Théorie de la complexité – Feuille de cours intégré n° 3
Machines de Turing

Nous allons introduire le **modèle de calcul** le plus célèbre en informatique théorique : les machines de Turing. De façon informelle, on peut voir la notion de « machine de Turing » comme un langage de programmation élémentaire. Syntaxiquement, **une machine de Turing est un programme qui prend en entrée un mot** (écrit sur un alphabet Σ qui est paramètre de la machine). Chaque mot w donne lieu à une exécution de la machine sur l'entrée w. Cette exécution peut se dérouler de trois façons différentes :

- Elle peut **terminer en acceptant** le mot d'entrée. Dans ce cas, un nouveau mot est renvoyé par la machine comme résultat de l'exécution.
- Elle peut **terminer en rejetant** le mot d'entrée. Dans ce cas, un nouveau mot est renvoyé par la machine comme résultat de l'exécution.
- Elle peut **ne pas terminer**. Dans ce cas, il n'y a pas de mot renvoyé.

On peut donc associer les objets sémantiques suivants à toute machine de Turing dont l'alphabet d'entrée est Σ :

1. Un langage sur Σ. Il contient les mots de Σ^* pour lesquels l'exécution de la machine termine en acceptant.
2. Une fonction **partielle** de Σ^* vers Σ^*. Pour chaque mot $w \in \Sigma^*$, elle associe le mot renvoyé par l'exécution de la machine sur l'entrée w si celle-ci termine. Si l'exécution ne termine pas, la fonction n'est pas définie sur l'entrée w (c'est pour cette raison qu'on parle de fonction *partielle*).

Remarque 1 - Les machines de Turing sont un modèle mathématique. Bien qu'on parle ici de « langage de programmation », l'introduction des machines de Turing n'est pas motivée par la programmation. La syntaxe des machines de Turing est tellement basique qu'on ne s'en sert jamais pour programmer.
Cependant, il est possible **d'implémenter n'importe quel algorithme** avec une machine de Turing. La classe des fonctions qui peuvent être implémentées par une machine de Turing est donc particulièrement intéressante : celle-ci contient exactement les fonctions réalisables par un ordinateur. On parle de fonction **calculable**. On se sert des machines de Turing pour étudier cette notion. En particulier, elles servent souvent d'outil mathématique qu'on utilise pour **montrer qu'une fonction donnée n'est pas calculable**.

1 Machines de Turing déterministes

Il existe de nombreuses variantes de la définition des machines de Turing. Nous allons en voir plusieurs. On commence par la variante la plus simple : les machines déterministes.

Definition - ❗ - Machine de Turing déterministe (syntaxe).

Une machine de Turing déterministe est un tuple $M = (\Sigma, \Gamma, Q, q_i, q_a, q_r, \delta)$ où :

- Σ est l'alphabet fini d'entrée. Il contient les symboles avec lesquels on écrit l'entrée donnée à la machine.
- Γ est l'alphabet fini de travail. Il étend Σ avec des symboles supplémentaires utilisés lors de l'exécution de la machine. On a donc $\Sigma \subseteq \Gamma$. De plus, Γ contient deux symboles spéciaux « $\square$ » et « \$ » qui n'appartiennent pas à Σ.
- Q est un ensemble fini d'états (aussi appelés états de contrôle).
- $q_i \in Q$ est l'état initial.
- $q_a \in Q$ et $q_r \in Q$ sont les états finaux appelés "état acceptant" et "état rejetant" respectivement.
- $\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright, \triangledown\}$ est la fonction de transition. Pour chaque paire (q, x) où $q \in Q \setminus \{q_a, q_r\}$ est un état non-final et $x \in \Gamma$ est une lettre, elle associe un triplet (r, y, d) où $r \in Q$ est un nouvel état, $y \in \Gamma$ une nouvelle lettre et $d \in \{\triangleleft, \triangleright, \triangledown\}$ est une direction.
  Cette fonction doit satisfaire la condition suivante pour la lettre spéciale $\$ \in \Gamma$: pour tout état q, on a $\delta(q, \$) = (r, \$, \triangleright)$ pour un certain état $r \in Q$.

On passe maintenant à la *sémantique*. Pour toute machine de Turing déterministe M et tout mot w sur son alphabet d'entrée, on va définir l'exécution de M sur w.

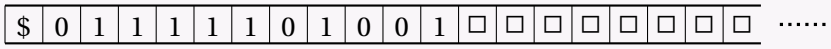
Definition - ❗ - Exécution d'une machine de Turing déterministe (sémantique).

On considère une machine de Turing $M = (\Sigma, \Gamma, Q, q_i, q_f, \delta)$. Pour tout mot d'entrée $w \in \Sigma^*$, on va définir l'exécution de M sur w. Celle-ci est nécessairement **unique**[a]. Une exécution de la machine M est une séquence (possiblement infinie) de **configurations**. On va commencer par définir cette notion préliminaire.

Configurations. Une configuration de la machine M est donnée par les trois éléments suivants :

- Un état de contrôle $q \in Q$.
- Une bande mémoire (infinie) B qu'on représente de la façon suivante :

Bande mémoire



  Chaque case mémoire contient une lettre de Γ. La première case (celle qui est la plus à gauche) contient nécessairement la lettre "$" (son rôle est de marquer cette case).
- Un entier $n \geqslant 0$ appelé **pointeur** ou **tête de lecture** qui désigne une case sur la bande mémoire (0 désigne la case la plus à gauche, contenant le symbole $, 1 désigne la case suivante, etc.).

Configurations finales. Une configuration est **finale** si l'état de contrôle q est l'un des deux états finaux de M (*i.e.*, si $q = q_a$ ou $q = q_r$). Une configuration finale peut être **acceptante** (si $q = q_a$) ou **rejetante** (si $q = q_r$).

Configuration suivante. Considérons une configuration $C = (q, B, n)$ de la machine M. Si q n'est **pas** un état final, alors on associe une unique configuration suivante à C. Celle-ci est définie en utilisant la fonction de transition δ de M. Soit $x \in \Gamma$ la lettre qui se trouve dans la n-ième case de la bande mémoire B (c'est-à-dire la case pointée par la tête de lecture). De plus soit, $(r, y, d) = \delta(q, x) \in Q \times \Gamma \times \{\nabla, \lhd, \rhd\}$. Alors, la configuration suivante de C est le triplet $C' = (q', B', n')$ où,

1. $q' = r$,
2. la bande mémoire B' est obtenue à partir de B en remplaçant la lettre x dans la n-ième case par la lettre y.
3. si $d = \lhd$, alors $n' = n - 1$, si $d = \nabla$, alors $n' = n$ et si $d = \rhd$ alors $n' = n + 1$. C'est-à-dire que la tête de lecture est déplacée d'un cran[b] dans la direction indiquée par d.

On notera $C \to C'$ pour indiquer que C' est la configuration suivante de C.

Configurations initiales. Enfin, pour tout mot *d'entrée* $w \in \Sigma^*$, on associe une *configuration initiale* de M qu'on notera $C[w]$. Soient $a_1, \ldots, a_n \in \Sigma$ les lettres telles que $w = a_1 \cdots a_n \in \Sigma^*$. On définit $C[w] = (q_i, B, 1)$ où q_i est l'état initial de M et B est la bande mémoire contenant le mot w suivi de symboles "□", représentée ainsi :

$C_0 =$

Exécutions de la machine M. Soit $w \in \Sigma^*$ un mot d'entrée pour M. L'exécution de M sur w est la séquence de configurations $C_0, C_1, \ldots, C_k, \ldots$ telle que $C_0 = C[w]$ est la configuration initiale de M pour le mot d'entrée w et et $C_i \to C_{i+1}$ pour tout indice i. Il y a trois situation possibles :

1. Il existe un indice $k \in \mathbb{N}$ tel que C_k est une configuration finale acceptante. Dans ce cas, l'exécution s'arrête à la configuration C_k et on dit que l'unique exécution de M sur w est **acceptante**.
2. Il existe un indice $k \in \mathbb{N}$ tel que C_k est une configuration finale rejetante. Dans ce cas aussi, l'exécution s'arrête à la configuration C_k et on dit que l'unique exécution de M sur w est **rejetante**.
3. Pour tout $k \in \mathbb{N}$, la configuration C_k n'est pas finale. Dans ce cas, l'exécution de M sur w **ne termine pas** : la séquence est infinie.

a. Nous définirons plus tard les machines non-déterministes pour lesquelles un même mot d'entrée peut donner lieu à plusieurs exécutions distinctes.

b. On notera que si la tête est en position 0, on lit nécessairement le symbole spécial « $ ». La condition sur la fonction de transition dans la définition garantit donc que dans ce cas, le symbole « $ » est préservé et que la tête est déplacée sur la droite.

Munis de cette définition, nous pouvons maintenant précisément associer un langage ainsi qu'une fonction partielle à toute machine de Turing déterministe.

Definition - ❗ - Langage accepté par une machine de Turing déterministe.

On considère une machine de Turing déterministe $M = (\Sigma, \Gamma, Q, q_i, q_f, \delta)$. Le **langage accepté par** M, noté $L(M) \subseteq \Sigma^*$, contient exactement les mots $w \in \Sigma^*$ tels que l'exécution de M sur w termine et est acceptante.

Exercice 1 – ❗ ☆ – Soit $M = (\Sigma, \Gamma, Q, q_i, q_a, q_r, \delta)$ une machine de Turing déterministe. Étant donné un mot $w \in \Sigma^*$, énoncer sans négation ce que veut dire la phrase « w n'appartient pas au langage $L(M)$ ».

Definition - ❗ - Fonction partielle calculée par une machine de Turing déterministe.

On considère une machine de Turing $M = (\Sigma, \Gamma, Q, q_i, q_f, \delta)$. La **fonction calculée par** M est la fonction partielle $f : \Sigma^* \to \Sigma^*$ définie comme suit pour tout mot d'entrée $w \in \Sigma^*$:

- Si l'exécution de M sur w ne termine pas, alors la fonction f n'est pas définie en w.
- Si l'exécution de M sur w termine, soit $C = (q, B, n)$ la configuration finale de cette exécution. Alors $f(w)$ est le plus grand mot dans Σ^* écrit à partir de la case 1 sur la bande mémoire B.

En pratique nous nous restreindrons souvent au sous-ensemble des machines de Turing qui terminent sur tous les mots d'entrée. Nous allons donc introduire une terminologie spéciale pour ces machines.

Definition - ❗ - Machines de Turing déterministes totales.

Une machine de Turing déterministe $M = (\Sigma, \Gamma, Q, q_i, q_a, q_r, \delta)$ est **totale** si pour tout mot d'entrée $w \in \Sigma^*$, l'exécution de M sur w termine.

Munis de ces définitions, nous pouvons maintenant introduire les notions fondamentales de la partie calculabilité du cours. On va définir ce qu'est un langage décidable ou semi-décidable et ce qu'est une fonction calculable.

Definition - ❗ - Langages décidables et langages semi-décidables.

Soit Σ un alphabet et $H \subseteq \Sigma^*$ un langage. On dit que,

- H est **semi-décidable** si il existe une machine de Turing déterministe M telle que $H = L(M)$.
- H est **décidable** si il existe une machine de Turing déterministe *totale* M telle que $H = L(M)$.

Definition - ❗ - Fonctions calculables.

Soit Σ un alphabet et $f : \Sigma^* \to \Sigma^*$ une fonction partielle. On dit que f **calculable** si il existe une machine de Turing déterministe M telle que f est la fonction calculée par M.

Remarque 2 - Fonctions calculées par une machine de Turing totale. Par définition, si une machine de Turing est totale, alors la fonction qu'elle calcule est totale : elle est définie sur tout son domaine. Les fonctions calculables qui sont totales sont donc exactement celles qui sont calculées par une machine totale.

Nous allons maintenant illustrer ces définitions avec des exercices.

Exercice 2 – ☆ – Soit $\Sigma = \{0, 1\}$ et $f : \Sigma^* \to \Sigma^*$ la fonction telle que pour tout $w \in \Sigma^*$, le mot $f(w)$ est la séquence obtenue à partir de w en transformant chaque 0 en 1 et vice-versa. Par exemple, $f(010110) = 101001$. Montrer que f est calculable.

Exercice 3 – ❗ ☆ – Soit Σ un alphabet. Montrer que pour toute fonction calculable $f : \Sigma^* \to \Sigma^*$, il existe une infinité de machines de Turing distinctes qui calculent f.

Exercice 4 – ☆ – Soit $\Sigma = \{0,1\}$. On utilise cet alphabet pour coder des entiers en binaire :

1. Montrer que le langage des mots codant un entier naturel pair est décidable.
2. On considère la fonction $mul_2 : \Sigma^* \to \Sigma^*$ définie par $mul_2(\varepsilon) = \varepsilon$ et pour tout mot non-vide $x \in \Sigma^* \setminus \{\varepsilon\}$, $mul_2(x)$ est égal au codage binaire de l'entier $2 \times x$. Montrer que mul_2 est calculable.

Exercice 5 – ★ – On utilise l'alphabet $\Sigma = \{a, b\}$. Montrer que le langage des palindromes est décidable.

Exercice 6 – ★ – Soit $\Sigma = \{0,1\}$ et $f : \Sigma^* \to \Sigma^*$ la fonction qui, pour tout mot $w \in \Sigma^*$ (par exemple 0101100), associe le miroir de w (par exemple 0011010). Montrer que f est calculable.

Exercice 7 – ❗ ☆ – Soit $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$. On considère le langage $L \subseteq \Sigma^*$ de l'ensemble des suites de 42 chiffres qui sont des facteurs de longueur 42 du développement décimal du nombre π. Ce langage est-il décidable ?

Exercice 8 – ★ – On utilise l'alphabet $\Sigma = \{0,1\}$ pour coder des entiers en binaire. On considère la fonction successeur $succ : \Sigma^* \to \Sigma^*$. C'est-à-dire que $succ(\varepsilon) = \varepsilon$ et pour tout mot non-vide $x \in \Sigma^* \setminus \{\varepsilon\}$, $succ(x)$ est égal au codage binaire de l'entier $x + 1$. Montrer que $succ$ est calculable.

Exercice 9 – ❗ ★ – Montrer que tout langage régulier est décidable.

Exercice 10 – ★ – On reprend l'exercice 4, mais cette fois, on utilise la représentation unaire des nombres. On utilise l'alphabet à une lettre $\Sigma = \{1\}$ pour coder des entiers en unaire. On considère la fonction $mul_2 : \Sigma^* \to \Sigma^*$ définie par $mul_2(\varepsilon) = \varepsilon$ et pour tout mot non-vide $x \in \Sigma^* \setminus \{\varepsilon\}$, $mul_2(x)$ est égal au codage unaire de l'entier $2 \times x$. Montrer que mul_2 est calculable.

Exercice 11 – ★ – On utilise l'alphabet $\Sigma = \{a, b, c\}$. Montrer que le langage des mots qui contiennent le même nombre de a, de b et de c est décidable. Montrer ensuite que le langage des mots qui ont la forme $a^n\, b^n\, c^n$ (où $n \in \mathbb{N}$ est un entier naturel quelconque) est également décidable.

2 Machines de Turing non-déterministes

Nous allons introduire une variante plus générale : les machines de Turing non-déterministes. Ici, le mot « non-déterministe » est utilisé avec le même sens que pour les automates. Dans une telle machine M, la fonction de transition est remplacée par un **ensemble de transitions**. Ainsi, une unique configuration de M pourra potentiellement donner lieu à plusieurs configurations suivantes distinctes. De même, M pourra avoir plusieurs exécutions distinctes sur le même mot d'entrée w.

Nous allons néanmoins voir que le non-déterministe ne rend pas les machines de Turing « plus puissantes » : les langages qui peuvent être acceptés par une machine non-déterministe sont exactement les mêmes que ceux qui peuvent être acceptés par une machine déterministe.

Remarque 3 - Fonctions et non-déterminisme. Puisqu'une machine non-déterministe peut avoir plusieurs exécutions pour le même mot d'entrée, y associer une fonction partielle n'a plus de sens. **On associera donc uniquement un langage à chaque machine** pour ce modèle.

Nous pouvons maintenant définir formellement les machines de Turing non-déterministes. La définition est syntaxiquement quasi-identique à celle des machines déterministes.

Definition - ❗ - Machine de Turing non-déterministe (syntaxe).

Une machine de Turing non-déterministe est un tuple $M = (\Sigma, \Gamma, Q, q_i, q_a, q_r, \delta)$ où :

- Σ est l'alphabet fini d'entrée. Il contient les symboles avec lesquels on écrit l'entrée donnée à la machine.
- Γ est l'alphabet fini de travail. Il étend Σ avec des symboles supplémentaires utilisés lors de l'exécution de la machine. On a donc $\Sigma \subseteq \Gamma$. De plus, Γ contient deux symboles spéciaux « $\square$ » et « \$ » qui n'appartiennent pas à Σ.
- Q est un ensemble fini d'états de contrôle.
- $q_i \in Q$ est l'état initial.
- $q_a \in Q$ et $q_r \in Q$ sont les états finaux appelés "état acceptant" et "état rejetant" respectivement.
- $\delta \subseteq (Q \setminus \{q_a, q_r\}) \times \Gamma \times Q \times \Gamma \times \{\triangleleft, \triangleright, \nabla\}$ est **l'ensemble de transitions**.
  Cet ensemble doit satisfaire la condition suivante pour la lettre spéciale $\$ \in \Gamma$. Pour tout tuple $(s, \$, r, y, d)$ contenu dans δ, on doit avoir $y = \$$ et $d = \triangleright$.

Nous pouvons maintenant étendre la notion d'exécution aux machines de non-déterministes.

Definition - Machines de Turing non-déterministes (sémantique).

On considère une machine de Turing non-déterministe $M = (\Sigma, \Gamma, Q, q_i, q_f, \delta)$. Pour tout mot $w \in \Sigma^*$, on associe une ou plusieurs exécution(s) de M sur w (il est aussi possible qu'il n'existe pas d'exécution de M sur w). La définition est basée sur la notion de **configuration** de la machine M qui reste inchangée par rapport aux machines déterministes puisque la définition ne dépendait pas des transitions. Nous allons par contre adapter la notion de « configuration suivante » au non-déterminisme.

Configurations suivantes. Considérons deux configurations $C_1 = (q_1, B_1, n_1)$ et $C_2 = (q_2, B_2, n_2)$ de M. On dit que C_2 est une *configuration qui suit* C_1 si il existe une transition $(q_1, x, q_2, y, d) \in \delta$ telle que,

1. les bandes B_1 et B_2 sont identiques sauf éventuellement sur la case n_1 : elle contient la lettre "x" dans B_1 et la lettre "y" dans B_2, et,
2. l'une des trois conditions suivantes est satisfaite :
   - $n_2 = n_1 - 1$ et $d = \triangleleft$, ou,
   - $n_2 = n_1$ et $d = \nabla$, ou,
   - $n_2 = n_1 + 1$ et $d = \triangleright$.

On notera $C_1 \to C_2$ pour indiquer que C_2 est une configuration qui suit C_1. De plus, on note $\xrightarrow{*}$ pour la clôture réflexive transitive de $\to$: étant données deux configurations C et C', on a $C \xrightarrow{*} C'$ si il existe $n \in \mathbb{N}$ et $C_0, \ldots, C_n$ telles que,

$$C = C_0 \to C_2 \to \cdots \to C_n = C'.$$

On appelle la séquence de configurations $C_0, \ldots, C_n$ une *exécution partielle* de M depuis la configuration $C = C_0$.

Exécutions de la machine M. Soit $w \in \Sigma^*$ un mot d'entrée pour M. Une exécution de M sur w est une séquence de configurations $C_0, C_1, \ldots, C_k, \ldots$ telle que $C_0 = C[w]$ est la configuration initiale de M pour le mot d'entrée w et $C_i \to C_{i+1}$ pour tout indice i. Il y a trois situation possibles :

1. Il existe un indice $k \in \mathbb{N}$ tel que C_k est une configuration finale acceptante. Dans ce cas, l'exécution s'arrête à la configuration C_k et on dit que cette exécution de M sur w est **acceptante**.
2. Il existe un indice $k \in \mathbb{N}$ tel que C_k est une configuration finale rejetante. Dans ce cas, l'exécution s'arrête à la configuration C_k et on dit que cette exécution de M sur w est **rejetante**.
3. Pour tout $k \in \mathbb{N}$, la configuration C_k n'est pas finale. Dans ce cas, l'exécution de M sur w **ne termine pas** : la séquence est infinie.

Remarque 4 - Attention. Étant donnés une machine de Turing non-déterministe M et un mot d'entrée w, il peut exister plusieurs exécutions de M sur w. À l'inverse, il se peut qu'il n'existe pas d'exécution de M sur w (par exemple si l'ensemble de transitions est vide, ce qui est autorisé par la définition).

Munis de cette définition, nous pouvons maintenant associer un langage à toute machine de Turing non-déterministe.

Definition - ❗ - Langage accepté par une machine de Turing non-déterministe.

On considère une machine de Turing non-déterministe $M = (\Sigma, \Gamma, Q, q_i, q_f, \delta)$. Le **langage accepté par** M, noté $L(M) \subseteq \Sigma^*$ contient exactement les mots $w \in \Sigma^*$ tels qu'**il existe** une exécution de M sur w qui est acceptante.

Exercice 12 – ❗ ☆ – Soit $M = (\Sigma, \Gamma, Q, q_i, q_a, q_r, \delta)$ une machine de Turing non-déterministe. Étant donné un mot $w \in \Sigma^*$, énoncer sans négation ce que veut dire la phrase « w n'appartient pas au langage $L(M)$ ».

Enfin, nous allons adapter la notion de « machine totale » aux machines non-déterministes.

Definition - ❗ - Machines de Turing non-déterministes totales.

Une machine de Turing non-déterministe $M = (\Sigma, \Gamma, Q, q_i, q_a, q_r, \delta)$ est **totale** si pour tout mot d'entrée $w \in \Sigma^*$, **toutes** les exécutions de M sur w terminent.

Nous allons maintenant pouvoir énoncer un résultat clé concernant les machines non-déterministes : sémantiquement, elles sont équivalentes aux machines déterministes.

Théorème 5

Soit $\Sigma \subseteq A^*$ un alphabet et $L \subseteq \Sigma^*$. Les deux propriétés suivantes sont équivalentes :

1. L est accepté par une machine de Turing déterministe.
2. L est accepté par une machine de Turing non-déterministe.

De même, les deux propriétés suivantes sont équivalentes :

1. L est accepté par une machine de Turing déterministe totale.
2. L est accepté par une machine de Turing non-déterministe totale.

Exercice 13 – ❗ ★★ – Montrer le Théorème 5.

Le Théorème 5 implique que les notions de décidabilité et de semi-décidabilité pour un langage ne dépendent pas du déterminisme. On peut les définir de façon équivalente en utilisant un modèle ou l'autre. Pour cette raison, dans ce contexte, on parle souvent de « machine de Turing » sans préciser si il s'agit d'une machine de Turing déterministe ou d'une machine de Turing non-déterministe.

Remarque 6 - Impact sur la complexité. Nous parlons pour l'instant de **calculabilité** : un langage donné est-il décidable ou non ? Dans ce contexte le déterminisme n'a pas d'impact. En revanche la distinction entre machines déterministes et non-déterministes sera cruciale plus tard lorsque nous parlerons de **complexité** (classifier les langages décidables selon les moyens qui sont nécessaires à leur résolution).

Exercice 14 – ★★½ – Montrer que tout langage hors-contexte est décidable.

3 Machines de Turing à plusieurs bandes

Nous allons introduire un dernier modèle qui rend la programmation un peu plus simple : les machines de Turing à plusieurs bandes. Ces machines peuvent être déterministes ou non. Elles restent équivalentes aux modèles de base en termes de sémantique.

Definition - Machines de Turing à plusieurs bandes (syntaxe).

Soit $k \geqslant 1$ un entier. Une machine de Turing (déterministe ou non-déterministe) à k bandes est un tuple $M = (\Sigma, \Gamma, Q, q_i, q_a, q_f, \delta)$. Ces objets sont similaires à ceux d'une machine classique. La différence est dans l'ensemble ou la fonction de transitions δ qui doit maintenant gérer k bandes (alors qu'une machine classique n'en gère qu'une seule).

1. Si M est déterministe, alors δ est une fonction :

$$\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma^k \to Q \times (\Gamma \times \{\triangleleft, \triangleright, \nabla\})^k.$$

   De plus, pour $q \in Q$ et $x_1, \ldots, x_k \in \Gamma$, si $(r, (y_1, d_1), \ldots, (y_k, d_k)) = \delta(q, x_1, \ldots, x_k)$ et $x_j = \$$ pour un certain $j \leqslant k$, alors on doit avoir $y_j = \$$ et $d_j = \triangleright$.

2. Si M est non-déterministe, alors δ est un ensemble :

$$\delta \subseteq (Q \setminus \{q_a, q_r\}) \times \Gamma^k \times Q \times (\Gamma \times \{\triangleleft, \triangleright, \nabla\})^k.$$

   De plus, pour tout $(q, (x_1, \ldots, x_k), r, (y_1, d_1), \ldots, (y_k, d_k)) \in \delta$ et pour tout $j \leqslant k$, si $x_j = \$$, alors $y_j = \$$ et $d_j = \triangleright$.

La sémantique des machines à plusieurs bandes est naturellement adaptée de celle des machines de Turing classiques. On va la décrire de façon informelle. les configurations d'une machine de Turing à k bandes contiennent k bandes mémoire et k têtes de lecture. Dans la configuration initiale, les k têtes valent 1, les k cases en position 0 contiennent le symbole \$, et le mot d'entrée est écrit sur la première bande à partir de la position 1 (les autres bandes ne contenant que des symboles $\square$ après le \$). Une transition $(q, (x_1, \ldots, x_k), r, (y_1, d_1), \ldots, (y_k, d_k))$ peut s'appliquer lorsque la machine est dans l'état q et que les lettres lues aux cases déterminées par les pointeurs sont $x_1, \ldots, x_k$ sur les bandes 1 à k. L'application de cette transition fait passer dans l'état r, remplace chaque x_i par y_i et modifie la i-ème tête de lecture selon la valeur de d_i, ce qui produit une *configuration suivante*.

Le résultat important sur les machines de Turing à plusieurs bandes est le suivant : leur pouvoir d'expression est le même que celle des machines classiques.

Théorème 7

Soit Σ un alphabet et $L \subseteq \Sigma^*$. Les deux propriétés suivantes sont équivalentes :

1. L est accepté par une machine de Turing.
2. Il existe $k \geqslant 1$ tel que L est accepté par une machine de Turing à k bandes.

Il existe un résultat similaire pour les machines déterministes et les fonctions calculables.

Théorème 8

Soit Σ un alphabet et $f : \Sigma^* \to \Sigma^*$ une fonction partielle. Les deux propriétés suivantes sont équivalentes :

1. f est la fonction calculée par une machine déterministe.
2. Il existe $k \geqslant 1$ tel que f est la fonction calculée par une machine déterministe à k bandes.

Exercice 15 – ★★☆ – Montrer les Théorèmes 7 et 8.

Le Théorème 7 implique que les machines de Turing à plusieurs bandes définissent aussi les notions de décidabilité d'un langage et de calculabilité d'une fonction. On pourra donc les utiliser librement.

Exercice 16 – ★☆ – On utilise l'alphabet $\Sigma = \{0, 1\}$. Refaire l'exercice des palindromes avec une machine à deux bandes qui n'écrit pas sur sa première bande (l'entrée n'est jamais modifiée).

**Exercice 17 – – On utilise l'alphabet $\Sigma = \{0,1,\#\}$ et on code des entiers en binaire avec $\{0,1\}$. Décrire une machine déterministe qui calcule la fonction addition $add : \Sigma^* \to \Sigma^*$. C'est-à-dire que pour tout mot w de la forme $x\#y$ où $x, y \in \{0,1\}^* \setminus \{\varepsilon\}$ codent deux entiers (par exemple, 101#11, pour $x = 5$ et $y = 3$), $add(w)$ est le codage en binaire de l'entier $x + y$ (par exemple 1000 pour $x + y = 8$).

**Exercice 18 – – On utilise l'alphabet $\Sigma = \{0,1\}$ pour coder des entiers. Décrire une machine déterministe qui transforme les codages unaires en codages binaires (par exemple 11111 en 101).

**Exercice 19 – – On utilise l'alphabet $\Sigma = \{0,1\}$ pour coder des entiers. Décrire une machine déterministe qui calcule la fonction exponentielle $f : x \mapsto 2^x$, c'est-à-dire qui prend en entrée un entier naturel x codé en binaire et renvoie en sortie le codage binaire de 2^x.

**Exercice 20 – – On utilise l'alphabet $\Sigma = \{0,1,\#\}$ et on code des entiers en binaire avec $\{0,1\}$. Décrire une machine déterministe qui calcule la fonction multiplication $mul : \Sigma^* \to \Sigma^*$. C'est-à-dire que pour tout mot w de la forme $x\#y$ où $x, y \in \{0,1\}^* \setminus \{\varepsilon\}$ codent deux entiers (par exemple 101#11, pour $x = 5$ et $y = 3$), $mul(w)$ est le codage en binaire de l'entier $x \times y$ (par exemple 1111 pour $x \times y = 15$).

**Exercice 21 – – On utilise l'alphabet $\Sigma = \{0\}$. Décrire une machine de Turing qui accepte les mots dont la longueur est un carré.

**Exercice 22 – – On considère une variante des machines de Turing classiques (*i.e.*, à une seule bande) qui utilise une bande bi-infinie. On supprime le symbole spécial \$ et les contraintes associées sur la fonction de transition, de sorte que les machines peuvent lire et écrire aussi loin que nécessaire vers la gauche. Montrer que ces machines peuvent être simulées par les machines classiques.