{M}$ sur w, noté $\mathcal{M}(w)$ est défini ainsi.

— S'il existe une suite finie $C_1, \ldots, C_n$ de configurations telles que $C_0 \rightsquigarrow C_1 \rightsquigarrow \cdots \rightsquigarrow C_n$ avec C_n une configuration finale, alors on pose $\mathcal{M}(w) = C_n$. On dit alors que **la machine termine/s'arrête** sur w.

— Sinon on pose $\mathcal{M}(w) = \bot$ et on dit que **la machine diverge/boucle** sur w.

On insiste bien sur le fait qu'il y a trois possibilités pour un calcul :

— se terminer dans une configuration acceptante,

— se terminer dans une configuration refusante,

— diverger.

Il est parfois utile pour les raisonnement d'écrire de façon compact le résultat d'une étape de calcul sur une configuration donnée. Etant donné une configuration C d'une machine $\mathcal{M}$, on notera $\mathcal{M}(C)$ l'unique (puisque $\mathcal{M}$ est déterministe) configuration telle que $C \rightsquigarrow \mathcal{M}(C)$. Si C n'a pas de successeur, alors on pose $\mathcal{M}(C) = \bot$. Par convention, on posera aussi $\mathcal{M}(\bot) = \bot$. Enfin, pour tout entier n, on pose $\mathcal{M}^{[n]}(C)$ la n-ième itérée de $\mathcal{M}$, c'est à dire que $\mathcal{M}^{[0]}(C) = C$ et $\mathcal{M}^{[n+1]} = \mathcal{M}(\mathcal{M}^{[n]}(C))$ pour tout entier n.

2.3 Langages

La notion la plus simple à définir une fois qu'on une machine de Turing est celle de langage. Il s'agit d'une généralisation de la notion de langage régulier. Cette notion est toutefois beaucoup plus subtile car il faut prendre en compte la possibilité qu'ont les machines de Turing de diverger. Dans la suite on se fixe un alphabet Σ.

Définition 8 (Langage décidé par machine de Turing)**.** Un langage $\mathcal{L} \subseteq \Sigma^*$ est décidé par machine de Turing (ou décidable) s'il existe une machine $\mathcal{M}$ sur l'alphabet [2] Σ telle que

— pour tout mot $w \in \mathcal{L}$, $\mathcal{M}(w)$ est une configuration acceptante,

— pour tout mot $w \notin \mathcal{L}$, $\mathcal{M}(w)$ est une configuration refusante.

Il est **très important** de noter que cette définition implique que la machine $\mathcal{M}$ doit **terminer** sur toutes les entrées, elle n'a pas le droit de diverger. On peut aussi noter que le contenu final du ruban n'a aucune importante.

2. Il s'agit de l'alphabet d'entrée, la machine peut utiliser n'importe quel espace de travail Γ.

2.4 Fonctions

La notion de langage a l'avantage de la simplicité mais limitante dans certains contexte. Nous voulons donc définir plus généralement ce qu'est une fonction calculable. Là encore il faut faire attention au cas où la machine diverge. Cette fois-ci, le contenu du ruban à la fin du calcul sera importante. Là encore il existe plusieurs conventions : on peut prendre tout le ruban, ou bien seulement la partie droite (ou gauche). Dans tous les cas, un point important est que l'on doit « nettoyer » le ruban afin d'oublier les symboles blancs en trop. Pour cela on définit la fonction auxiliaire suivante pour tout mot $w \in \Gamma$ et $\mathbf{B} \in \Gamma$:

$$\mathrm{str}(w) = \begin{cases} \varepsilon & \text{si } w = \varepsilon, \\ \mathrm{str}(u) & \text{si } w = u\mathbf{B}, \\ \mathrm{str}(v) & \text{si } w = \mathbf{B}v, \\ w & \text{sinon.} \end{cases} \tag{2}$$

Autrement dit, la fonction str enlèvent les blancs au début et la fin du mot. Une définition équivalente est que pour tout mot u ne contenant pas $\mathbf{B}$, $\mathrm{str}(\mathbf{B}^n u \mathbf{B}^m) = u$ pour tous $n, m \in \mathbb{N}$.

Définition 9 (Fonction totale calculable)**.** Pour tous alphabets Σ et Σ', une fonction $f : \Sigma \to \Sigma'$ est calculable s'il existe une machine de Turing $\mathcal{M}$ sur l'alphabet $\Sigma \cup \Sigma'$ telle que pout tout $w \in \Sigma^*$, on a $f(w) = \mathrm{str}(\tilde{x}\sigma y)$ où $\mathcal{M}(w) = (q, x, \sigma, y)$.

Autrement dit, pour tout entrée w la machine $\mathcal{M}$ doit **s'arrêter** sur w et contenir $f(w)$ sur le ruban (après nettoyage des blancs).

Remarque 3 (Fonctions partielles)**.** Il est possible de définir la notion de fonction partielle calculable mais il existe (au moins) deux définitions possibles en fonction de comment on veut traiter les éléments en dehors du domaine. La première possibilité est de demander à la machine de s'arrêter mais refuser sur les entrées en dehors du domaine de définition. La deuxième possibilité est de ne poser aucune contrainte (la machine peut s'arrêter ou diverger). La première notion est beaucoup plus forte puisque l'on voit en particular qu'elle implique que le domaine de définition est décidable.

2.5 Exemples et techniques de preuves

On considère maintenant quelques exemples de langages et machines. Ceci va nous permette d'introduire une convention graphique, similaire à cette des automates finis, et de voir différentes techniques pour montrer qu'une machine décide un langage.

2.5.1 Un langage très simple

Considérons le langage $\mathcal{L} = \mathtt{a}^*\mathtt{b} = \{\mathtt{a}^n\mathtt{b} : n \in \mathbb{N}\}$ sur l'alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. On se propose de construire une machine $\mathcal{M}$ qui décide $\mathcal{L}$ en posant

$$\mathcal{M} = (\{q_a, q_b, q_f\}, \Sigma, \Sigma \cup \{\mathbf{B}\}, \mathbf{B}, \delta, q_a, \{q_f\})$$

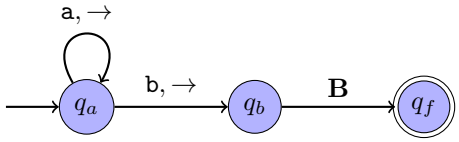
où $\mathbf{B}$ est un nouveau symbole, différent de $\mathtt{a}, \mathtt{b}$. Formellement on peut définir δ par la table de transition suivante (une entrée vide signifie que δ n'est pas définie) et aucune transition à partir de q_f.

σ	$\delta(q_a, \sigma)$
$\mathtt{a}$	$(q_a, \mathtt{a}, \rightarrow)$
$\mathtt{b}$	$(q_b, \mathtt{b}, \rightarrow)$
$\mathbf{B}$	

σ	$\delta(q_b, \sigma)$
$\mathtt{a}$	
$\mathtt{b}$	
$\mathbf{B}$	$(q_f, \mathbf{B}, \downarrow)$

ou bien graphique comme ci-dessous. Chaque arête entre deux états q et q' annotée par « $\sigma/\sigma', d$ » signifie que $\delta(q, \sigma) = (q', \sigma', d)$. Afin de ne pas trop alourdir les notations, on écrira simplement σ, d lorsque $\sigma' = \sigma$ et on peut omettre d lorsque $s = \downarrow$. Rappelons-nous que δ est une fonction partielle, ainsi s'il n'existe pas d'arête pour une paire (q, σ) donnée, alors δ n'est pas définie sur cette paire. On notera l'état initial avec une flêche entrante, et les états finaux avec un double cercle.

$\mathcal{M}$:



Montrer qu'une machine de Turing reconnaît un langage est un exercice subtil : il est très facile de se perdre dans les détails sans pour autant expliquer l'idée. Toutefois, simplement donner la machine et annoncer que « ça marche » n'est pas très convainquant. Il s'agit de trouver le bon équilibre. On présente ici deux preuves : une informelle qui explique l'intuition de la construction, et une complètement formelle. En pratique, on a rarement besoin d'être aussi formel mais il est important de savoir le faire si besoin.

Lemme 1. *La machine $\mathcal{M}$ décide le langage $\mathcal{L}$.*

Démonstration intuitive. On va scanner le mot de gauche à droite et essentiellement simuler un automate fini, c'est à dire que l'on ne va jamais modifier le ruban. Si on tombe sur $\mathbf{B}$, on sait que l'on a tout lu (puisque l'alphabet d'entrée ne contient pas $\mathbf{B}$), on va donc s'arrêter. Si en lisant le mot, on se rend compte que celui-ci n'appartient pas à $\mathcal{L}$, on bloque dans un état non final, ce qui signifie que le mot est rejeté.

On commence dans un état q_a et tant que l'on voit des $\mathtt{a}$, on reste dans cet état en se déplaçant vers la droite. Si jamais on rencontre un $\mathbf{B}$ dans cet état, la machine bloque. Puisque l'état q_a n'est pas final, cela garanti que si le mot ne possède pas de $\mathtt{b}$, alors la machine le rejette (i.e. bloque dans un état non final).

Supposons maintenant que le mot possède un $\mathtt{b}$, c'est à dire qu'il est de la forme $\mathtt{a}^n\mathtt{b}w$. La machine se déplace vers la droite dans l'état q_a puis lit $\mathtt{b}$ dans l'état q_a, et passe donc dans l'état q_b et se déplace vers la droite. La machine est maintenant dans l'état q_b au début du mot w. On veut rejeter tous les mot w sauf ε, ainsi on veut bloquer si on lit un $\mathtt{a}$ ou $\mathtt{b}$. Si on lit un $\mathbf{B}$, c'est que l'on a atteint la fin du mot : on passe dans un nouvel état q_f qui est final. Par définition, la machine bloque et accepte dès qu'elle atteint un état final donc le mot est accepté seulement si $w = \varepsilon$. □

Démonstration très formelle. On considère trois cas pour le mot en entrée : ε, $\mathtt{a}^n\mathtt{b}w$ et $\mathtt{a}^n$. On vérifie bien que tout mot sur Σ est de l'une de ces formes.

Sur l'entrée ε, la configuration initiale est $C_0 = (q_a, \varepsilon, \mathbf{B}, \varepsilon)$ par définition. Or $\delta(q_a, \mathbf{B})$ n'est pas définie donc la machine bloque en C_0 dans un état non final et rejette le mot.

Sur l'entrée $\mathtt{a}^n\mathtt{b}w$, il y a deux sous-cas. Si $n > 0$ alors la configuration initiale est $C_0 = (q_a, \varepsilon, \mathtt{a}, \mathtt{a}^{n-1}\mathtt{b}w)$. Puisque $\delta(q_a, \mathtt{a}) = (q_a, \mathtt{a}, \rightarrow)$, on voit que

$$C_0 \rightsquigarrow (q_a, \mathtt{a}, \mathtt{a}, \mathtt{a}^{n-2}\mathtt{b}w) \rightsquigarrow \cdots \rightsquigarrow C' := (q_a, \mathtt{a}^n, \mathtt{b}, w).$$

Si $n = 0$ alors la machine commence directement dans l'état C'. Mais alors $C' \rightsquigarrow C'' := (q_b, \mathtt{a}^n\mathtt{b}, \mathrm{hd}(w), \mathrm{tl}(w))$ puisque $\delta(q_a, \mathtt{b}) = (q_b, \mathtt{b}, \rightarrow)$. Il faut donc distinguer deux cas : si $w = \varepsilon$ alors $\mathrm{hd}(w) = \mathbf{B}$ donc $C'' \rightsquigarrow (q_f, \mathtt{a}^n\mathtt{b}, \mathbf{B}, \varepsilon)$ qui est une configuration acceptante. Si $w \neq \varepsilon$ alors $\mathrm{hd}(w) \neq \mathbf{B}$ donc la machine bloque en C'' qui est une configuration refusante.

Sur l'entrée $\mathtt{a}^n$, on observe que d'une part le ruban ne contient initialement aucun $\mathtt{b}$, et que par ailleurs la machine n'écrit jamais un $\mathtt{b}$ si elle ne lit pas un $\mathtt{b}$. Ainsi par induction immédiate, toute configuration accessible depuis la configuration initiale ne contient jamais de $\mathtt{b}$ et donc la machine ne pourra jamais atteindre l'état q_f qui est le seul état acceptant. La machine rejette donc le mot. □

2.5.2 Un langage plus complexe

Une particularité des langages réguliers est justement de ne pas nécessiter de ruban. L'exemple suivant montre comment le ruban nous permet de décider des langages plus complexes. En particulier, il illustre une technique classique de programmation de machines de Turing : faire des aller-retours sur le ruban en modifiant le contenu avec de nouveaux symboles pour ajouter de l'information. Considérons le langage

$$\mathcal{L} = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^{n-1}\mathtt{b}\cdots\mathtt{b}\mathtt{a}\mathtt{b} : n \geqslant 0\}$$

sur l'alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

L'idée pour décider ce langage est d'abord de généraliser le problème. On pose $\Gamma = \Sigma \cup \{x, \mathbf{B}\}$ et on considère un mot de la forme

$$x^{k_1}\mathtt{a}^{n_1}\mathtt{b}x^{k_2}\mathtt{a}^{n_2}\mathtt{b}\cdots\mathtt{b}x^{k_p}\mathtt{a}^{n_p}\mathtt{b}.$$

Initialement, tout mot (qui se termine par $\mathtt{b}$) sur l'alphabet Σ est forcément de cet forme, avec $k_i = 0$ pour tout i. La machine de Turing que nous allons construire va procéder en un certain nombre de tours. A chaque tour, la machine va scanner le mot puis revenir au point de départ.

— Durant le scan de gauche à droite, la machine va changer chaque $x^{k_i}\mathtt{a}^{n_i}$ en $x^{k_i+1}\mathtt{a}^{n_i-1}$, c'est à dire changer un $\mathtt{a}$ en un x. Elle bloque si $n_i = 0$.

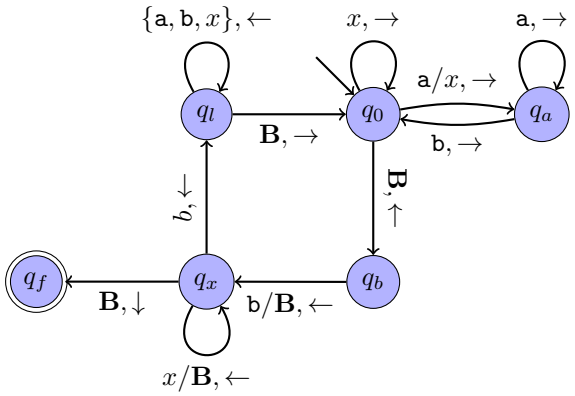
— Durant le scan de droite à gauche, elle vérifie que $n_p = 0$ et efface le suffixe $x^{k_p}\mathtt{b}$ du mot.

Après un tour, le contenu du ruban est donc

$$x^{k_1+1}\mathtt{a}^{n_1-1}\mathtt{b}x^{k_2+1}\mathtt{a}^{n_2-1}\mathtt{b}\cdots\mathtt{b}x^{k_{p-1}+1}\mathtt{a}^{n_{p-1}-1}\mathtt{b}.$$

On observe qu'en répétant cette procédure, soit la machine bloque, soit elle atteint le ruban vide. Ce dernier cas survient si et seulement si le mot initial était dans $\mathcal{L}$. La machine qui correspond à cet algorithme est décrite ci-dessous. Afin de simplifier les notations, on introduit une nouvelle convention : une transition peut être annoté par « X, d » avec X un ensemble, pour indiquer qu'il s'agit en fait de l'ensemble des transitions x, d pour $x \in X$.

$\mathcal{M}$:



Les états correspondent intuitivement aux actions suivantes :

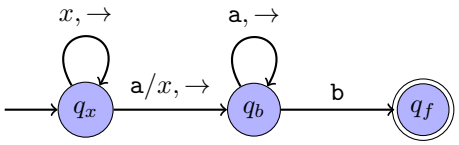
- q_0 : on avance vers la droite en ignorant les x au début de chaque bloc $x^{k_i}\mathtt{a}^{n_i}\mathtt{b}$, jusqu'à rencontrer un $\mathtt{a}$. On le remplace alors par un x et on va en q_a. Si l'on rencontre $\mathbf{B}$, on recule vers la gauche et on va en q_b.
- q_a : on avance vers la droite en ignorant les $\mathtt{a}$ jusqu'à rencontrer un $\mathtt{b}$. On se déplace alors à droite et on retourne en q_0 pour procéder de même avec le prochain bloc.
- q_b : on efface le dernier $\mathtt{b}$, on recule vers la gauche et on va en q_x.
- q_x : on efface tous les x en reculant vers la gauche jusqu'à tomber sur un $\mathtt{b}$. On va alors en q_l. Si jamais on rencontre $\mathbf{B}$, alors on va en q_f pour accepter.
- q_l : on recule vers la gauche en ignorant tous les $\mathtt{a}, \mathtt{b}$ et x, jusqu'à rencontrer $\mathbf{B}$. On retourne alors en q_0 pour commencer le prochain tour.

Lemme 2. *La machine $\mathcal{M} = (\{q_0, q_a, q_b, q_x, q_l, q_f\}, \Sigma, \Gamma, \boldsymbol{B}, \delta, q_0, \{q_f\})$ décide le langage $\mathcal{L}$, où δ est définie graphiquement ci-dessus.*

Montrer ce lemme directement et formellement est possible mais pénible. Surtout une telle stratégie de preuve (construire la machine directement) ne marche pas pour des langages plus complexes. Une meilleur stratégie est plutôt de constuire des petites machines qui effectuent des tâches simples, puis de les combiner en des machines de plus en plus sophistiquées.

Si on reprend notre intuition, la première chose qu'il faut savoir faire c'est convertir $x^n\mathtt{a}^p\mathtt{b}$ en $x^{n+1}\mathtt{a}^{p-1}\mathtt{b}$, mais dans un context plus général : il peut y avoir des symboles à gauche et droite. Autrement dit, notre machine commence dans une configuration non-initiale. Construisons cette machine :

$\mathcal{M}_1$:

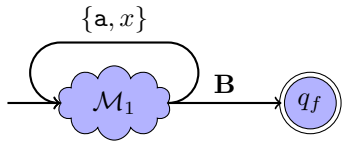


On se convainc facilement du résultat suivant (qui peut se prouver formellement !). On utilise la Notation 1 $(\!|q, x, y|\!)$ afin de simplifier la présentation.

Lemme 3. *Soit $C = (\!|q_x, u, v|\!)$ une configuration. Si v est de la forme $x^k\mathtt{a}^n\mathtt{b}w$ avec $k \geqslant 0$, $n \geqslant 1$, $w \in \Gamma^*$ alors $\mathcal{M}_1$ accepte à partir de C dans la configuration $(\!|q_f, \mathtt{b}\mathtt{a}^{n-1}x^{k+1}u, w|\!)$, sinon $\mathcal{M}_1$ refuse.*

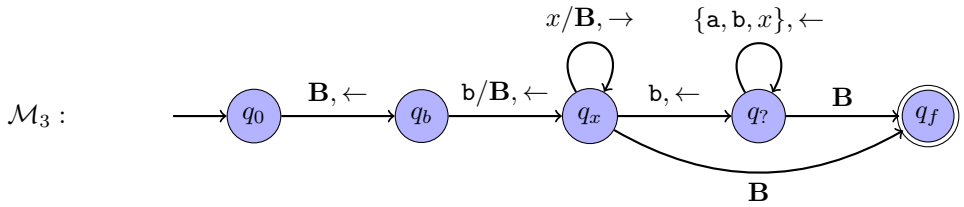
On peut maintenant construire une machine qui va scanner le ruban de gauche à droite et appliquer la machine $\mathcal{M}_1$ à chaque bloc.

$\mathcal{M}_2$:



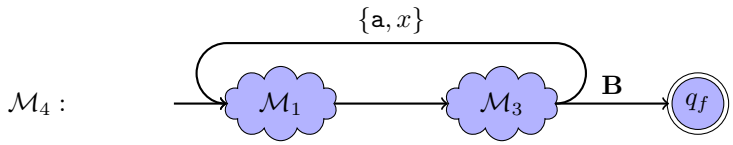
Lemme 4. *Soit $C = (\!|q_0, u, v|\!)$ une configuration (q_0 état initial de $\mathcal{M}_2$). Si v est de la forme $x^{k_1}\mathtt{a}^{n_1}\mathtt{b}\cdots x^{k_p}\mathtt{a}^{n_p}\mathtt{b}$ avec $p \geqslant 1$, $k_1, \ldots, k_p \geqslant 0$, $n_1, \ldots, n_p \geqslant 1$ alors $\mathcal{M}_2$ accepte à partir de C dans la configuration $(\!|q_f, \tilde{w}u, \varepsilon|\!)$ où $w = x^{k_1+1}\mathtt{a}^{n_1-1}\mathtt{b}\cdots x^{k_p+1}\mathtt{a}^{n_p-1}\mathtt{b}$, sinon $\mathcal{M}_2$ refuse.*

Après avoir scanné le ruban de gauche à droite, on veut effacer le dernier bloc en vérifiant qu'il est de la forme $x^k\mathtt{b}$, puis retourner au début du mot.



Lemme 5. *Soit $C = (\!|q_0, u, \varepsilon|\!)$ une configuration (q_0 état initial de $\mathcal{M}_3$). Si $\tilde{u}$ est de la forme $wx^k\mathtt{b}$ avec $w = \varepsilon$ ou w se terminant par $\mathtt{b}$ alors $\mathcal{M}_3$ accepte à partir de C dans la configuration $(\!|q_f, \varepsilon, w|\!)$ sinon $\mathcal{M}_3$ refuse.*

Finalement, on peut combiner tout ensemble : on scanne de gauche à droite puis de droite gauche jusqu'à bloquer ou arriver au mot vide.



Lemme 6. *La machine $\mathcal{M}_4$ décide le langage $\mathcal{L}$.*

2.6 Variantes

Il existe de nombreuses variantes des machines de Turing. Nous avons déjà vu quelques variations syntaxiques mais il existe des variations plus sémantique. Nous listons ici les plus importantes. Comme nous allons le voir, toutes ces variantes produisent exactement les mêmes langage décidables. On peut voir cela comme un cas particulier de la thèse de Church (Section 2.8).

2.6.1 Alphabet binaire

Une question naturelle est de savoir si l'alphabet d'entrée ou de travail change fondamentalement le modèle. La réponse est non car on peut simuler n'importe quelle machine avec une autre machine travaillant sur un alphabet binaire (plus le symbole blanc). Il existe plusieurs façon d'énoncer formellement ce résultat. Le plus simple est probablement pour les langages.

Soit Σ un alphabet quelconque, $k \geqslant 1$ et $\phi : \Sigma \to \{\mathtt{0}, \mathtt{1}\}^k$ une fonction injective[3]. Autrement dit, ϕ est un encodage binaire quelconque des lettres de Σ. Par example si $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, on pourrait choisir $k = 2$, $\phi(\mathtt{a}) = \mathtt{00}$, $\phi(\mathtt{b}) = \mathtt{01}$ et $\phi(\mathtt{c}) = \mathtt{10}$. On peut alors s'intéresser à la question de savoir si décider un langage sur Σ est relié à la question de décider son image par ϕ.

Proposition 1. *Pour tout $\mathcal{L} \subseteq \Sigma^*$, $\mathcal{L}$ est décidable par machine de Turing si et seulement si $\phi(\mathcal{L})$ l'est avec $\Gamma = \{\mathtt{0}, \mathtt{1}, \boldsymbol{B}\}$ comme alphabet de travail.*

Idée de la démonstration. L'idée est de prendre une machine $\mathcal{M}$ qui reconnaît $\mathcal{L}$, et de la transformer en une machine qui travaille sur l'alphabet $\{\mathtt{0}, \mathtt{1}\}$. Pour cela, on introduit pour chaque état q, k états intermédiaires qui vont lire k symboles sur le ruban. Puisque chaque symbole $\sigma \in \Sigma$ est encodé par k symboles binaire, on peut alors voir à quoi cet encodage correspondait et simuler la transition d'origine dans $\mathcal{M}$. Pour cela, il faut de nouveaux introduire k états pour écrire le nouveau symbole sur le ruban puis encore k états pour pouvoir faire le déplacement si nécessaire. □

On peut aussi montrer un résultat plus précis de simulation, qui relie précisément le comportement d'une machine sur Σ et sur $\{\mathtt{0}, \mathtt{1}\}$. Pour cela, il faut définir une notion d'encodage des configuration. On montre alors qu'une étape de calcul de $\mathcal{M}$ correspond à un nombre fixé d'étapes d'une machine $\mathcal{M}'$ travaillant sur $\{\mathtt{0}, \mathtt{1}\}$.

Lemme 7. *Pour toute machine $\mathcal{M}$ sur Σ, il existe un entier p, une machine $\mathcal{M}'$ et une fonction injection $\psi : Q \to Q'$ où Q (resp. Q') est l'ensemble des états de $\mathcal{M}$ (resp. $\mathcal{M}'$), tels que pour toute configuration C de $\mathcal{M}$, on a ${\mathcal{M}'}^{[p]}([C]) = [\mathcal{M}(C)]$ où l'on pose pour toute configuration, $[(q, x, \sigma, y)] = (\psi(q), \phi(x), \phi(\sigma), \phi(y))$.*

Diagramme commutatif ?

Démonstration.
TODO

□

3. Plus généralement, on peut prendre $\phi : \Sigma \to \{\mathtt{0}, \mathtt{1}\}^*$ *prefix free* : pour tout σ, $\phi(\sigma)$ n'est un préfixe d'aucun $\phi(\sigma')$ pour $\sigma' \neq \sigma$.

2.6.2 Machines à plusieurs rubans

Une modification très courante des machines de Turing consiste à ajouter plusieurs rubans. Fixons $k \geqslant 1$ le nombre de rubans. On modifie la fonction de transition qui devient une fonction de $Q \times \Gamma^k$ dans $Q \times \Gamma^k \times \{\leftarrow, \downarrow, \rightarrow\}^k$. Une configuration devient la donnée de $(q, (x_i, \sigma_i, y_i)_{i=1,\dots,k})$, c'est à dire que l'on a k rubans (x_i, σ_i, y_i). La machine possède k têtes de lectures, une étape de calcul dépend donc de la valeur des symboles sous les k têtes de lectures et on peut bouger chaque tête indépendamment.

Proposition 2. *Pour tout $\mathcal{L} \subseteq \Sigma^*$, $\mathcal{L}$ est décidable par machine de Turing à k rubans si et seulement si $\mathcal{L}$ est décidable par machine de Turing à 1 ruban.*

Idée de la démonstration. L'idée est assez simple même si les détails sont techniques. Etant donné une machine $\mathcal{M}$ à k rubans sur Γ, on va construire une machine à un ruban sur $\Gamma' = \Gamma \cup \{\#, \star\}$. Si $R_1, \dots, R_k$ est le contenu complet des k rubans, alors on va l'encoder sur un seul ruban en séparant les R_i par des $\#$. De plus, on va insérer un $\star$ dans chaque R_i pour indiquer où se trouve les têtes de lectures de $\mathcal{M}$. Ainsi initialement le ruban est $\star R_1 \# \star R_2 \# \cdots \# \star R_k$. Afin de simuler une étape de calcul de $\mathcal{M}$, on va scanner le ruban de gauche à droite et noter dans l'état de la machine le symbole à droite des k têtes de lectures identifiée par $\star$. Une fois fait, on sait quelle transition la machine doit effectuer (le nouvel état, les nouveaux symbole, les déplacements des têtes). On va donc reparcourir le ruban et effectuer les modifications sur chacun des k ruban et déplacer les $\star$ qui représentent les têtes.

Une subtilité de la construction est que les rubans peuvent grandir. Par exemple si la tête de lecture se trouve au début de R_2 et que la machine veut déplacer cette tête de lecture vers la droite, il ne faut pas écraser le contenu de R_1. Dans ce cas, on peut décaler tous les symboles du ruban vers la droite pour créer une espace libre. Concrètement, le ruban va passer de $R_1 \# \star R_2$ à $R_1 \# \mathbf{B} \star R_2$ et ensuite à $R_1 \# \star \mathbf{B} R_2$. □

2.6.3 Machines de Turing oublieuses

On termine par une variante surprenante : les machines dites oublieuse (« oblivious » en anglais). On dit qu'une machine est oublieuse si les mouvements de la tête de lecture sont les mêmes pour toutes les entrées de même taille.

Formellement, on peut le définir de la façon suivante : pour tout mot w, soit $C_0(w) \rightsquigarrow C_1 \rightsquigarrow \cdots \rightsquigarrow C_k$ la suite des configurations de la machine sur l'entrée w, avec C_k une configuration finale. On pose alors $d(w) = (d_1, \dots, d_k)$ où $d_i \in \{\leftarrow, \downarrow, \rightarrow\}$ est la direction dans la laquelle la tête s'est déplacée pour passer de C_{i-1} à C_i. La machine est dite oublieuse si pour tous mots w, $d(w)$ ne dépend que de $|w|$. Autrement dit, si w et w' sont deux mots de même longueur, alors $d(w) = d(w')$.

Proposition 3. *Pour tout $\mathcal{L} \subseteq \Sigma^*$, $\mathcal{L}$ est décidable par machine de Turing oublieuse si et seulement si $\mathcal{L}$ est décidable par machine de Turing.*

2.7 Non-déterminisme

Une extension importante des machines de Turing consiste à ajouter du non déterminisme dans le calcul. Pour cela, on modifie simplement la fonction de transition δ qui peut maintenant retourner non pas une mais plusieurs valeurs possibles.

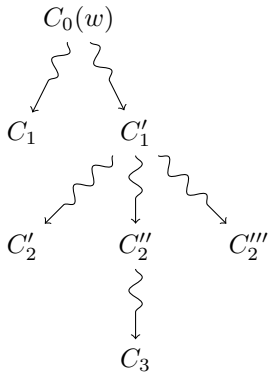
Définition 10 (Machine non déterministe)**.** Une machine non déterministe est donnée par $\mathcal{M} = (Q, \Sigma, \Gamma, \mathbf{B}, \delta, q_0, F)$ comme dans la Definition 1 mais où δ est une fonction (totale) de transition de $Q \times \Gamma$ dans $\mathcal{P}(Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\})$.

Formellement, une machine déterministe est une machine non déterministe où $\delta(q, \sigma)$ est soit l'ensemble vide soit un singleton pour tout q et σ. La notion de configuration reste la même, de même pour les configurations finales, acceptantes et refusante. Le changement majeur qu'introduit le non déterminisme est le fait qu'une configuration **peut avoir plusieurs successeurs**.

Définition 11 (Étape de calcul non déterministe)**.** Pour une configuration $C = (q, x, \sigma, y)$ et une autre configuration C', on dit que C' est **un** successeur de C, et on note $C \rightsquigarrow C'$, si $q \notin F$ et il existe $q' \in Q$ et $\sigma' \in \Gamma$ tels que

- si $(q', \sigma', \downarrow) \in \delta(q, \sigma) =$ alors $C' = (q', x, \sigma', y)$;
- si $(q', \sigma', \leftarrow) \in \delta(q, \sigma) =$ alors $C' = (q', \mathrm{tl}(x), \mathrm{hd}(x), \sigma' y)$,
- si $(q', \sigma', \rightarrow) \in \delta(q, \sigma) =$ alors $C' = (q', \sigma' x, \mathrm{hd}(y), \mathrm{tl}(y))$.

On se convaint aisément que cette notion est compatible avec celle des machines déterministes. La conséquence principale est que maintenant, un calcul n'est plus une suite de configuration mais **un arbre de configuration** avec la configuration initiale à la racine et chaque configuration ayant ses successeurs comme enfants.



Remarque 4 (Importante !)**.** Nous avons vu qu'une machine déterministe peut diverger, c'est à dire produire une suite infinie de configurations qui ne sont jamais finales. De la même façon, une ou plusieurs ou toutes les branches de l'arbre peuvent être infinie.

Maintenant qu'un mot possède non pas une mais plusieurs configurations finales, il faut définir ce que l'on entend par décider ou refuser. La notion d'acceptation reste la même qu'en Definition 7 : un mot w est accepté s'il existe un chemin acceptant (une suite finie de configuration menant à une configuration acceptante). Toutefois la notion de refus est moins claire en général car la machine pourrait ne poser aucun chemin acceptant, des chemins refusant et des chemins divergeant. On évite le problème en définissant directement la notion de langage décidé et en demandant (comme dans la cas déterministe) que la machine ne diverge jamais.

Définition 12 (Langage décidé par machine de Turing non déterministe)**.** Un langage $\mathcal{L} \subseteq \Sigma^*$ est décidé par machine de Turing (ou décidable) s'il existe une machine $\mathcal{M}$ sur l'alphabet Σ telle que

— pour tout mot $w \in \mathcal{L}$, $\mathcal{M}$ possède un chemin acceptant à partir de la configuration initiale pour w.

— pour tout mot $w \notin \mathcal{L}$, tous les chemins de $\mathcal{M}$ à partir de la configuration initiale pour w sont refusants.

En particulier, $\mathcal{M}$ ne possède aucun chemin divergeant.

On peut naturellement se poser la question de savoir si l'ensemble des langage décidé par des machines déterministes est le même que celui décidé par des machines non déterministes est le même. La réponse est oui car on peut toujours simuler une machine non déterministe avec une machine déterministe (au prix d'un coût important en complexité).

Théorème 1. *Pour tout langage $\mathcal{L}$, $\mathcal{L}$ est décidé par machine déterministe si et seulement si $\mathcal{L}$ est décidé par machine non déterministe.*

Idée de la démonstration. Afin de simplifier la démonstration, on peut procéder en deux temps. Dans un premier temps, on montre qu'on peut se limiter à $|\delta(q,\sigma)| \leqslant 2$ puis qu'on peut simuler avec une machine déterministe.

Le premier point est le plus simple : si $X = \delta(q,\sigma)$ est de taille $k > 2$ alors on peut introduire deux états intermédiaires q_1 et q_2 et poser $\delta(q,\sigma) = \{(q_1,\sigma,\downarrow),(q_2,\sigma,\downarrow)\}$. On décompose $X = X_1 \cup X_2$ avec $|X_1|, |X_2| \leqslant |X|/2$ puis on pose $\delta(q_1,\sigma) = X_1$ et $\delta(q_2,\sigma) = X_2$. On a donc réduit la taille de X par deux. En procédant de même récursivement pour chaque paire (q,σ), on voit qu'on arrivera à limiter $\delta(q,\sigma)$ à un ensemble de taille au plus 2.

Supposons maintenant que $|\delta(q,\sigma)| \leqslant 2$ pour tout (q,σ). Puisque la machine est non déterministe, elle peut à un moment donner se trouver dans un ensemble possible de configurations. On a donc devoir construire une machine dont le ruban est de la forme $C_1\#C_2\#\cdots\#C_k$ où C_i est un encode d'une configuration. La machine va alors simuler une étate non déterministe de la machine d'origine : elle traite chaque C_i et simule une transition. Si cette transition bloque, elle efface C_i du ruban. Si cette transition est déterministe, elle modifie C_i. Enfin si cette transition a deux successeurs, elle commence par copier C_i à la fin du ruban puis simule chaque possibilité. □

2.8 Thèse de Church

Thèse de Church

3 Calculabilité

Un fait fondamental des machines de Turing (ou plus généralement tout modèle de calcul suffisamment puissant) est qu'elles peuvent se simuler elles-mêmes. C'est sur ce résultat qu'est basée toute la théorie de la calculabilité.

3.1 Codage des paires et autres structure

Un problème récurrent est de définir des langages ne portant pas simplement sur des mots mais des structures plus complexes comme des paires ou des listes. Un exemple typique est un langage de paires $(\mathcal{M}, w)$ avec $\mathcal{M}$ une machine et w un mot. Afin de formaliser cette notion, nous allons introduire un encodage pour les paires. Comme dans la section précédente, on suppose pour simplifier que l'alphabet d'entrée est $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ et on se permet un alphabet de travail étendu. On peut donc introduire un nouveau symbole $\# \notin \Sigma$ et poser, pour tout $u, v \in \Sigma^*$,

$$\langle u, v \rangle = u\#v.$$

Cet encodage est calculable dans le sens où les fonctions $\langle u, v\rangle \mapsto u$ et $\langle u, v\rangle \mapsto v$ sont calculables par machine de Turing (exercice laissé au lecteur). On voit que cet encodage se généralise immédiatement aux listes : pour tout k, pour tous $U \in (\Sigma^*)^*$, on peut poser

$$\langle U \rangle = U_1\# \cdots \#U_{|U|}.$$

On montre alors qu'étant donné l'encodage d'une liste, on peut calculer sa longueur et pour tout i, on peut extraire U_i. Plus formellement, les fonctions

$$\langle U \rangle \mapsto |U|, \qquad \langle \langle U \rangle, \mathtt{1}^i \rangle \mapsto U_i$$

sont calculables (on a ici pris en encodage unaire pour les entiers pour simplifier mais un codage binaire marche aussi).

A partir de ces primitives, on peut facilement représenter des objects plus complexes comme des graphes par exemple. Dans la suite, on se permettra donc, lorsque l'encodage est clair, de parler de langages de paires, ou de machines prenant ou retournant des paires en entrée/sorties, et ainsi de suite.

3.2 Codages des machines

Afin de formaliser l'idée qu'une machine puisse en simuler une autre, il faut tout d'abord décrire *comment* on peut « donner en entrée » une machine à une autre. Autrement, il faut décrire un encodage des machines de Turing sous forme de symboles. Rappelons-nous que par la Proposition 1, il suffit de savoir décrire les machines travaillant sur $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ avec $\Gamma = \{\mathtt{0}, \mathtt{1}, X\}$ où pour éviter toute confusion nous avons noté le symbole blanc différemment. Posons donc

$$\mathfrak{M} = \{\mathcal{M} : \mathcal{M} \text{ machine sur } \Sigma \text{ et travaillant sur } \Gamma\}.$$

Notre but est donc de définir une fonction d'encodage

$$\begin{cases} \mathfrak{M} & \to & \Delta^* \\ \mathcal{M} & \mapsto & [\mathcal{M}] \end{cases}$$

avec Δ un alphabet fixé. Il est possible de prendre $\Delta = \{\mathtt{0}, \mathtt{1}\}$ mais pour simplifier la présentation, nous allons nous autoriser un alphabet plus grand. Toutefois par la Proposition 1, on sait qu'on peut s'y ramener par un morphisme de Δ vers $\{\mathtt{0}, \mathtt{1}\}$ et donc en composant ce morphisme avec notre fonction d'encodage, on obtient un encodage binaire. On pose donc

$$\Delta = \Gamma = \{\mathtt{0}, \mathtt{1}, X\}.$$

L'encodage que l'on définit doit être raisonnable, par cela on entend principalement qu'il doit être décodable : en lisant $[\mathcal{M}]$ on doit pouvoir retrouver $\mathcal{M}$ (ou une machine équivalente). Nous allons formaliser cette idée dans la prochaine section.

Définition 13 (Encodage d'une machine de Turing)**.** Soit $\mathcal{M} = (Q, \Sigma, \Gamma, \mathbf{B}, \delta, q_0, F)$ une machine. On numérote les éléments de Q par $Q = \{q_1, \ldots, q_{|Q|}\}$ avec la convention que $q_1 = q_0$ est l'état initial, et on pose $[q_i] = \mathtt{1}^i\mathtt{0}$ pour tout i. On pose aussi $[\leftarrow] = \mathtt{0}$, $[\rightarrow] = \mathtt{1}$ et $[\downarrow] = X$. Soit $T_1, \ldots, T_t \in Q \times \Gamma$ les éléments du domaine de δ. On pose

$$[\delta] = [T_1][\delta(T_1)] \cdots [T_t][\delta(T_t)]$$

où pour tout $q \in Q$, $\sigma \in \Gamma$ et $d \in \{\leftarrow, \downarrow, \rightarrow\}$,

$$[(q, \sigma)] = [q]\sigma, \qquad [(q, \sigma, d)] = [q]\sigma[d].$$

On pose

$$[F] = \tau_1 \cdots \tau_{|Q|}, \qquad \tau_i = \begin{cases} \mathtt{1} & \text{if } q_i \in F, \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

Enfin on pose

$$[\mathcal{M}] = [\delta]\mathtt{0}[F].$$

Exemple 1. Reprenons l'exemple de la Section 2.5.1 (avec le renommage $\mathtt{a} = \mathtt{0}$, $\mathtt{b} = \mathtt{1}$, $\mathbf{B} = X$) : on a $\mathcal{M} = (\{q_a, q_b, q_f\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{0}, \mathtt{1}, X\}, X, \delta, q_a, \{q_f\})$ et δ définie par le tableau ci-dessous.

σ	$\delta(q_a, \sigma)$
$\mathtt{0}$	$(q_a, \mathtt{0}, \rightarrow)$
$\mathtt{1}$	$(q_b, \mathtt{1}, \rightarrow)$
X	

σ	$\delta(q_b, \sigma)$
$\mathtt{0}$	
$\mathtt{1}$	
X	$(q_f, X, \downarrow)$

On commence par numéroter les états de Q, en commençant par l'état initial : $q_1 = q_a, q_2 = q_b, q_3 = q_f$. On a donc $[q_a] = \mathtt{10}$, $[q_b] = \mathtt{1^20}$ et $[q_f] = \mathtt{1^30}$. Le domaine de δ est $\{T_1, T_2, T_3\}$ avec $T_1 = (q_a, \mathtt{0}), T_2 = (q_a, \mathtt{1}), T_3 = (q_b, X)$. L'encodage de δ est donc

$$\begin{aligned}[\delta] &= [T_1][\delta(T_1)][T_2][\delta(T_2)][T_3][\delta(T_3)] \\ &= [(q_a, \mathtt{0})][(q_a, \mathtt{0}, \rightarrow)][(q_a, \mathtt{1})][(q_b, \mathtt{1}, \rightarrow)][(q_b, X)][(q_f, X, \downarrow)] \\ &= \underbrace{\mathtt{10}}_{[q_a]}\,\mathtt{0}\,\underbrace{\mathtt{10}}_{[q_a]}\,\mathtt{0}\,\underbrace{\mathtt{1}}_{[\rightarrow]}\,\underbrace{\mathtt{10}}_{[q_a]}\,\mathtt{1}\,\underbrace{\mathtt{110}}_{[q_b]}\,\mathtt{1}\,\underbrace{\mathtt{1}}_{[\rightarrow]}\,\underbrace{\mathtt{110}}_{[q_b]}\,X\,\underbrace{\mathtt{1110}}_{[q_f]}\,X\,\underbrace{X}_{[\downarrow]}.\end{aligned}$$

Par ailleurs, le seul état final est q_3 donc

$$[F] = \underbrace{\mathtt{0}}_{q_1 \notin F}\,\underbrace{\mathtt{0}}_{q_2 \notin F}\,\underbrace{\mathtt{1}}_{q_3 \in F}.$$

Finalement, l'encodage de $\mathcal{M}$ est

$$[\mathcal{M}] = \underbrace{\mathtt{10010011011101110}X\mathtt{1110}X\,X}_{[\delta]}\,\mathtt{0}\,\underbrace{\mathtt{001}}_{[F]}.$$

3.3 Machine universelle

Montrer l'existence d'une machine universelle

3.4 Indécidabilité

Montrer l'existence de language non calculables, introduire la notion de semi-calculable (récursivement énumérable), montrer qu'il existe des langage non semi-calculables

3.5 Réductions et conséquences

Introduire la notion de réduction, les conséquences pour la calculabilité

3.6 Problème indécidables classiques

Donner (sans preuve) des classique comme le dixième problème de Hilbert

3.7 Théorèmes importants

Rice, SMN, point fixe, machine qui affiche son code

4 Complexité en temps

4.1 Définition

4.2 La classe P

4.3 Réductions en temps polynomial

4.4 La classe NP

4.5 NP-complétude

4.6 Le problème 3-SAT

4.7 Problèmes NP-complet classiques

4.8 Théorème de hierarchie

4.9 Problèmes intermédiaires et en dehors de NP

5 Complexité en espace

5.1 Définition

5.2 La classe PSPACE

5.3 Réductions en espace polynomial

5.4 Théorème de hierarchie

5.5 Théorème de Savitch

5.6 Problèmes en dehors de la classe PSPACE

6 Fonctions récursives

TODO ?

6.1 Définition

6.2 Équivalence avec les machines de Turing

7 Machines non-digitales

TODO