Calculabilité et Complexité (11X008) - Printemps 2025

2. Problèmes, Machine, Simulation

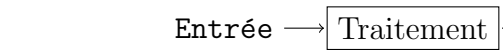
Enseignant: Arnaud Casteigts

Assistants: A. Berger, M. De Francesco
Moniteurs: E. Bussod, N. Beghdadi

Le premier cours correspond à la présentation donnée en amphithéâtre (slides sur la page du cours). Dans ce deuxième cours, nous définissons différents types de problèmes informatiques. Puis, nous rappelons la définition des machines de Turing (avec 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.

2.1 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 Σ, 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 depuis le 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 la description d'une carte routière et deux points A et B sur l'alphabet $\{0, 1\}^*$:

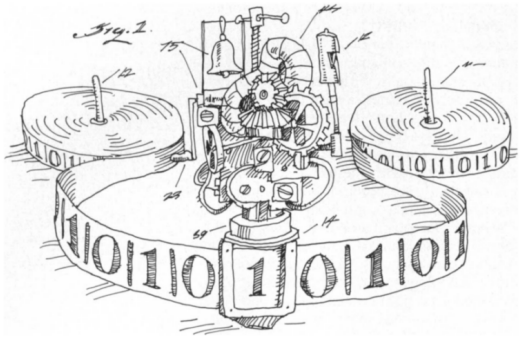
- Problème de *décision* $f : \{0,1\}^* \to \{0,1\}$
  Existe-t-il un chemin de A vers B ?
- Problème de *recherche* $f : \{0,1\}^* \to \{0,1\}^*$
  Trouver un chemin de A vers B.
- Problème d'*optimisation* $f : \{0,1\}^* \to \{0,1\}^*$
  Trouver le plus court chemin de A vers B.
- Problème de *dénombrement*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

Quand on étudie la calculabilité ou la complexité des problèmes, on se rend vite compte que la plupart des questions peuvent se réduire à une variante décisionnelle du problème. Par exemple, pour compter le nombre de chemin, 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 décisions.

Pour les problèmes de décisions, le but est de répondre OUI (1) ou NON (0) pour une entrée donnée, appelée ici une *instance* du problème. L'ensemble des entrées pour lesquelles la réponse est OUI (*instances positives*) forment donc un langage formel, et résoudre le problème revient à décider ce langage au même sens que nous l'entendions au premier semestre.

2.2 Machines de Turing

Les machines de Turing sont un modèle mathématique d'ordinateur très simple, qui a la même *expressivité* : tout traitement qu'un algorithme est capable d'effectuer peut être effectué par une machine de Turing et vice versa (conjecture de Church-Turing). Formellement, une machine de Turing est constituée 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, différents types d'alphabets, etc. Tous ces modèles sont équivalents en termes de calculabilité, car ils peuvent *se simuler* les uns les autres.



Comme pour les automates, on peut considérer des machines de Turing déterministes, où un seul choix est possible à chaque étape, ou des machines de Turing non-déterministes, où plusieurs choix sont possibles et ces choix sont effectués simultanément dans des univers parallèles (on parle de *branchement* non-déterministe). Dans le cas d'un problème de décision, on dira qu'une machine non-déterministe répond OUI (accepte) si au moins une de ses branches d'exécution accepte. En termes de calculabilité, le non-déterminisme n'augmente pas les capacités : une machine non-déterministe peut être simulée par une machine déterministe. La situation est différente en termes de complexité, nous y reviendrons.

2.2.1 Définition mathématique

Le modèle que nous utiliserons ce semestre est le suivant (version déterministe) :

Définition 2.1. Une machine de Turing est un triplet $M = (\Gamma, Q, \delta)$ tel que :

- Γ est l'alphabet de la machine (qui inclut l'alphabet d'entrée Σ)
- Q est un ensemble fini d'états, comprenant au minimum l'état q_{start} et l'état q_{halt}.
- δ est une fonction de transition de la forme $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$, où k est le nombre de bandes.

Autrement dit, selon l'état actuel de l'automate et le symbole qui se trouve sur chaque bande, on change d'état et on remplace potentiellement le symbole courant sur chaque bande, puis on déplace la tête de lecture de chaque bande (potentiellement, de manière différente). Mathématiquement, Γ^k signifie $\Gamma \times \Gamma \times \cdots \times \Gamma$ (un symbole pour chaque bande). De même, $\{L, R, S\}^k$ représente k mouvements de type left, right, stay (un pour chaque bande).

Les problèmes que nous considèrerons ne sont pas tous des problèmes de décisions (parfois, la machine calcule des choses aussi), nous remplaçons donc les états q_{accept} et q_{reject} par un état plus général, appelé q_{halt}, qui termine l'exécution. Si l'on considère un problème de décision, on peut écrire 0 ou 1 sur la bande de sortie pour signifier le rejet ou l'acceptation avant de terminer en allant sur l'état q_{halt}, ce qui est équivalent à q_{accept} et q_{reject}.

Par convention, nous supposerons que la première case de chaque bande est toujours marquée du symbole $\triangleright$, qui indique le début de la bande. 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 $\square$).

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 δ, 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 écrit son résultat sur la bande de sortie, elle se déplace sur l'état q_{halt}, ce qui termine l'exécution.

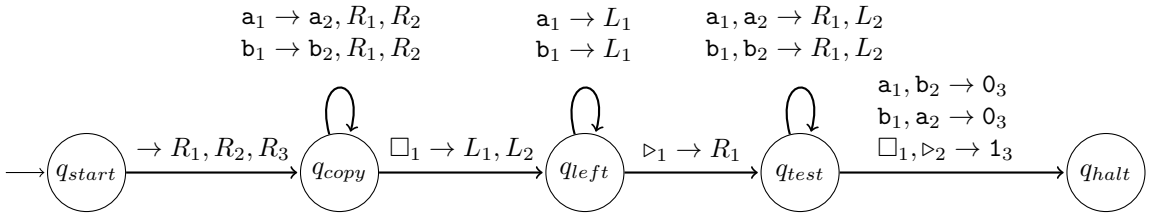
2.2.2 Exemple : Palindromes (reloaded)

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 Σ, le langage des palindromes sur Σ est $L = \{w \in \Sigma^* \mid w = w^R\}$.

Nous avons déjà vu un exemple de machine de Turing reconnaissant ce langage, en utilisant une bande et huit états. Nous allons voir comment l'utilisation de plusieurs bandes simplifie les choses. 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 $\mathtt{1}$ sur T_3 et on va dans l'état q_{halt}. Sinon, si les deux symboles sont différents, on écrit $\mathtt{0}$ 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. L'alphabet est $\Gamma = \{\triangleright, \square, \mathtt{0}, \mathtt{1}\}$. 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 $\mathtt{abba}$. 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	$\triangleright\,\hat{\mathtt{1}}\,\square\cdots$

2.3 Description d'une machine et simulation

Jusqu'à présent, nous avons supposé qu'une machine de Turing peut en simuler une autre, si on lui donne sa description. Mais nous n'avons pas vu comment le faire. Et d'ailleurs, comment représenter une machine pour la donner en entrée à une autre ?

2.3.1 Encodage d'une machine

Pour simuler une machine M, il faut déjà se mettre d'accord sur la description textuelle $\langle M \rangle$, pour qu'une autre machine puisse prendre cette description en entrée. Concentrons-nous sur le cas le plus simple, où M n'utilise qu'une seule bande et a pour alphabet $\Gamma = \{0, 1, \triangleright, \square\}$. Les autres descriptions sont analogues (mais forcément plus complexes).

La machine M a un certain nombre d'états $Q = \{q_1, q_2, \dots\}$ et sa fonction de transition δ correspond à une liste de transitions telles que représentées dans le tableau suivant :

État de départ	Symbole lu	État d'arrivée	Symbole écrit	Mouvement
q_1	$\triangleright$	q_2	1	R
q_2	0	q_1	$\square$	L
...	...	...	...	...

Ici, par exemple, si M est dans l'état q_2 et lit le symbole 0, alors M efface ce symbole, va dans l'état q_1 et déplace la tête de lecture vers la gauche. Nous allons encoder une telle transition en un mot composé de 5 parties (un facteur pour chaque colonne) dont les valeurs n'utilisent que des 0 (encodage *unaire*), le tout séparés par des 1.

Par exemple, on peut encoder l'état q_1 par 0, q_2 par 00, q_3 par 000, et plus généralement q_i par 0^i. De même pour l'alphabet, on encode 0 par 0, 1 par 00, $\triangleright$ par 000 et $\square$ par 0000. Enfin, on encode les mouvements L par 0, R par 00 et S par 000. À l'arrivée, une transition telle que celle ci-dessus peut être encodée comme :

$$(q_2, 0, q_1, \square, L) = 0010101000010$$

En fait, la machine M peut être entièrement décrite par ses transitions. On pourrait penser que ce n'est pas suffisant, mais ces dernières contiennent bel et bien toute l'information nécessaire pour décrire M. En particulier, on peut y découvrir quels sont les états de M, en supposant *par convention* que q_1 (donc 0) désigne toujours l'état de départ (q_{start}) et q_2 (donc 00) désigne toujours l'état final (q_{halt}).

Il suffit donc, pour décrire M intégralement, d'enchaîner les encodages de chacune de ses transitions, séparées (par exemple) par le facteur 11, en y ajoutant un délimiteur 111 au tout début et à toute la fin de la description globale, par exemple :

`111`	`0010101000010`	`11`	`01000100100100`	`11`	$\cdots$	`111`
début	transition 1		transition 2			fin

2.4 Machine de Turing universelle

Une **Machine de Turing universelle** est une machine M_U qui prend en entrée la description $\langle M \rangle$ d'une autre machine M et un mot d'entrée w pour M, et qui produit la même sortie que $M(w)$. Autrement dit, $M_U(\langle M \rangle, w) = M(w)$. On dit que M_U simule l'exécution de M sur l'entrée w.

Ce fonctionnement est comparable à nos ordinateurs, à qui l'on donne un programme et une entrée pour ce programme, et qui "simulent" l'exécution de ce programme en utilisant leurs circuits électroniques.

Pour simplifier, nous allons expliquer le fonctionnement d'une machine universelle M_U à 4 bandes (on pourrait le faire avec une seule bande, mais ce serait plus compliqué). L'entrée de cette machine est une description $\langle M \rangle$ d'une machine à une bande et un mot w. Comme vu précédemment, $\langle M \rangle$ se présente sous la forme 111 t_1 11 t_2 11 $\cdots$ 11 t_ℓ 111, où t_i correspond à l'encodage de la $i^{\text{ème}}$ transition de M (qui en a ℓ au total).

Notre machine M_U utilise les 4 bandes suivantes : une d'entrée (T_1), deux de travail (T_2 et T_3) et une de sortie (T_4). Au départ, le mot $\langle M \rangle w$ se trouve sur la bande d'entrée T_1 de M_U. La bande T_2, quant à elle, simulera l'unique bande de M. La première action de M_U est donc de recopier w sur T_2. Il suffit pour cela d'avancer sur T_1 jusqu'à la fin du deuxième facteur `111` et recopier la suite sur T_2. La bande T_3 nous sert à stocker l'état courant de M. On y écrit donc `0` pour commencer, ce qui correspond bien à l'état de départ de M. L'initialisation est terminée, nous sommes dans la configuration suivante :

T_1 (entrée de M_U)	111 t_1 11 t_2 11 $\cdots$ 11 t_k 111 w
T_2 (entrée de M)	w
T_3 (état de M)	`0`
T_4 (sortie de M)	

La machine M_U entre alors dans une boucle dont chaque itération simule un pas de calcul de M comme suit :

1. Chercher sur T_1 une transition $u1v1x1y1z$ telle que u correspond au contenu de T_3 et v correspond au symbole courant sur T_2. (v est encodé par des 0 et T_2 ne l'est pas, on compare donc v à l'encodage de ce symbole.)
2. Remplacer le contenu de T_3 par x et remplacer le symbole courant de T_2 par y (en le décodant).
3. Déplacer la tête de lecture de T_2 selon la valeur de z.

4. Si le contenu de T_3 est égal à 00 (état final), on recopie T_2 sur la sortie et on va dans l'état q_{halt}. Sinon, on recommence.

Chaque itération de la simulation (étapes 1 jusqu'à 4) correspond au fait d'exécuter *une* transition de M, dont la bande correspond à T_2. Par ailleurs, M_U termine en produisant le contenu de cette bande dès que M entre dans son état final. On a donc bien $M_U(\langle M \rangle w) = M(w)$. Bien sûr, certaines étapes pourraient être plus détaillées (notamment, l'encodage/décodage), mais cela est suffisant pour comprendre le principe de la simulation.

Notez que nous avons supposé ici que la description $\langle M \rangle$ est toujours valide. Notamment, le comportement de M est entièrement spécifié. Si l'on ne souhaite pas faire cette hypothèse, on peut effectuer une vérification de la validité de $\langle M \rangle$ avant de commencer la simulation.

2.4.1 Autres résultats du même genre

Il existe d'autres techniques similaires, notamment :

1. Une machine utilisant un alphabet arbitraire peut être simulée par une machine utilisant l'alphabet minimal $\{\mathtt{0}, \mathtt{1}, \triangleright, \square\}$.
2. Une machine utilisant plusieurs bandes peut être simulée par une machine n'utilisant qu'une seule bande.
3. Une machine non-déterministe peut être simulée par une machine déterministe.

Ces résultats impliquent qu'il suffit de pouvoir simuler une machine déterministe à une bande sur l'alphabet $\{\mathtt{0}, \mathtt{1}, \triangleright, \square\}$ pour être capable de simuler n'importe quelle autre machine. Lorsqu'un nouveau modèle de calcul est inventé, il suffit donc de montrer qu'il peut faire cela pour lui faire rejoindre le club des modèles *Turing-complets*. De nombreux modèles Turing-complets existent, par exemple :

- Le λ-calcul (lambda calcul), qui est un système formel de calcul basé, entre autres, sur des règles de substitution et qui sert de fondements à la programmation récursive.
- Nos ordinateurs actuels, ou plutôt leur modèle mathématique, les *machines RAM* (random-access machine), qui permettent d'accéder à des cellules mémoires directement (sans déplacer une tête de lecture) et utilisent des registres.
- Le modèle de Post et le modèle de Wang.

On peut aussi trouver des modèles plus farfelus comme :

- Modèles à base d'ADN : on peut coder une instance du problème avec des brins d'ADN et les manipuler par les outils classiques de la biologie moléculaire pour simuler les opérations qui isoleront la solution. (Certains de ces modèles sont Turing-complets.)
- Les origamis : encoder n'importe quel traitement sous forme de pliage de papier.

Où que l'on regarde dans la nature, les modèles de calculs que l'on trouve semblent être ni plus ni moins puissants que les machines de Turing. Cela inclut aussi les ordinateurs quantiques (même si ces derniers sont probablement différents en termes de *complexité*).