Calculabilité et Complexité (11X008) - Printemps 2026

# 1. Problèmes & Machines

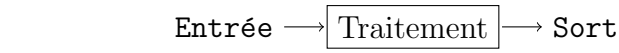
*Enseignant: Arnaud Casteigts*

*Assistants: M. De Francesco, M. Marseloo*
*Moniteurs: E. Bussod, N. Beghdadi*

Pour la première partie de cette séance, voir les slides de survol du module. Ici, nous mentionnons les différents types de problèmes qu'un algorithme peut résoudre. Puis, nous rappelons la définition des machines de Turing (avec un modèle un peu différent) ainsi qu'un exemple. Puis, nous expliquons le fonctionnement d'une machine universelle, capable de simuler une autre machine dont la description lui est passée en entrée.

## 1.1 Qu'est-ce qu'un problème ?

L'informatique est la science du traitement de l'information. Un traitement consiste à recevoir une entrée et produire une sortie qui dépend de cette entrée, les deux pouvant être vus comme des mots sur un certain alphabet $\Sigma$, par exemple l'alphabet binaire $\Sigma = \{0, 1\}$.




De manière générale, on appelle *problème* la spécification du traitement à effectuer, c'est à dire la relation à satisfaire entre l'entrée et la sortie. Formellement, cela correspond à une fonction mathématique $f$ du domaine des entrées vers le domaine des sorties. On distingue souvent quatre types de problèmes, à savoir les problèmes de *décision*, de *recherche*, d'*optimisation* et de *dénombrement*. Illustrons-les par un exemple où l'entrée consiste en une carte routière et deux points A et B, le tout encodé sur l'alphabet $\{0, 1\}$ :

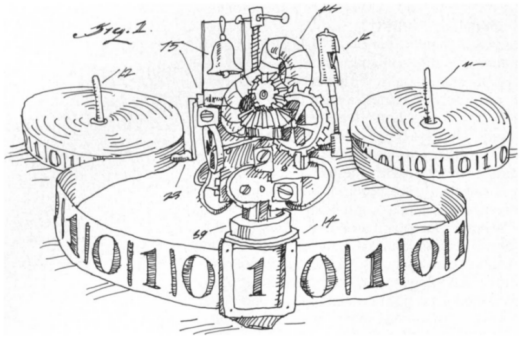
- Problème de *décision* $f : \{0,1\}^* \to \{0,1\}$
  *Existe-t-il un chemin de A vers B ?*
- Problème de *recherche* $f : \{0,1\}^* \to \{0,1\}^*$
  *Trouver un chemin de A vers B.*
- Problème d'*optimisation* $f : \{0,1\}^* \to \{0,1\}^*$
  *Trouver le plus court chemin de A vers B.*
- Problème de *dénombrement* $f : \{0,1\}^* \to \{0,1\}^*$
  *Combien y a-t-il de chemins possibles de A vers B ?* (domaine des sorties interprété comme $\mathbb{N}$)

Quand on étudie la calculabilité ou la complexité des problèmes, on se rend vite compte que la plupart des questions peuvent se formuler comme des problèmes de décision. Par exemple, pour compter le nombre de chemins, on pourrait étudier le problème "Existe-t-il au moins $k$ chemins ?" et y faire appel plusieurs fois en faisant varier la valeur de $k$. On se concentre donc naturellement sur les problèmes de décision.

Pour cette raison, nous utiliserons souvent "problème" et "langage" de manière interchangeable. Dans la terminologie des problèmes, l'entrée est appelée une **instance** du problème. Si cette instance induit une réponse oui (autrement dit, si elle fait partie du langage), alors on dit que c'est une instance positive. Sinon, c'est une instance négative.

## 1.2 Machine de Turing

La machine de Turing est un modèle mathématique d'ordinateur très simple, qui a supposément la même *expressivité* : tout traitement d'information physiquement réalisable peut être effectué par une machine de Turing (conjecture de Church-Turing). Formellement, une machine de Turing est constituée d'une partie de contrôle (proche d'un automate fini) et d'une mémoire infinie sur laquelle elle peut se déplacer, lire et écrire librement. Il existe différents modèles, selon que la machine utilise une ou plusieurs bandes, qu'elle utilise un seul ou plusieurs états finaux, qu'elle soit déterministe ou non, etc. Tous ces modèles sont équivalents en termes de *calculabilité*, car ils peuvent *se simuler* les uns les autres.



### 1.2.1 Définition mathématique

Dans la partie calculabilité du cours, nous utiliserons une version déterministe à $k$ bandes définie comme suit :

**Définition 1.1.** Une machine de Turing est un 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_{start}, q_{halt})$ tel que :

- $Q$ est un ensemble fini d'états, comprenant au minimum l'état $q_{start}$ et l'état $q_{halt}$.
- $\Sigma$ est l'alphabet d'entrée

- $\Gamma$ est l'alphabet de la machine (qui inclut $\Sigma$)
- $\delta$ est une fonction de transition de la forme générale :

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$$

Dans le cas d'une machine à une bande, cela correspond à la fonction

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}.$$

Lorsqu'on utilise une machine à plusieurs bandes, on suppose généralement que $k \geq 3$, avec (dans cet ordre) une bande d'entrée, $k-2$ bandes "de travail", et une bande de sortie. On appelle les bandes $T_1, ...T_k$ (T pour *tape*, en anglais).

Par convention, nous supposerons que chaque bande n'est infinie que d'un côté (à droite). La première case de chaque bande (sur laquelle pointe la tête de lecture initialement) est marquée du symbole spécial $\triangleright$. Le mot d'entrée se trouve à droite de ce symbole sur la bande d'entrée. Initialement, toutes les autres cases de toutes les bandes sont vides (symbole spécial $\square$). Les symboles $\triangleright$ et $\square$ font donc toujours partie de $\Gamma$.

Le fonctionnement de la machine est similaire à ce que l'on a vu au premier semestre : la machine est initialement dans l'état $q_{start}$. À chaque étape, son action est définie par la fonction $\delta$, et dépend donc de l'état courant et des symboles courants sur chaque bande. Son action consiste alors à choisir un nouvel état (potentiellement le même), à écrire un symbole sur chaque bande (potentiellement, le même), et à déplacer (ou non) chacune des têtes de lecture. Quand la machine a terminé et que son résultat est écrit sur la bande de sortie, elle se déplace sur l'état $q_{halt}$, ce qui termine l'exécution. Pour les problèmes de décisions, la sortie consiste en un seul symbole 1 ou 0 (selon que la réponse est oui ou non), ce qui est équivalent à d'autres modèles qui ont des états dédiés $q_{accept}$ et $q_{reject}$.

### 1.2.2 Exemple : Palindromes (revisité)

Pour rappel, $w^R$ désigne le mot $w$ écrit à l'envers. Si $w = w^R$, alors $w$ est un *palindrome*. Étant donné un alphabet $\Sigma$, le langage des palindromes sur $\Sigma$ est $L = \{w \in \Sigma^* \mid w = w^R\}$.

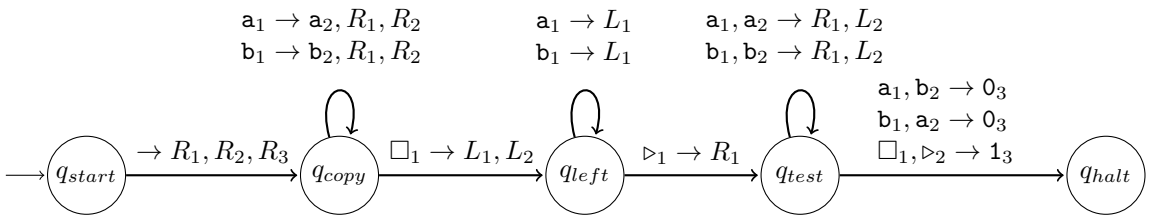
Nous avons déjà vu un exemple de machine de Turing décidant ce langage. Nous allons voir comment l'utilisation de plusieurs bandes permet de simplifier le traitement. La stratégie est simple :

1. Recopier le mot d'entrée (qui est sur $T_1$) sur la bande de travail $T_2$.
2. Se placer sur le premier symbole du mot d'entrée sur $T_1$, et sur le dernier symbole du mot recopié sur $T_2$.
3. Comparer les deux symboles. Si le premier vaut $\square$ et le second vaut $\triangleright$, alors on a terminé, on écrit donc `1` sur $T_3$ et on va dans l'état $q_{halt}$. Sinon, si les deux symboles

sont différents, on écrit `0` sur $T_3$ et on termine. Sinon, on se déplace à droite sur $T_1$ et à gauche sur $T_2$, et on recommence l'étape 3.

Voyons une spécification plus détaillée de cette machine. Les alphabets sont $\Sigma = \{a, b\}$ et $\Gamma = \{\triangleright, \square, \mathtt{a}, \mathtt{b}, \mathtt{0}, \mathtt{1}\}$. Nous avons besoin ici de 5 états : $q_{start}$, $q_{halt}$ et trois autres états qui nous servent à (1) recopier le mot d'entrée sur $T_2$, (2) se repositionner tout à gauche de $T_1$ et (3) effectuer la comparaison symbole après symbole. Appelons ces états $q_{copy}$, $q_{left}$ et $q_{test}$.

Pour simplifier les notations, nous noterons $\mathtt{s}_i$ pour désigner un symbole $\mathtt{s}$ lu ou écrit sur la $i^{\text{ème}}$ bande. De même pour les mouvements. Ce qui n'est pas indiqué reste inchangé.




**Exécution.** Voyons quelques étapes de l'exécution sur le mot d'entrée `abba`. On utilise un accent circonflexe pour indiquer où la tête de lecture se trouve sur chaque bande.

Initialement, dans l'état $q_{start}$ :

| | |
|---|---|
| 1 | $\hat{\triangleright}\,\mathtt{a}\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}\,\square\,\square\cdots$ |
| 2 | $\hat{\triangleright}\,\square\,\square\cdots$ |
| 3 | $\hat{\triangleright}\,\square\,\square\cdots$ |

En arrivant dans l'état $q_{copy}$ :

| | |
|---|---|
| 1 | $\triangleright\,\hat{\mathtt{a}}\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}\,\square\,\square\cdots$ |
| 2 | $\triangleright\,\hat{\square}\,\square\cdots$ |
| 3 | $\triangleright\,\hat{\square}\,\square\cdots$ |

En arrivant dans l'état $q_{left}$ :

| | |
|---|---|
| 1 | $\triangleright\,\mathtt{a}\,\mathtt{b}\,\mathtt{b}\,\hat{\mathtt{a}}\,\square\,\square\cdots$ |
| 2 | $\triangleright\,\mathtt{a}\,\mathtt{b}\,\mathtt{b}\,\hat{\mathtt{a}}\,\square\,\square\cdots$ |
| 3 | $\triangleright\,\hat{\square}\,\square\cdots$ |

En arrivant dans l'état $q_{test}$ :

| | |
|---|---|
| 1 | $\triangleright\,\hat{\mathtt{a}}\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}\,\square\,\square\cdots$ |
| 2 | $\triangleright\,\mathtt{a}\,\mathtt{b}\,\mathtt{b}\,\hat{\mathtt{a}}\,\square\,\square\cdots$ |
| 3 | $\triangleright\,\hat{\square}\,\square\cdots$ |

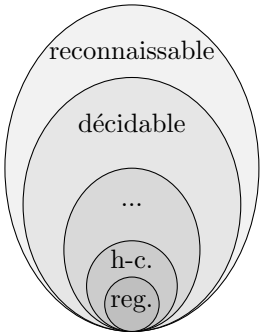
En arrivant dans l'état $q_{halt}$ :

| | |
|---|---|
| 1 | $\triangleright\,\mathtt{a}\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}\,\hat{\square}\,\square\cdots$ |
| 2 | $\hat{\triangleright}\,\mathtt{a}\,\mathtt{b}\,\mathtt{b}\,\mathtt{a}\,\square\,\square\cdots$ |
| 3 | $\triangleright\,\hat{\mathtt{1}}\,\square\cdots$ |

## 1.3 Décidable versus reconnaissable (rappels)

Comme nous l'avons déjà vu plusieurs fois, les machines de Turing ne terminent pas toujours, ce qui impose de distinguer deux familles distinctes : les langages décidables et les langages reconnaissables.

Un langage $L$ est **décidable** s'il existe une machine $M$ qui accepte les mots de $L$ et rejette les mots de $\overline{L}$. Un langage $L$ est **reconnaissable** s'il existe une machine $M$ qui accepte les mots de $L$ et n'accepte pas les mots de $\overline{L}$ (en rejettant ou en bouclant). Un langage décidable est donc aussi reconnaissable, mais l'inverse n'est pas forcément vrai.




Synonymes :
- décidable : *récursif*
- reconnaissable : *récursivement énumérable*, *semi-décidable*.

Certains langages sont reconnaissables, mais pas décidables. Par exemple, le langage de l'arrêt $L_H = \{(\langle M \rangle, w) \mid M \text{ termine sur l'entrée } w\}$ n'est pas décidable, mais il est reconnaissable : il suffit d'utiliser une machine universelle $M_U$ qui simule $M$ sur l'entrée $w$ et se contente de répondre OUI lorsque la simulation se termine. Si la simulation ne termine pas, $M_U$ ne termine pas non plus, mais ce n'est pas un problème pour *reconnaître* $L_H$.

Au cours du semestre, nous reviendrons plusieurs fois sur la distinction entre reconnaître un langage et décider un langage.