Modèles de calcul – machines de Turing

Exercice 1. *Questions*

Répondre aux questions suivantes, où on considère le modèle standard de machine de Turing, sans variante.

1. Une machine peut-elle écrire un $\square$ sur son ruban ?
2. Une étape de calcul peut-elle laisser la tête de lecture immobile ?
3. Une étape de calcul peut-elle laisser le ruban non modifié ?
4. Combien de cases du ruban peuvent être modifiées à chaque étape ?
5. Si la machine possède n états et l'alphabet k symboles, combien de transitions la machine possède-t-elle au maximum ? Et au minimum ?

Exercice 2. *Exemples de machines*

1. Décrire des machines de Turing effectuant les calculs suivants.
   - **i.** Étant donné un mot $w \in \{0,1\}^*$, la machine calcule son *complémentaire* $\overline{w}$ défini par $\overline{w}_i = 1 - w_i$ (par ex., 0010 devient 1101).
   - **ii.** Étant donné un entier $n > 0$ écrit en binaire, la machine calcule $n-1$.
   - **iii.** Étant donné $a\#b$ où a et b sont deux entiers écrits en binaire, la machine calcule $a+b$.
   - **iv.** Étant donné un mot $w \in \{a,b\}^*$, la machine calcule $w\#w$.
   - **v.** Étant donné un mot $u \oplus v$ où u, $v \in \{0,1\}^*$, la machine calcule $w = u \text{ XOR } v$.
   - **vi.** Étant donné deux entiers a et b, en binaire, sur deux rubans différents, la machine calcule $a+b$ sur un troisième ruban, en temps linéaire en la taille de a et b.
2. Décrire des machines de Turing reconnaissant les langages suivants.
   - **i.** $L = \{w \in \{0,1\}^* : w \text{ est l'écriture binaire d'un entier pair}\}$
   - **ii.** $L = \{w \in \{0,1\}^* : w \text{ ne contient jamais trois 0 à la suite}\}$
   - **iii.** $L = \{w \in \{0,1,\#\}^* : w = u\#v \text{ où } u,v \in \{0,1\}^* \text{ ont la même longueur}\}$
   - **iv.** $L = \{a^n b^n : n \in \mathbb{N}\}$
   - **v.** $L = \{w \in \{a,b\}^* : w \text{ contient autant de } a \text{ que de } b\}$
   - **vi.** $L = \{w \in \{a,b\}^* : w \text{ contient deux fois plus de } a \text{ que de } b\}$
   - **vii.** $L = \{w \in \{a,b\}^* : w \text{ ne contient pas deux fois plus de } a \text{ que de } b\}$

Exercice 3. *Qu'est-ce qu'elles font ?*

On considère les deux tables de transitions suivantes.

Table 1

q_0	a	a	$\rightarrow$	q_a
	b	b	$\rightarrow$	q_b
q_a	a	a	$\rightarrow$	q_a
	b	b	$\rightarrow$	q_a
	$\square$	$\square$	$\leftarrow$	q'_a
q_b	a	a	$\rightarrow$	q_b
	b	b	$\rightarrow$	q_b
	$\square$	$\square$	$\leftarrow$	q'_b
q'_a	a	a	$\rightarrow$	oui
q'_b	b	b	$\rightarrow$	oui

Table 2

q_0	0	$\square$	$\rightarrow$	q_1
q_1	0	X	$\rightarrow$	q_2
	X	X	$\rightarrow$	q_1
	$\square$	$\square$	$\rightarrow$	oui
q_2	0	0	$\rightarrow$	q_3
	X	X	$\rightarrow$	q_2
	$\square$	$\square$	$\leftarrow$	q_4
q_3	0	X	$\rightarrow$	q_2
	X	X	$\rightarrow$	q_3
q_4	0	0	$\leftarrow$	q_4
	X	X	$\leftarrow$	q_4
	$\square$	$\square$	$\rightarrow$	q_1

1. **i.** Dessiner l'automate correspondant à la table **1** et lister l'ensemble des états et l'alphabet.

ii. Écrire la suite des configurations obtenues sur les entrées `aaba`, `bab` et `aab`.

iii. Décrire le langage reconnu par la machine.

2. **i.** Dessiner l'automate correspondant à la table **2** et liste l'ensemble des états et l'alphabet.

ii. Écrire la suite des configurations obtenues sur les entrées `0`, `00`, `000` et `000000`.

On souhaite maintenant comprendre ce que fait cette machine. Supposons que l'entrée est 0^n pour un certain $n > 1$.

iii. Montrer que si n est impair, la machine se bloque dans l'état q_3.

iv. Montrer que si n est pair, après n étapes de calcul, la configuration est $q_2 : (X0)^{n/2-1}X\check{\square}$. En déduire qu'après quelques étapes supplémentaires, la machine est dans la configuration $q_1 : \check{X}(0X)^{n/2-1}$.

v. En déduire que depuis une configuration $q_1 : \check{w}$ où $w \in \{0,X\}^*$ où w possède k symboles 0, la machine soit se bloque en q_3 si k est impair, soit atteint une configuration $q_1 : \check{w}'$ où w' possède $k/2$ symboles 0. *Note.* $\check{w}$ *signifie que la tête est sur la* $1^{ère}$ *lettre de* w.

vi. En déduire le langage reconnu par la machine.

Exercice 4. *Variantes*

1. On considère un modèle de machine de Turing où la tête de lecture est initialement placée sur la dernière lettre (la plus à droite) de l'entrée. Ce modèle est-il équivalent au modèle standard ?

2. On considère une variante des machines de Turing, où chaque étape de calcul peut soit écrire sur le ruban, soit déplacer la tête de lecture, mais pas les deux

en même temps. Formellement, les transitions sont de la forme $\delta(q, x) = (q', a)$ où $q, q' \in Q$ sont des états, $x \in \Sigma$ est un symbole et $a \in \Sigma \sqcup \{\leftarrow, \rightarrow\}$ est une *action* : si $a \in \Sigma$, la machine écrit a sur le ruban ; si $a \in \{\leftarrow, \rightarrow\}$, la machine ne modifie pas le ruban mais déplace sa tête à gauche ou à droite. Montrer que ce modèle modifié est équivalent au modèle standard. *Indication. Montrer qu'une transition d'une machine standard peut être simulée sur une machine modifiée, et réciproquement.*

3. **i.** On considère une machine de Turing $\mathcal{M}$ qui, à chaque transition, déplace sa tête vers la droite (mais jamais vers la gauche). Montrer que $L(\mathcal{M})$ est régulier, c'est-à-dire peut être reconnu par un automate fini.

ii. Montrer que le résultat reste vrai si on autorise la machine à soit déplacer sa tête vers la droite, soit la garder immobile.

Exercice 5. *Reconnaître, décider, calculer*

1. Montrer que si un algorithme A *décide* un langage L, alors A le *reconnaît* également.

2. Soit $\mathcal{M}_c$ une machine de Turing qui *calcule* une fonction totale $f : \Sigma^* \to \{0, 1\}$. Montrer qu'on peut construire une machine de Turing $\mathcal{M}_d$ qui *décide* $L_f = \{w \in \Sigma^* : f(w) = 1\}$.

3. On applique la même construction qu'à la question précédente pour une machine $\mathcal{M}_c$ qui calcule une fonction *partielle* $f : \Sigma^* \to \{0, 1\}$. Que fait la machine $\mathcal{M}_d$ obtenue ?

4. Inversement, soit $\mathcal{M}_d$ une machine de Turing qui *décide* un langage L. Montrer qu'on peut construire une machine $\mathcal{M}_c$ qui *calcule* la fonction caractéristique χ_L de L.

Exercice 6. *Programmes* WHILE *et machines RAM*

On considère des programmes WHILE et des machines RAM dont les variables ou registres sont sur n bits.

1. Pour les deux fonctions ci-dessous, écrire un programme WHILE puis le traduire en machine RAM.

i. $\text{SUB}(x, y) = x - y \bmod 2^n$

ii. $\text{MUL}(x, y) = x \times y \bmod 2^n$.

2. On s'intéresse à des machines dont les entrées et sorties sont dans $\{0, 1\}$: quelque soit la taille de mot n, on suppose que les valeurs des entrées sont dans $\{0, 1\}$ (sans avoir à le vérifier).

i. Écrire le code d'une machine RAM qui étant $x \in \{0, 1\}$ calcule $\neg x$.

ii. Écrire le code d'une machine RAM qui étant donné $x, y \in \{0, 1\}$ calcule $x \wedge y$.

iii. Écrire le code d'une machine RAM qui étant donné $x, y \in \{0, 1\}$ calcule $x \vee y$.

iv. En déduire toute fonction booléenne $f : \{0,1\}^k \to \{0,1\}$ peut être calculée par une machine RAM.

Exercice 7. *Traduction RAM vers* WHILE

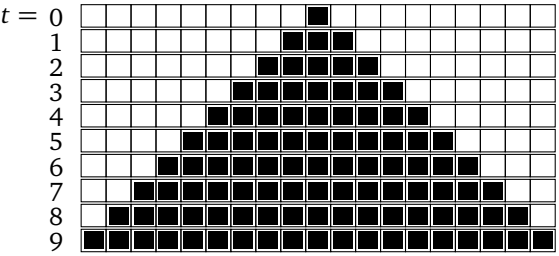
On veut montrer que tout programme en assembleur RAM peut être simulé par un programme LOOP. Pour cela, on considère un programme en assembleur RAM. Chaque instruction est l'une des suivante : COPY(R_i, R_j), INC(R_i), DEC(R_i), JUMP(R_i, ℓ) ou STOP. L'idée de la simulation est de représenter chaque registre R_i par une variable x_i dans le langage WHILE, et d'avoir une variable supplémentaire pc (pour *program counter*) qui retient l'instruction en cours. (La variable pc est aussi de la forme x_i pour un certain i, mais la notation permet d'être plus clair.)

1. Quelle doit être la valeur initiale de pc ?
2. Comment peut-on traduire une instruction du type COPY(R_i, R_j), INC(R_i) ou DEC(R_i) dans le langage WHILE ? *La valeur de pc doit être mise à jour.*
3. Comment traduire une instruction JUMP(R_i, ℓ) ?
4. Donner l'architecture globale du programme WHILE qui simule une machine RAM, en gérant en particulier l'instruction STOP.

Exercice 8. *Automates cellulaires unidimensionnels*

Un automate cellulaire (unidimensionnel) consiste en un ruban constitué d'une infinité de cases, chacune possédant un symbole d'un alphabet Σ, et d'une fonction de transition locale $f : \Sigma^3 \to \Sigma$ qui indique comment chaque cellule doit évoluer à chaque étape en fonction de son symbole et des symboles de ses deux voisines. Formellement, les cases du ruban sont indexées par les entiers relatifs : $R = (r_i)_{i\in\mathbb{Z}}$. Initialement, le ruban est dans la configuration initiale $R^{(0)} = (r_i^{(0)})_{i\in\mathbb{Z}}$. Une étape de calcul fait passer de la configuration $R^{(t)}$ à $R^{(t+1)}$ définie par $r_i^{(t+1)} = f(r_{i-1}^{(t)}, r_i^{(t)}, r_{i+1}^{(t)})$ pour tout i. On suppose que la fonction de transition est totale.

Exemple. On définit l'automate sur l'alphabet $\Sigma = \{0,1\}$ par $f(0,0,0) = 0$ et $f(a,b,c) = 1$ si $(a,b,c) \neq (0,0,0)$. On dessine ci-dessous les configurations $R^{(0)}$ à $R^{(9)}$ où $r_0^{(0)} = 1$ et $r_i^{(0)} = 0$ si $i \neq 0$. On représente $r_i^{(t)} = 0$ par une case blanche et $r_i^{(t)} = 1$ par une case noire.



1. On s'intéresse à des automates où $\Sigma = \{0,1\}$ et on utilise la même représenta-

tion que dans l'exemple. Pour chacune des tables de transitions, représenter les configurations $R^{(0)}$ à $R^{(9)}$ où $r_0^{(0)} = 1$ et $r_i^{(0)} = 0$ so $i \neq 0$.

i. $f(1,0,0) = f(0,0,1) = 1$ et $f(a,b,c) = 0$ sinon ;
ii. $f(0,1,0) = f(1,0,0) = f(0,1,1) = 1$ et $f(a,b,c) = 0$ sinon ;
iii. $f(1,0,0) = f(0,1,1) = f(0,1,0) = f(0,0,1) = 1$ et $f(a,b,c) = 0$ sinon.

Notre objectif est de démontrer l'équivalence entre le modèle des automates cellulaires unidimensionnels et les machines de Turing. Pour cela, on va faire deux hypothèses sur les automates cellulaires : d'une part, il existe un symbole spécial « blanc » $\square$ (le 0 dans nos exemples) et la configuration initiale ne contient qu'un nombre fini de symboles $\neq \square$ (l'entrée) ; d'autre part, $f(\square,\square,\square) = \square$.

2. Justifier l'intérêt de ces hypothèses.
3. Lister les principales similarités et différences entre une machine de Turing et un automate cellulaire.
4. On veut simuler une machine $\mathcal{M} = (Q, \Sigma, q_0, \delta)$. L'idée est que l'alphabet de l'automate cellulaire contienne non seulement Σ, mais également, pour chaque état $q \in Q$ et symbole $x \in \Sigma$, un symbole x^q.

i. Décrire formellement l'alphabet de l'automate cellulaire.
ii. Comment utiliser cet alphabet pour représenter la tête de lecture ?
iii. Considérons une transition $\delta(q,x) = (q', y, \leftrightarrow)$ de la machine de Turing. Traduire cette transition en des règles $f(x^q, \cdot, \cdot)$, $f(\cdot, x^q, \cdot)$ et $f(\cdot, \cdot, x^q)$, en fonction du type de déplacement.
iv. Comment doit-on définir $f(x,y,z)$ si x, y, $z \in \Sigma$?

5. La fonction de transition locale d'un automate cellulaire est totale, ce qui implique que l'automate ne termine jamais. La table de transition de la machine n'est pas totale, donc la question précédente laisse encore des transitions à compléter.

i. Supposons que $\mathcal{M}$ calcule une fonction f. Proposer une façon de compléter la fonction f pour la rendre totale, et une convention pour définir le résultat du calcul de l'automate cellulaire.
ii. Supposons que $\mathcal{M}$ reconnaît un langage L. Proposer une façon de compléter la fonction f pour la rendre totale, et une convention pour affirmer qu'un mot en entrée de l'automate cellulaire appartient ou non à L.

6. Expliquer comment simuler un automate cellulaire par une machine de Turing, toujours avec les deux hypothèses ci-dessus.

Exercice 9. *Nécessité des fonctions partielles*

L'entreprise Disrupt® propose son super nouveau langage *Halt!* avec la promesse qu'aucun programme écrit dans son langage ne peut rentrer dans une boucle infinie. Bien sûr, Disrupt® promet également qu'on peut programmer *tout ce qu'on veut* dans son langage, tant que ça ne boucle pas. C'est vraiment fantastique…

1. **i.** Traduire la promesse « aucun programme ne peut rentrer dans une boucle infinie » en une propriété sur les fonctions calculées par les programmes écrits en *Halt!*.
   ii. Justifier, intuitivement, que si on sait calculer une fonction $f : \mathbb{N} \to \mathbb{N}$ avec *Halt!*, alors on sait également calculer la fonction $n \mapsto f(n) + 1$.
   iii. Démontrer que l'ensemble des fonctions qu'on peut programmer avec *Halt!* est dénombrable.
2. Montrer que Disrupt® fait de la publicité mensongère : les trois propriétés de la question précédente impliquent qu'il existe une fonction *g intuitivement calculable* qu'on ne peut pas programmer avec *Halt!*. *Indication. Utiliser une preuve par diagonalisation.*
3. Dans la preuve précédente, qu'est-ce qui ne marche pas si le langage permet des fonctions qui bouclent ? Autrement dit, pourquoi cela ne démontre pas que pour tout langage, il existe un algorithme *intuitivement calculable* non programmable dans le langage ?

Exercice 10. *Rice ou pas Rice ?*

Pour chacun des problèmes suivants, déterminer si le théorème de Rice s'applique. Déterminer ensuite si le problème est décidable. Pour les questions (∗), déterminer également si le problème est reconnaissable, co-reconnaissable, ou aucun des deux.

(∗) **1.** Étant donné un programme dans votre langage préféré, est-ce que le programme affiche `J'adore la calculabilité.` ?
2. Étant donné un algorithme A, est-ce qu'il existe deux entrées x et y telles que $A(x) = A(y)$?
(∗) **3.** Étant donné un algorithme A, est-ce que A possède deux boucles imbriquées ?
4. Étant donné un algorithme A, est-ce que f_A est une fonction totale ?
5. Étant donné la fonction de transition d'un automate cellulaire et une configuration initiale (avec un nombre fini de cases non blanches), est-ce que l'automate atteint une configuration *stable*, qui n'évolue plus ?
(∗) **6.** Étant donné un algorithme A, est-ce que $L(A) = \emptyset$?
7. Étant donné un algorithme A, est-ce que pour tout $w \in L(A)$, $\overline{w} \in L(A)$? *Le mot $\overline{w}$ est le complémentaire de w obtenu en inversant chaque bit.*
8. Étant donné deux algorithmes A et B, est-ce que $A(w) = B(w)$ pour toute entrée (c'est-à-dire en particulier $A(w)\uparrow \iff B(w)\uparrow$) ?
(∗) **9.** Étant donné un automate fini déterministe A,

   i. est-ce que A accepte le mot 001100 ?
   ii. est-ce que le langage accepté par A est infini ?

Exercice 11. *Décidabilité et réductions*

1. **i.** Montrer qu'on peut simuler une machine de Turing $\mathcal{M}$ par une machine

qui, sur l'entrée w, ne visite jamais de case située à gauche la position initiale de sa tête.

ii. En déduire qu'étant donné une machine de Turing $\mathcal{M}$ et une entrée w, déterminer si elle visite une case située à gauche de la position initiale de sa tête pendant le calcul est indécidable.

2. Déterminer la décidabilité des problèmes suivants : étant donné une machine de Turing $\mathcal{M}$,

i. est-ce que $\mathcal{M}$ effectue plus de 2025 étapes de calcul sur l'entrée vide ?

ii. est-ce qu'il existe une entrée w sur laquelle $\mathcal{M}$ effectue plus de 2025 étapes de calcul ?

iii. est-ce que $\mathcal{M}$ effectue plus de 2025 étapes de calcul sur toute entrée ?

3. On considère un programme dans votre langage préféré. Montrer qu'on ne peut pas décider, étant donné le programme et une entrée x, si la ligne 17 du programme est exécutée au cours du calcul.

Exercice 12. *Valeurs numériques incalculables*

1. On définit la fonction *busy beaver* $\text{BB} : \mathbb{N} \to \mathbb{N}$ comme suit : pour un entier k, on considère toutes les machines de Turing à k états, qui terminent sur l'entrée vide ; soit $\text{BB}(k)$ le nombre maximal d'étapes nécessaires pour une de ces machines pour atteindre sa configuration d'arrêt. Autrement dit, toutes celles qui s'arrêtent le font en $\leq k$ étapes de calcul, et l'une le fait en exactement k étapes de calcul.

i. Montrer que si la fonction BB est calculable, on peut décider le problème de l'arrêt.

ii. Qu'en déduit-on sur la fonction BB ?

2. Pour un fichier f, on note $\text{TAILLE}(f)$ sa taille en octet. Étant donné votre langage de programmation préféré, on définit la *complexité*[1] $K(f)$ d'un fichier f comme la taille du plus petit programme qui, sans entrée, écrit le fichier f en mémoire. *La taille du programme est la taille de son fichier source.*

i. Justifier que $K(f) \leq \text{TAILLE}(f) + c$ où c est une constante qui ne dépend que du langage de programmation.

On veut montrer que $K(f)$ est incalculable. Supposons au contraire qu'on a un programme pour la fonction K.

ii. Montrer qu'on peut alors écrire un programme qui prend en entrée un entier k et écrit sur le disque un fichier f de complexité $\geq k$.

iii. En codant k en dur dans le programme, trouver une contradiction.

iv. En déduire qu'il n'existe aucun programme de compression sans perte optimal.

1. La notion théorique correspondante s'appelle *complexité de Kolmogorov*.

Exercice 13. *Un peu de théorie*

1. Montrer que L est décidable si et seulement si $\overline{L}$ est décidable.
2. Montrer que si un langage L est reconnaissable et si $L \leq_m \overline{L}$, alors L est décidable.
3. **i.** Montrer que L est reconnaissable si et seulement s'il est le *domaine de définition* d'un algorithme : il existe un algorithme A tel que $A(w)\downarrow$ si et seulement si $w \in L$.
   ii. Montrer que L est reconnaissable si et seulement s'il est l'*image* d'un algorithme : il existe un algorithme A tel que $L = \{A(x) : x \in \{0,1\}^*\}$.
4. Montrer que L est reconnaissable si et seulement si $L \leq_m K$ où K est le problème de l'arrêt.
5. Soit D un langage décidable, non vide et $\neq \{0,1\}^*$. Montrer que L est décidable si et seulement si $L \leq_m D$.
6. Soit L un langage reconnaissable. Montrer que si L contient exactement un mot de chaque longueur, alors L est décidable.
7. Soit L un langage énumérable par un algorithme E qui énumère les mots de L par ordre de taille, puis lexicographique (pour ceux de même taille). Montrer que L est décidable.

Exercice 14. *Calculabilité relative*

On imagine posséder une instruction magique pour résoudre le problème de l'arrêt : étant donné $\langle A\rangle$ et w, STOP?$(\langle A\rangle, w)$ renvoie 1 si $A(w)\downarrow$ et 0 si $A(w)\uparrow$. Cette instruction magique s'appelle l'*oracle* K (le problème de l'arrêt) et un algorithme qui l'utilise est un *algorithme avec oracle* K, noté A^K. On peut, comme pour un algorithme sans oracle, représenter A^K par un mot binaire $\langle A^K\rangle$.

1. Montrer que déterminer, étant donné $\langle A^K\rangle$ et un mot w, si $A^K(w)\downarrow$ est indécidable.
2. Le problème est-il décidable par des algorithmes à oracle ? Autrement dit, existe-t-il un algorithme à oracle H^K tel que $H^K(\langle A^K, w\rangle) = 1$ si $A^K(w)\downarrow$ et 0 si $A^K(w)\uparrow$?

Exercice 15. *Autres problèmes indécidables*

1. Déterminer si le problème de correspondance de Post est décidable dans les cas suivants :
   i. lorsque chaque domino possède deux mots de même longueur ;
   ii. lorsque l'alphabet utilisé ne possède qu'une seule lettre ;
   iii. lorsque l'alphabet utilisé possède deux lettres.
2. Le problème du pavage est-il décidable en dimension 1 ? *Précisions : on a en entrée un ensemble de tuiles possédant une couleur à gauche et une à droite, et on cherche à remplir une ligne complète en respectant les couleurs, étant donné une tuile de départ.*