Calculabilité et Complexité (11X008) - Printemps 2026

2. Simulation et opérations sur les langages

Enseignant: Arnaud Casteigts

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

Dans ce deuxième cours, nous expliquons comment décrire une machine de Turing sous forme d'une chaîne de caractères (un mot). Nous expliquons ensuite le fonctionnement d'une machine universelle, capable de simuler une machine dont une telle description lui serait passée en entrée. Cette capacité de simulation entre machines de Turing est très utile. Nous l'utilisons par exemple pour étudier comment les langages décidables et reconnaissables se comportent sous les opération d'union, d'intersection et de complément.

2.1 Description d'une machine et simulation

Dans cette partie, nous allons d'abord voir comment une machine de Turing (à une bande) peut être complètement décrite par une simple chaîne de caractère. Puis nous verrons comment une autre machine (à 4 bande) peut simuler la première à partir de sa description.

2.1.1 Description d'une machine

Pour simuler une machine M, il faut déjà se mettre d'accord sur la description textuelle $\langle M \rangle$. 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 $\texttt{0}^i$. 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 :

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

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 mot `11`, en y ajoutant un délimiteur `111` au tout début et à toute la fin de la description globale, par exemple :

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

2.1.2 Simulation d'une machine

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 $111\ t_1\ 11\ t_2\ 11\ \cdots\ 11\ t_\ell\ 111$, 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)	$\mathtt{0}$
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.

2.1.3 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 simuler les machines de Turing à une bande 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*).

2.1.4 Conclusion

Maintenant qu'on sait qu'une machine de Turing peut en simuler une autre, on va pouvoir utiliser cela fréquemment. Fort heureusement, on aura plus besoin d'expliquer en détail comment on le fait, on se contentera de le dire ou de le dessiner (comme dans la section suivante).

2.2 Opérations sur les langages

Comment les langages décidables ou reconnaissables se comportent-ils sous les opérations d'union, d'intersection et de complément ?

Pour rappel, $w \in L_1 \cup L_2$ signifie que w est dans L_1 ou dans L_2 (ou les deux) ; $w \in L_1 \cap L_2$ signifie que w est dans L_1 et dans L_2 ; et $w \in \overline{L}$ signifie que w n'est pas dans L. Nous avons vu au premier semestre que ces opérations se comportent plus ou moins bien selon les familles de langages considérées. Notamment,

Langages réguliers :

- Si L_1 est régulier et L_2 est régulier, alors $L_1 \cup L_2$ est régulier
- Si L_1 est régulier et L_2 est régulier, alors $L_1 \cap L_2$ est régulier
- Si L est régulier, alors $\overline{L}$ est régulier

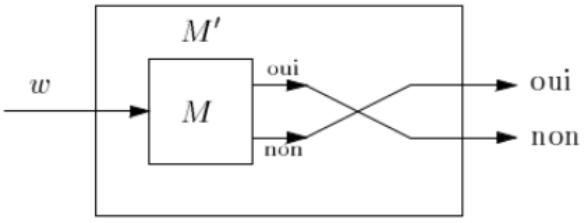
Langages hors-contextes :

- Si L_1 et L_2 sont hors-contextes, alors $L_1 \cup L_2$ est hors-contexte.
- Si L_1 et L_2 sont hors-contextes, alors $L_1 \cap L_2$ n'est pas forcément hors-contexte.
- Si L est hors-contexte, alors $\overline{L}$ peut ne pas être hors-contexte.

2.2.1 Langages décidables

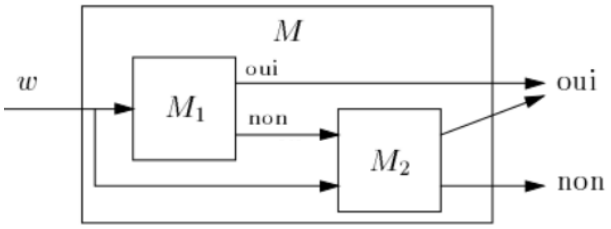
- Si L est décidable, alors $\overline{L}$ est décidable.

  Preuve : Soit L un langage décidable. Il existe donc une machine de Turing M qui s'arrête toujours et décide L. Pour décider $\overline{L}$, il suffit de créer une machine M' qui simule M sur l'entrée souhaitée, puis d'en inverser la sortie : $M'(w)$ accepte si $M(w)$ rejette et $M'(w)$ rejette si $M(w)$ accepte. Cette construction peut être représentée comme suit :



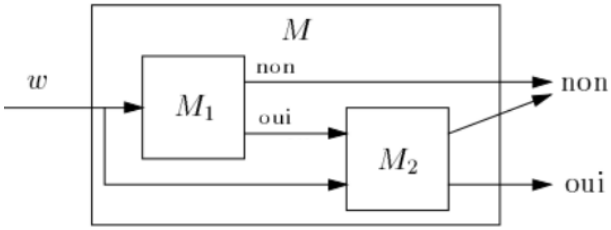
- Si L_1 et L_2 sont décidables, alors $L_1 \cup L_2$ sont décidables.

  Preuve : Soit L_1 et L_2 deux langages décidables. Il existe donc deux machines M_1 et M_2 qui s'arrêtent toujours et telles que M_1 décide L_1 et M_2 décide L_2. On peut créer une machine M qui simule d'abord M_1 sur l'entrée souhaitée. Si $M_1(w)$ accepte, on peut déjà s'arrêter et accepter w (c'est suffisant). Si $M_1(w)$ refuse, on simule $M_2(w)$ et on renvoie la même réponse que $M_2(w)$.



- Si L_1 et L_2 sont décidables, alors $L_1 \cap L_2$ sont décidables.

  Preuve : Similaire à l'union, mais en acceptant seulement si les deux acceptent.



2.2.2 Langages reconnaissables

- Si L est reconnaissable, alors $\overline{L}$ peut ne pas être reconnaissable.

  Preuve : L est reconnaissable, donc il existe une machine M qui reconnaît L. Si $\overline{L}$ est reconnaissable, alors il existe aussi une machine M' qui reconnaît $\overline{L}$. On peut alors créer une machine M_D qui utilise M et M' pour décider L, ce qui est une contradiction pour certains langages reconnaissable qui ne sont pas décidables.

  Remarque : En fait, les langages décidables correspondent exactements aux cas particuliers des langages reconnaissables dont le complément est aussi reconnaissable.

- Si L_1 et L_2 sont reconnaissables, alors $L_1 \cup L_2$ est reconnaissable.

  Preuve (plus subtile) : Soit L_1 et L_2 deux langages reconnaissables. Il existe donc deux machines M_1 et M_2 tels que M_1 reconnaît L_1 et M_2 reconnaît L_2. On souhaiterait créer une machine M qui simule ces deux machines et qui accepte l'entrée w si au moins l'une des deux accepte. La difficulté est que si $w \notin L_1$, on ne peut pas attendre que M_1 termine pour tester si $w \in L_2$, car M_1 ne termine pas forcément dans ce cas. L'astuce consiste à simuler M_1 et M_2 en même temps. Par exemple, M peut simuler quelques pas de calcul sur M_1, puis quelques pas de calcul sur M_2, puis à nouveau M_1, et ainsi de suite en alternant les deux (oui, c'est possible !). Si $w \in L_1 \cup L_2$, l'une des deux machine finira par terminer en acceptant w, et M pourra à son tour terminer en acceptant w. Si $w \notin L_1 \cup L_2$, alors M ne terminera pas forcément, mais cela est OK puisqu'on souhaite seulement *reconnaître* $L_1 \cup L_2$.

- Si L_1 et L_2 sont reconnaissables, alors $L_1 \cap L_2$ est reconnaissable.

  Preuve : C'est plus facile que l'union. Il suffit de créer une machine M qui simule M_1 puis M_2 et qui n'accepte que si les deux acceptent. Si M_1 ou M_2 ne s'arrête pas, ce n'est pas grave, car cela veut dire que $w \notin L_1 \cap L_2$ et qu'il est donc valide de ne pas répondre (puisqu'on veut seulement *reconnaître* $L_1 \cap L_2$).