Calculabilité et Complexité (11X008) - Printemps 2024

2. Rappels de langages formels

Enseignant: Arnaud Casteigts *Exercices: A. Berger, M. De Francesco, L. Heiniger*

Le premier cours correspond à la présentation donnée en amphithéâtre, disponible dans le fichier `complexite-1-slides.pdf` [1].

Dans ce deuxième cours, nous effectuons quelques rappels sur les langages formels, les automates et les machines de Turing. Pour ces dernières, nous utiliserons ce semestre une définition un peu différente (mais équivalente).

2.1 Langages formels

On considère des mots formés sur un alphabet Σ, par exemple l'alphabet binaire $\Sigma = \{0, 1\}$ ou l'alphabet $\Sigma = \{a, b, c\}$. L'ensemble de tous les mots possibles sur un alphabet Σ est noté Σ^*. Un *langage* $L \subseteq \Sigma^*$ est un ensemble de mots. Il peut être fini ou infini.

Lorsqu'on considère un *problème de décision* (problème dont la réponse est OUI ou NON), on peut modéliser ce problème par un langage L qui contient toutes les *instances* (entrées du problème) dont la réponse est OUI. Résoudre un problème est alors équivalent à décider si un mot donné appartient ou non au langage correspondant.

2.2 Automates finis et langages réguliers

Un modèle de machine très simple est celui des automates finis. Ces automates lisent le mot d'entrée, un symbole à la fois (sans retour en arrière). Ils possèdent un nombre fini d'états Q (dont un état initial et un ou plusieurs états finaux) et un fonction de transition $\delta : Q \times \Sigma \to Q$ qui indique comment changer d'état en fonction du symbole lu. Lorsque la lecture d'un mot se termine, si son état courant est un état final, l'automate répond OUI (il *accepte* le mot w), sinon il répond NON (il *rejette* le mot).

Les automates finis sont capables de reconnaître les langages *réguliers*. [2] Les langages réguliers sont aussi ceux que l'on peut décrire par des *expressions régulières*.

Pour un état courant et un symbole lu, la fonction de transition δ ne donne qu'un seul nouvel état. Le fonctionnement de cet automate est donc *déterministe*. Dans la version

1. `https://arnaudcasteigts.net/teaching/calculabilite-complexite/`
2. La distinction entre "reconnaître" et "décider" n'apparaît qu'avec les machines de Turing. Pour les automates, c'est la même chose car ils terminent toujours.

non-déterministe des automates finis, on peut avoir plusieurs choix de transitions. On a alors $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, à savoir que δ renvoit un sous-ensemble d'états et non un seul. Moralement, vous pouvez imaginer que $\mathcal{P}$ signifie *plusieurs* (en vrai, *partie*).

L'exécution se duplique alors pour chacun des choix possibles, chaque choix donnant une *branche* d'exécution. L'automate accepte si et seulement si au moins une branche accepte. Ces automates ont la même puissance que leur version déterministe (langages réguliers).

Un exemple de langage non-régulier est : $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$, autrement dit l'ensemble des mots dont la première moitié est faite de `a` et la deuxième moitié d'autant de `b`.

2.3 Automates à pile et langages hors-contextes

Les automates à piles fonctionnent comme les automates finis, mais ils peuvent utiliser en plus une mémoire de capacité infinie utilisable sous forme d'une *pile* : les symboles qu'on y ajoute sont déposés sur les symboles déjà présents et on ne peut accéder qu'au dernier symbole ajouté. Les transitions de l'automate peuvent alors dépendre, en plus du symbole lu sur la bande d'entrée, du symbole qui se trouve au sommet de la pile, et elles peuvent y ajouter des élements. Ici, la version non-déterministe est strictement plus puissante que la version déterministe. Par défaut, un automate à pile désigne la version non-déterministe, plus commune car elle correspond exactement aux *langages hors-contextes* (décrits par les grammaires hors-contextes). À noter que lorsqu'on effectue un branchement non déterministe, c'est toute la machine qui est "dupliquée" : l'état courant, la position de la tête de lecture, et le contenu de la pile. Comme précédemment, un automate à pile non-déterministe accepte un mot si et seulement si au moins une branche accepte.

Un exemple de langage qui n'est pas hors-contexte est : $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$, autrement dit l'ensemble des mots dont le premier tiers des lettres sont des `a`, le deuxième tiers sont autant de `b` et le troisième tiers sont autant de `c`.

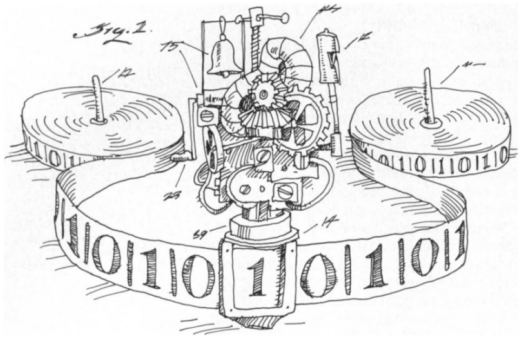
2.3.1 Hiérarchie de Chomsky

La hiérarchie de Chomsky regroupe les équivalences que nous avons vu entre différentes familles de langages (et grammaires correspondantes) et différents modèles de calculs. Les correspondances sont exactes. Les voici :

Langages reconnus	**Grammaire**	**Machine**
Régulier	Régulière	Automate fini
Hors-contexte	Hors-contexte	Automate à pile (non-déterministe)
Contextuel	Contextuelle	MT non-déterministe linéairement bornée
(Turing-)Reconnaissable	Générale	Machine de Turing

2.4 Machines de Turing

Une machine de Turing a aussi un fonctionnement proche de celui d'un automate. Ici, cependant, la mémoire se présente comme une ou plusieurs bandes qui peuvent être lues ou écrites à n'importe quel endroit.



La définition d'une machine de Turing est très robuste, dans le sens où quasiment toutes les définitions possibles correspondent à la même puissance de calcul. Notamment, utiliser une ou plusieurs bandes et utiliser une version déterministe ou non-déterministe n'affecte pas le type de langages que ces machines peuvent reconnaître ou décider. Nous reviendrons sur ces aspects là. En attendant, définissons ces machines plus en détail.

2.4.1 Définition

La version que nous utiliserons ce semestre utilise plusieurs bandes : une bande d'entrée (qu'on utilisera en lecture), une ou plusieurs bandes de travail (qu'on utilisera en lecture et écriture) et une bande de sortie (qu'on utilisera en écriture). Chaque bande a sa propre tête de lecture. Lors d'une transition, nous avons la possibilité de déplacer chacune des têtes de lecture indépendemment vers la gauche (L), la droite (R) ou la laisser sur place (S). 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}.

Définition 2.1 (Machine de Turing)**.** 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.

Note : Γ^k signifie $\Gamma \times \Gamma \times \cdots \times \Gamma$ (k fois, autrement dit un symbole pour chaque bande). De même, $\{L, R, S\}^k$ représente k mouvements (un pour chaque bande).

Fonctionnement. Par convention, la première bande (T_1, pour *tape* 1) est toujours la bande d'entrée et la dernière (T_k) est toujours la bande de sortie. Les $k-2$ autres bandes sont les bandes de travail, sur lesquelles la machine peut stocker ses résultats intermédiaires. La première cellule de chaque bande sera 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 cellules de toutes les bandes contiennent le symbole spécial $\square$, qui représente une cellule vide.

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 que son résultat est écrit sur la bande de sortie, elle se déplace sur l'état q_{halt}.

2.4.2 Exemple (Palindromes)

Pour rappel, on note w^R le mot w écrit à l'envers. Si $w = w^R$, on dit que ce mot est un *palindrome*. Étant donné un alphabet Σ, on peut définir le langage des palindromes sur Σ comme :

$L = \{w \in \Sigma^* \mid w = w^R\}$.

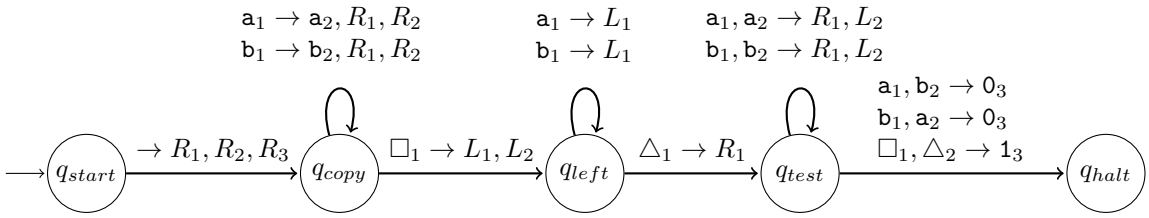
Nous avons déjà vu un exemple de machine de Turing reconnaissant ce langage. Cependant, elle n'utilisait qu'une bande. Nous allons voir comment l'utilisation de plusieurs bandes simplifie les choses.

Notre machine aura trois bandes (une d'entrée, une de travail, une de sortie). La stratégie est très simple :

1. Recopier le mot d'entrée 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 $\triangleright$, écrire 1 sur T_3 et terminer. Sinon, s'ils sont différents, écrire `0` sur T_3 et terminer. Sinon, se déplacer à droite sur T_1 et à gauche sur T_2, et recommencer l'étape 3.

Dans beaucoups de cas, nous nous contenterons d'une description à haut niveau telle que celle donnée ci-dessus. Mais nous souhaiterons occasionnellement plus de détails. 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_{accept}, q_{halt} et trois autres états qui nous serviront respectivement 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_{gauche} et q_{test}. Pour simplifier les notations graphiques, 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.



Représenter le contenu des bandes. Lorsqu'on représente le contenu des bandes, on peut utiliser un accent circonflexe sur un symbole pour indiquer que la tête de lecture y est positionnée.

Voyons quelques étapes de l'exécution de cette machine sur le mot d'entrée `abba`. L'état initial des bandes est comme suit :

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	$\triangleright\,\hat{\mathtt{1}}\,\square\cdots$

Voici la fonction δ sous forme d'un tableau :

État de départ	T_1	T_2	T_3	État d'arrivée	T_1	T_2	T_3	M_1	M_2	M_3
q_{start}				q_{copy}				R	R	R
q_{copy}	a			q_{copy}		a		R	R	
q_{copy}	b			q_{copy}		b		R	R	
q_{copy}	□			q_{left}				L	L	
q_{left}	a			q_{left}				L		
q_{left}	b			q_{left}				L		
q_{left}	▷			q_{test}				R		
q_{test}	a	a		q_{test}				R	L	
q_{test}	b	b		q_{test}				R	L	
q_{test}	a	b		q_{halt}			0			
q_{test}	b	a		q_{halt}			0			
q_{test}	□	▷		q_{halt}			1			

Pour être tout à fait complet, il faudrait beaucoup plus de transitions, afin qu'aucune case de ce tableau ne soit vide, en répétant les règles pour chaque symbole non-spécifié qui est compatible avec ces règles, afin que la fonction δ soit entièrement spécifiée. (Nous ne le ferons jamais, il faut juste comprendre que c'est faisable.)

2.4.3 Nombre de bandes

On peut toujours se limiter à une machine ayant une seule bande, mais c'est souvent plus fastidieux. À titre de comparaison, la machine que nous avions conçu pour reconnaître les palindromes avec une seule bande avait non pas 5, mais 8 états.

2.4.4 Non-déterminisme

Possibilité d'avoir plusieurs choix à un moment donné. Formellement :

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

On peut donc avoir plusieurs transitions dont les conditions sont les mêmes et les actions sont différentes, et ce, à partir du même état. Si c'est le cas, la machine effectue tous ces choix en même temps et l'exécution se dédouble autant de fois qu'il y a de choix (dans des univers "parallèles"). On parle de *branchement non-déterministe*. L'ensemble des branchements forme un *arbre* d'exécution.

Il faut voir chaque branchement comme une évolution indépendante du calcul, où toutes les informations de la machine (l'état courant, le contenu des bandes, les positions des têtes de lecture) évoluent *indépendemment* des autres branches. Pour que la machine termine, il faut que chaque branche termine. La machine accepte le mot d'entrée si et seulement si au moins l'une des branches accepte (écrit 1 sur la bande de sortie avant de terminer).

Lorsqu'on décrit une machine de Turing non-déterministe, on utilise souvent le terme de *deviner* (guess, en anglais). En effet, si l'on se concentre sur les exécutions qui ont aboutit à accepter un mot, on peut se dire que ces exécutions ont *deviné* les choix qu'il fallait faire.

Exemple : On veut jouer au Sudoku avec une machine non-déterministe, à partir d'une grille de jeu partiellement remplie. Le but est de décider si la grille admet une solution. Pour chaque case vide, la machine effectue 9 branchements non-déterministes (en écrivant une valeur différentes dans cette case pour chaque branchement). Si à un moment donné toutes les cases sont remplies, notre machine vérifie alors si les numéros sont valides (on pourrait aussi le faire progressivement, mais c'est plus simple comme ça). S'ils sont valides, la machine accepte, sinon, elle rejette. Ainsi, la grille de départ (le mot d'entrée) sera acceptée si et seulement si elle peut être complétée de manière valide. Le branchement qui l'acceptera aura *deviné* les numéros qu'il fallait jouer.

2.4.5 Machine de Turing universelle

Cela fera l'objet d'un cours prochain. Pour l'instant, contentons-nous de dire que l'on peut décrire une machine de Turing M (déterministe ou non) de manière textuelle, que l'on notera $\langle M \rangle$. Il est également possible de concevoir une machine de Turing déterministe M_U qui prend en entrée la description d'une machine $\langle M \rangle$ et un mot w, et qui produit la même sortie que M aurait produit avec l'entrée w.

Autrement dit, $M_U(\langle M \rangle, w) = M(w)$. Ainsi, M_U est une machine de Turing *universelle*, on peut la voir comme une équivalent de nos ordinateurs, qui d'une certaine manière exécutent des programmes qu'on leur passe en entrée.

Au passage, observons qu'une machine déterministe est capable de *simuler* une machine non-déterministe. Les deux ont donc les mêmes capacités de calcul (pour la complexité, la situation est différente, on pense que les machines non-déterministes sont plus rapides, même si l'on n'arrive pas à le démontrer).

2.4.6 Reconnaissable *versus* Décidable ?

Une machine M **décide** un langage L si :

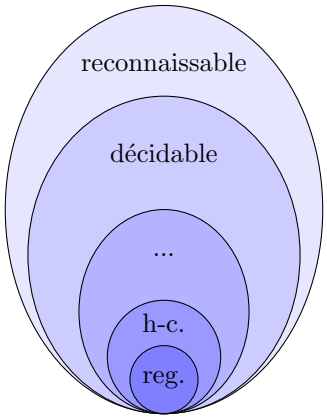
- $w \in L \implies M(L)$ termine et accepte w.
- $w \notin L \implies M(L)$ termine et rejette w.

Une machine M **reconnaît** un langage L si :

- $w \in L \implies M(L)$ termine et accepte w.
- $w \notin L \implies M(L)$ termine et rejette w, ou ne termine pas.

Décider est donc une notion plus forte, qui implique aussi reconnaître.

Un langage est **décidable** (synonyme : “récursif”) s'il existe une machine M qui décide ce langage. Il est **reconnaissable** (synonymes : “récursivement énumérable” ou “semi-décidable”) s'il existe une machine M qui reconnaît ce langage.



Certains langages sont reconnaissables, mais pas décidables, comme par exemple le problème de l'arrêt, où l'on cherche à déterminer si une machine donnée par sa description $\langle M \rangle$ termine sur une entrée donnée w. Ce langage est cependant reconnaissable en utilisant simplement une machine universelle $M_U(\langle M \rangle, w)$ qui se contente de simuler M et qui écrira 1 sur la sortie lorsque $M(w)$ termine (si elle termine).

Il existe des langages qui ne sont même pas reconnaissables. Nous en verrons un exemple concret prochainement.