Calculabilité et Complexité (11X008) - Printemps 2024

3. Décidable, reconnaissable, simulation

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

3.1 Autres modèles de calcul

Comme déjà évoqué, la conjecture de Church-Turing stipule que tout ce qui est calculable physiquement peut être calculé par une machine de Turing. Le consensus scientifique aujourd'hui est que cela semble vrai. Du moins, aucun contre-exemple n'a jamais été trouvé. De nombreux autres modèles de calcul ont été inventés, qui se sont tous avérés équivalents aux machines de Turing (on dit qu'ils sont *Turing-complets*).

Parmi les plus connus, on peut citer :

- Le λ-calcul (lambda calcul), qui est un système formel de calcul basé, entre autres, sur des règles de substitution et qui sert de fondation à la programmation récursive.
- Nos ordinateurs actuels, ou plutôt leur version mathématique, les *machines RAM* (random-access machine), qui permettent d'accéder à des cellules mémoires directement (c.à.d. 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 du problème. On sait depuis 2002 que certains de ces modèles sont Turing-complets.
- Les origamis : on peut encoder n'importe quel traitement sous forme de pliage de papier. Les origamis sont également Turing-complets.

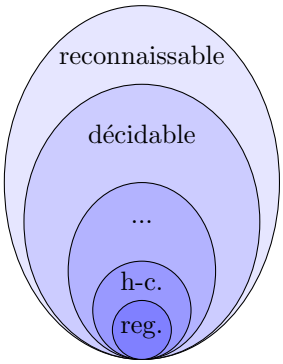
Où que l'on regarde dans la nature, les modèles de calculs que l'on trouve semblent être ni plus ni moins puissants que les machines de Turing. Cela inclut aussi les ordinateurs quantiques. Ces derniers pourraient être différents en termes de *complexité*, mais leur *expressivité* (ce qu'ils permettent de calculer) n'est pas supérieure à celle d'une machine de Turing.

3.2 Décidable versus reconnaissable

Reprenons le point de vue des langages formels (ou des problèmes de décisions), pour lesquels on s'intéresse aux ensembles de mots (d'instances) qu'une machine accepte. Le fait

que les machines de Turing ne terminent pas toujours impose de distinguer deux familles distinctes : les langages décidables et les langages reconnaissables.

3.2.1 Définition



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.

Un langage est **décidable** s'il existe une machine M qui décide ce langage. Il est **reconnaissable** s'il existe une machine M qui reconnaît ce langage. Décider est donc une notion plus forte, qui implique aussi reconnaître.

Sur wikipédia, vous trouverez souvent les synonymes suivants : décidable="récursif", et reconnaissable="récursivement énumérable"="semi-décidable".

3.3 Propriétés de ces langages

Voyons quelques propriétés des langages décidables et des langages reconnaissables, notamment la stabilité de ces familles sous les opérations ensemblistes usuelles que sont l'union, l'intersection et le complément. Pour rappel, $w \in L_1 \cup L_2$ signifie que w est dans L_1 ou dans L_2 (ou dans les deux) ; $w \in L_1 \cap L_2$ signifie que w est dans L_1 et dans L_2 ; et $w \in \bar{L}$ signifie que w n'est pas dans L. Commençons par quelques rappels :

Pour les 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 le langage complément $\bar{L} = \{w \in \Sigma^* | w \notin L\}$ est régulier

Pour les 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 $\bar{L}$ n'est pas forcément hors-contexte.

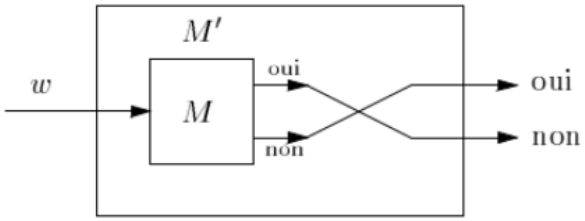
3.3.1 Langages décidables

Qu'en est-il des langages décidable ou reconnaissables par des machines de Turing ?

Complément

Si L est un langage décidable, alors $\bar{L}$ est un langage décidable.

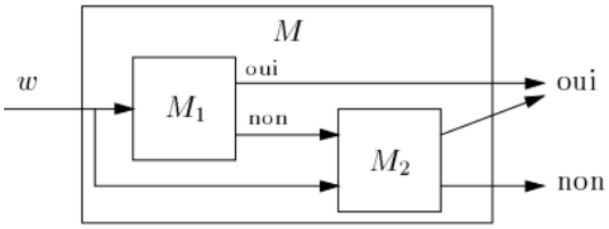
Preuve : Soit L un langage décidable. Il existe donc une machine de Turing M_L qui s'arrête toujours et décide L. Pour décider $\bar{L}$, il suffit de créer une machine M' qui simule M sur l'entrée w reçue, puis en inverse la sortie : $M'(w)$ accepte si $M(w)$ rejette et $M'(w)$ rejette si $M(w)$ accepte.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



Union

L'union de deux langages décidables est un langage décidable.

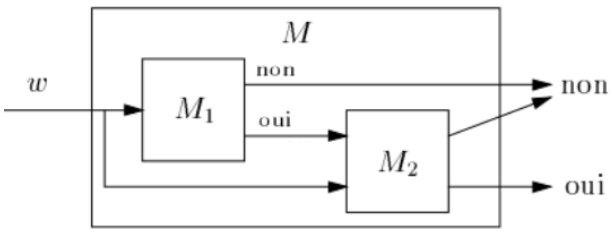
Preuve : Soit L_1 et L_2 deux langages décidables. Il existe donc deux machines M_1 et M_2 qui s'arrêtent toujours et telles que M_1 décide L_1 et M_2 décide L_2. On peut créer une machine M qui simule d'abord M_1 sur l'entrée w reçue. Si $M_1(w)$ accepte, on peut déjà s'arrêter et accepter w (c'est suffisant). Sinon, on simule $M_2(w)$. Si $M_2(w)$ accepte, on accepte. Sinon, cela signifie que les deux machines ont rejeté. On rejette donc w.



Intersection

L'intersection de deux langages décidables est un langage décidable.

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



3.3.2 Langages reconnaissables

Complément

Si L est reconnaissable, alors $\bar{L}$ n'est pas forcément reconnaissable.

En effet, si L est reconnaissable, il n'existe pas forcément de machine qui termine et accepte lorsque $w \in \bar{L}$. Si une telle machine existe (si $\bar{L}$ est reconnaissable), alors cela fait de L un langage dédicable. Réciproquement, si L est décidable, alors L et $\bar{L}$ sont tous deux reconnaissables.

Union

L'union de deux langages reconnaissables est un langage 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* $L_1 \cup L_2$.

Vous verrez ce type de techniques en exercices.

Intersection

L'intersection de deux langages reconnaissables est un langage reconnaissable.

Preuve : ~~L'idée est la même que pour l'union, sauf que M n'acceptera que si les deux machines M_1 et M_2 acceptent.~~ Suite à un commentaire pertinent, il n'est pas nécessaire pour l'intersection d'effectuer les simulations de manière alternée, on peut simuler M_1 puis M_2 et n'accepter que si les deux acceptent.

3.4 Description d'une machine et simulation

Jusqu'à présent, nous avons supposé qu'une machine de Turing peut en simuler une autre, mais nous n'avons pas vu comment. Entrons dans le vif du sujet. Il existe plusieurs résultats dans cette lignée, 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.

Nous ne détaillerons pas ces trois résultats, mais gardons-les en tête, car ils impliquent qu'il suffit de savoir 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. Cela tombe bien, c'est ce que nous allons faire !

3.4.1 Encodage d'une machine

Pour simuler une machine M, il faut déjà être capable de la décrire de manière textuelle $\langle M \rangle$, pour qu'une autre machine puisse prendre cette description en entrée.

Pour simplifier, supposons que la machine M que l'on veut simuler n'utilise qu'une seule bande, l'alphabet $\Gamma = \{0, 1, \triangleright, \square\}$ et les mouvements $\{L, R, S\}$. La machine M a un certain nombre d'états $Q = \{q_1, q_2, q_3, \dots\}$. Sa fonction de transition est entièrement spécifiée sous forme d'un tableau dont chaque ligne représente une transition, par exemple :

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

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

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

$$\mathtt{0010101000010}$$

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 découvrir quels sont les états de M en cherchant dans ce mot, 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 début et à la fin, par exemple :

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

3.5 Simulation par une machine universelle

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

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

Nous ne décrirons pas la machine M_u en détail, mais nous pouvons mentionner les principaux éléments de son fonctionnement pour se convaincre qu'une simulation est en effet possible. L'entrée de cette machine est un mot de la forme $\langle M \rangle w$, où w est l'entrée que l'on souhaite fournir à M lors de la simulation. Comme vu précédemment, $\langle M \rangle$ se présente sous la forme $111\ t_1\ 11\ t_2\ 11\ \cdots\ 11\ t_k\ 111$, où t_i correspond à l'encodage de la $i^{\text{ème}}$ transition de M (qui en a k au total).

Pour simuler cette machine, nous allons utiliser une machine de Turing M_u à 4 bandes : 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 ce qui suit sur T_2 jusqu'au premier symbole $\square$. La bande T_3 contiendra toujours l'encodage de 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_u)	

La machine M_u entre alors dans une boucle dont le but est à chaque fois de simuler un pas de calcul de la machine M comme suit :

1. Chercher sur T_1 une transition $u1v1x1y1z$ dont le premier facteur u correspond au contenu de T_3 et le second facteur 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 différent de 00 (état final), on recommence. Sinon, on recopie T_2 sur la sortie et on termine.

On voit bien qu'après chaque étape de la simulation (chaque transition de M), le contenu de la bande T_2 correspond au contenu qu'aurait M sur son unique bande. 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 mériteraient d'être plus détaillées, mais cette description est suffisante 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 (ou alternativement, mais c'est plus complexe, gérer les erreurs pendant la simulation).