Structures Discrètes

Jean Galte

January 2024

1 Calculabilité

Cette partie est donnée par Guillaume Feuillade. Ce cours décrira les données par des mots, là où le livre "Calculabilité" de Benoît Monin se concentrait sur l'étude des porblèmes qui impliquent surtout des entiers. En réalité, ça ne change pas grand chose, les problèmes qu'on arrive à modéliser en informatique peuvent se décrire dans des entiers.

1.1 Définitions informelles

Une machine peut être vue comme une boîte noire qui prend une entrée, et donne une sortie (ou "pas de sortie", par exemple return void, retourner 0, peuvent être interprétés comme "pas de retour", même si en réalité on retourne bien quelque chose), ou ne termine pas. Cette définition est vague, mais ce n'est pas grave, puisqu'on va travailler avec des définitions de machine beaucoup plus formelles. Elle sert à faire correspondre la définition "usuelle" d'une machine et d'une machine (de Turing).

Un modèle de calcul est une description de toutes les opérations élémentaires que l'on peut effectuer, qu'on peut appeler instructions.

Un programme est un enchainement d'opérations données par un modèle de calcul. Cette définition repose donc sur celle de modèle de calcul : aussi bien qu'on ne peut pas définir un groupe sans avoir sa loi, on ne peut pas définir un programme sans dire dans quel modèle il s'exprime. On remarque que cela coïncide avec la notion de machine : une machine peut être décrite par un couple (modèle, programme).

Un problème de décision est donné par :

- Une donnée : la représentation finie d'un éléments d'un ensemble, éventuellement infini.
- Une question : un énoncé portant sur les éléments de cet ensemble. La réponse à cet énoncé doit être "Vrai" ou "Faux", c'est une question fermée.

On dit qu'un problème est décidable si une machine (de Turing) peut répondre à ce problème, c'est à dire répondre à la question avec la donnée.
Voilà un exemple de problème de décision : la donnée est un entier en base 10, la question est "cet entier est-il premier ?". La donnée décrit, elle représente, l'ensemble $\mathbb{N}$. La question, elle, peut être vue comme un prédicat. Typiquement, ce problème est décidable (de la classe P, car on a un algorithme en temps polynomial qui décide ce problème, plus précisément en $o(\sqrt{(}n))$.
Voilà un autre exemple de problème de décision : la donnée est un graphe, et la question est "existe t-il un chemin Hamiltonien dans le graphe ?". Ce problème est décidable (en particulier, on peut énumérer tous les chemins). En revanche, il est de la classe NP (on n'a pas d'algorithme en temps polynomial pour répondre à ce problème).

Voilà un autre exemple de problème de décision : la donnée est une suite finie de couple de mots $(u_i, v_i)_{i\in[\![1,n]\!]}$. La question est "existe t-il une suite d'indices $(j_i)_{i\in[\![1,k]\!]}$ telle que $\cdot_{i=1}^{i=k} u_{j_i} = \cdot_{i=1}^{i=k} v_{j_i}$?" (où $\cdot$ est la concaténation). C'est un problème indécidable, i.e il n'existe pas de machine capable de répondre à la question avec la donnée. Il est assez connu, il s'appelle "Problème de correspondance de Post" (PCP).
Chaque donnée constitue une instance du problème (ici c'est un mot, mais ça pourrait ne pas être ça. Par exemple, le problème de trouver un chemin hamiltonien : on peut décrire le graphe avec un mot au format 'sommet1...——arrête1...' , mais on aurait pu décrire ça avec un entier naturel, puisqu'on sait décrire un mot avec un entier naturel). Les données correspondent donc à des mots, et en tant qu'ensemble, à un langage. Le sous langage des mots pour lesquels la réponse est "OUI" est le **langage caractéristique** du problème P, noté L_P.
Si P est un problème, $\bar{P}$ est le problème contraire. Son langage caractéristique est $L_{\bar{P}} = \bar{L_P}$.
On dit qu'une machine décide d'un problème si en un temps fini elle permet de déterminer pour tout mot s'il appartient à L_P où $\bar{L_P}$.

1.2 Machine de Turing

Une machine de Turing est une tentative de formaliser la notion de machine. Ce n'est pas la seule, en fait les termes du λ-calcul, les fonctions récursives (en tant que fonction, pas en tant que modèle, i.e pas les fonctions récursives générales), les programmes python, sont aussi des formalisations de la notion de modèle machine.
Une machine de Turing peut être vue comme un ruban infini découpé en cases, avec des lettres d'un alphabet $\sum$ écrites sur les cases, un pointeur (tête de lecture/écriture) qui pointe sur une case qui peut être déplacée à gauche ou à droite, et une mémoire auxiliaire finie : les états de la machine. Voilà une définition un peu plus formelle :

Une machine de Turing (qu'on abrège MT, ou TM si on est une merde anthropologique qui mange 4 fois par semaine à Burger King louée à la propagande américaine) est :

- Un alphabet $\sum$ ayant au moins deux symboles, l'un d'entre eux étant le caractère blanc, noté #
- Un ensemble fini Q d'états
- Un programme P qui est un sous ensemble de $Q \times \sum \times \sum \times M \times Q$ où $M = \{\leftarrow, \rightarrow\}$ qui désigne un mouvement de la tête de lecture. Les éléments de P sont donc des instructions (q, x, y, m, q') où q et q' décrivent l'état de départ et de fin, x décrit la lettre lue, y la lettre écrite, et m le déplacement effectué. On les note $(q, x) \rightarrow (y, m, q')$.

Une MT est donc définie par ce triplet $< \sum, Q, P >$. Notons que cette définition n'est pas déterministe : on peut avoir deux instructions qui partent du même état, et elle n'est pas non plus "complétiste" : on peut arriver dans un état sur une certaine case où le programme n'a plus d'instruction qui réponde à notre position. On appelle configuration d'une MT $< \sum, Q, P >$ un triplet (f, q, xy) de $\sum^* \times Q \times \sum \sum^*$ tel que :

- fxy constitue la partie utile de la bande
- q est l'état courant de la MT.
- la tête sur positionne sur la case contenant x.

Concrètement, si la machine se trouve dans un état q, f peut désigner le mot formé par les lettres à gauche de la case où se situe la tête de lecture, x désigne l'état de la machine, et y désigne le mot formé par les lettres à droite de la case. En fait, une configuration permet de décrire la machine dans son état "physique" : penser au ruban et à l'état.
Si la machine de Turing est dans la configuration (f, q, xy), on dit qu'il existe une transition $(f, q, xy) \rightarrow (f', q', x'y')$ si $(q, x) \rightarrow (y, m, q')$ est dans P et si $m = \rightarrow$, $f' = fy$ et $y = x'y'$ ou si $m = \leftarrow$, $f = f'x'$ et $y' = ...$
Si $< \sum, Q, P >$ est une MT et c une configuration, on dit qu'il existe un calcule valide menant à la configuarion c' si il existe une séquence $c_0...c_n$ avec $c = c_0$, $c' = c_n$ et $\forall i \in [\![1, n]\!], c_{i-1} \rightarrow c_i$ est une transition valide.

Si T est une MT, $L_T = \{\omega / \exists$un calcul valide sur T qui s'arrête$\}$. L_T est appelé "le langage reconnu par T". On adopte deux conventions pour ce cours :

- Au départ, la partie utile n'est jamais vide.
- Au départ, la partie utile ne contient jamais deux cases blances (#) consécutives.
- Au départ, la tête de lecture est positionnée au début de la partie utile.
- l'état de départ est toujours q_1 (ou q_0 s'il y est).

On dit qu'une machine de Turing est déterministe si à partir d'une configuration donnée, on a au plus une seule possibilité (exactement comme pour un automate : il est déterministe si pour passer d'un état à un autre, on a au plus un symbole) : T est déterministe si $\forall (q, x) \in Q \times \sum$ il existe au plus une règle (q, x, x', m, q') dans son programme. Dans ce cas là, le programme est une **fonction partielle**.

On se place dans le cas des MT déterministes. Soit $T = < \sum, Q, P >$, on peut définir $\phi_T : \sum^* \rightarrow \sum^*$. On peut définir le domaine de ϕ_T (au sens mathématique du terme, c'est à dire que la fonction n'est pas partielle : on peut alors la voir comme un ensemble de couples, et non un processus) est L_T. Dans ce cas, $(\epsilon, q_1, \omega) \rightarrow_{\max} (f, q, g) \Rightarrow \phi_T(\omega) = f \cdot g$.
Voici un exemple : Soit $T_1 = < \sum, Q, P >$, avec $\sum = \{0, 1\#\}$ et $Q = \{q_1, q_2, q_3\}$, et le programme est le suivant : $P =$

- $q_1, 1 \rightarrow (1, \rightarrow, q_1)$
- $q_2, 0 \rightarrow (1, \leftarrow, q_3)$
- $q_1, 0 \rightarrow (0, \rightarrow, q_1)$
- $q_2, 1 \rightarrow (0, \leftarrow, q_2)$
- $q_1, \# \rightarrow (\#, \leftarrow, q_2)$
- $q_2, \# \rightarrow (1, \leftarrow, q_3)$

On peut représenter ça par un graphe, (voir cahier). On peut dire qu'ici les états on des rôles, comme souvent lorsqu'on modélise des processus précis : l'état q_1 a pour rôle "d'aller tout à droite" sans rien changer puis de passer à q_2 en allant à gauche toujours sans écrire, l'état q_2 a pour rôle " ...

Une fonction partielle ϕ de $\sum^*$ dans $\sum^*$ est Turing-Calculable s'il existe une MT déterministe telle que $\phi_T = \phi$. Pour tout langage, $L \subset \sum^*$, la fonction caractéristique du langage est ϕ_L définie par : $\phi_L(\omega) = 0$ si $\omega \in \bar{L}$ et 1 si $\omega \in L$. Cette fonction n'est pas nécessairement réalisable par une machine de Turing ! cela correspond aux problèmes indécidables. On veut maintenant savoir si toute fonction $\sum^* \to \sum^*$ est Turing-calculable. On peut donner un argument diagonal pour répondre "non" : exhiber le problème de l'arrêt est en réalité une forme d'argument diagonal, mais on peut également raisonner simplement sur la cardinalité (même sans utiliser l'hypothèse du continu). On suppose que l'alphabet contient une lettre "non blanche" (sinon le problème n'a en fait que très peu d'intérêt).
Les MT sont dénombrables (en tant que triplets d'ensemble dénombrables). Les fonctions $\sum^* \to \sum^*$ ne le sont pas : soit T_i la i-ème MT, et ω_i le i-ème mot de $\sum^*$. Alors, $\phi_{T_i}(\omega_i)$ donne bien un résultat ω_i' (ou bien ne termine pas, mais dans ce cas on dit qu'elle est coincée dans cet état. On n'a pas besoin qu'elle s'arrête sur un mot, juste qu'elle se comporte d'une façon caractérisée pour pouvoir définir ϕ). On définit une fonction $\phi : \omega_i \to \epsilon(\epsilon \neq \omega_i')$ (ou alors : ϵ est un comportement différent de ω_i' si on le considère comme un comportement, ce qui sert parce que tout ne s'arrête pas). Donc pour toutes les machines T_i, T_i ne réalise pas ϕ car ϕ donne un résultat différent de T_i. La preuve est non constructive : ϕ est définie mathématiquement, mais on ne peut pas nécessairement arriver à ses images par un processus : par exmeple, une machine de Turing T_i pourrait ne pas terminer sur ω_i, et dans ce cas on ne peut pas construire ϕ.
On dit qu'une MT est normalisée si elle possède un sous ensemble A d'états qui ne figurent dans aucun membre gauche de règle et pour tout état q de la MT, $q \notin A$, alors pour toute lettre x, il exist eune règle qui commence par $q, x \to$: l'idée c'est que soit la machine s'arrête sur un état, soit elle va à droite ?
On dit qu'une MT est standardisée si lorsque son calcul s'arrête, sa tête de lecture est la tête de lecture est positionnée au début de la partie utile.
Les machines de Turing standardisées et normalisées peuvent se "composer" : celà donne lieu à une théorie entière qui repose sur cette composition, une théorie des catégories adaptée pour les MT.
Quelques machines standardisées normalisées élémentaires :

- Fonction constante (efface toute la bande, et écrit un mot donné)
- Les projections : envoie $\omega_1 \# ... \# \omega_n$ sur un ω_i
- La fonction successeur, prédecesseur, sur un nombre binaire
- Les structures conditionnelles : si, alors, sinon.
- Boucles "tant que"

1.3 Variantes des machines de Turing

On peut définir des variantes des machines de Turing. Pour exprimer les différences de puissance entre ces modèles de calcul (on considère pas les programmes ici, on s'intéresse vraiment au modèle de calcul : varier les programmes sur une MT nous apporte juste une autre MT), on parle de simulation.
On dit qu'une MT T est simulée par T' si on peut trouver une injection i des configurations de T dans les configurations de T' telle que si $c \to_T c'$, alors $i(c) \to_{T'}^* i(c')$. Ça correspond à la commutativité de ce diagramme :

$$\begin{array}{ccc} C & \xrightarrow{T} & C' \\ {\scriptstyle i}\downarrow & & \downarrow{\scriptstyle i} \\ I(C) & \xrightarrow[T'*]{} & I(C') \end{array}$$

Voici un exemple de simulation :
Si $T =< \sum, Q, P >$ est une MT qui réalise ϕ, on peut construire $T' =< \sum', Q', P' >$ avec $\sum' = \sum \cup \{a, b\}$ qui simule T de telle manière qu'une configuration (f, q, g) de T est codée par $(a \cdot f, q', g \cdot b)$.

1.3.1 Machines à k bandes

Une machine à k bandes est une machine sur laquelle on a k bandes infinies (à droite et à gauche), chacune munie d'une tête de lecture/écriture, un ensemble d'états. Son programme est consititué de règles de la forme $q, x_1, ..., x_k \to y_1, m_1, ..., y_k, m_k, q'$.
Pour la simuler par une machine de Turing usuelles on peut remplacer les symboles de l'alphabet par des k-uplets, c'est la façon la plus simple. Mais on peut également découper notre bande unique en k-bandes en séparant les parties utiles de chaque bande. L'intérêt est grand : bien des processsus peuvent être représentés par des machines à k bandes.

1.4 Simplification de MT

1.4.1 Déterminisation de MT

On simule une machine de Turing non déterministe par une machine à trois bandes.

- Sur la bande 1, on écrit le mot d'entrée ω
- La bande 2 simule la machine
- La bande 3 énumère les séquences de règles de T (dans l'ordre lexicographique disons, tant que ça énumère de façon raisonnable c'est bon).

Plus formellement : 1) on liste une nouvelle séquence sur b3, 2) on recopie la bande 1 sur la bande 2, puis 3) on exécute la séquence écrite sur la bande 3 sur la bande 2. Si l'exécution arrive sur une configuration finale, on termine (le résultat, ou la sortie, est disponible sur la bande 2). Sinon, on efface le contenu de la bande 2 et on retoourne à la première étape.

1.4.2 Réduction d'alphabet

Soit T une MT sur $\sum$. On va simuler cette machine par une machine T' qui ne contient que deux symboles en plus de la case vide (traditionnellement 0 et 1). Formellement, on explique pas dans ce cours comment faire mais c'est asssez simple : on code les lettres de $\sum$ en binaire, et on crée des états et règles pour écrire ces lettres en binaire. Par exemple, pour une règle $q, x \to y, m, q'$ on donne la séquence suivante : 1) on lit x (en binaire) 2) on écrit y (en binaire) 3) on passe en q' en effectuant le mouvement m. La séquence pour lire un nombre binaire demande de créer des états "avec mémoire", c'est à dire de stocker les caractères précédents dans l'état. De la même façon, pour écrire un nombre binaire on a besoin de stocker les "étapes d'écriture" : la séquence pour lire un symbole et le remplacer par un autre est composée du même nombre d'états qu'on a de chiffres dans la concaténation des deux mots en binaire.
Réduction à 2 symboles : $\{\#, 1\}$. Pour ça, on se ramène à 3 symboles. On va alors coder la case vide par 1#11, le 0 par 11#1 et le 1 par 111#1 : on encode chaque symbole par une suite de 4 symboles, la position de la case blanche indique le caractère. Alors, le début et la fin de la partie utile seront codés par ##.

2 Décidabilité, indécidabilité

C'est un peu le coeur de ce cours.

2.1 Problème de l'arrêt

Le problème de l'arrêt est bien connu (en fait, c'est une sorte d'argument diagonal sur les machines de Turing).

2.1.1 Machine de Turing universelle

On suppose qu'il est possible d'encoder une MT sur un mot (par exemple : on crée des caractères spéciaux pour séparer, et on ajoute à la chaine son alphabet, et ses instructions). Donc chaque MT est codée par un entier naturel i (car on peut coder un mot par un entier naturel). On utilisera la notation T_i pour désigner la MT numéro i.
Proposition 1 : il est possible de fabriquer une MT, nommée T_d qui prend en entier le nombre i et produit en sortie le programme de T_i.
Proposition 2 : il existe une machine T_u telle que l'exécution de T_u sur l'entrée $i\#\omega$ produit le résultat de T_i sur ω.

Construction de T_u : plusieurs possibilités existent, mais en voici une simple. Au début, T_u contient 3 bandes b_1, b_2, b_3. b_1 est dite "bande de travail" : elle contient ω , l'évaluation de la MT sur l'entrée. b_2 contient le programme de T_i. b_3 contient le mot q_1.
Le principe est alors assez clair : lire la lettre sur b_1 et l'état sur b_3, rechercher sur b_2 la règle correspondante, appliquer la règle sélectionnée sur b_1 et écrire le nouvel état sur b_3.

2.1.2 Problème de l'arrêt (d'une MT)

Donnée : une MT T_i et un mot ω
Question : Est-ce que l'exécution de T_i sur ω termine ? (enfin, en version déterministe. Sinon se pose plutôt la question : est-ce qu'il existe une exécution de T_i qui termine sur ω)

Ce problème est indécidable : aucune MT ne peut y répondre. La preuve ressemble à un argument diagonal. Supposons l'existence d'une telle machine T_a qui retourne 1 si le calcul termine sur ω, 0 sinon. Alors, on construit une machine T telle que $T(i)$ retourne 1 si $T_a(i, i)$ retourne 0 et ne termine pas si $T_a(i, i)$ termine. Il existe $k \in \mathbb{N}$ tel que $T = T_k$. Si $T_k(k)$ retourne 1, c'est que $T_a(k, k)$ retourne 0 : donc que $T_k(k)$ ne termine pas, c'est absurde : $T_k(k)$ ne peut retourner 1. Si $T_k(k)$ ne termine pas : alors $T_a(k, k)$ retourne 1 donc $T_k(k)$ termine, ce qui est également absurde.

Le problème de l'arrêt est donc indécidable. Cependant, il est semi décidable (équivalent de "calculatoirement énumérable" dans le bouquin de Monin), on peut énumérer les machines de Turing qui terminent sur une entrée donnée.

3 Problèmes de décision de langages

Définition : on dit qu'un langage est récursif s'il exsite une MT qui a un calcul qui s'arrête sur toute donnée et qui reconnaît ce langage.
Définition : un langage est récursivement énumérable s'il existe une MT qui le reconnaît (elle ne s'arrête pas forcément sur toute entrée).

On note $\mathfrak{R}$ la famille des langages récursifs, et $\mathfrak{R}_e$ la famille des langages récursivement énumérables.

On peut déjà noter que tout ce qui est récursif est en particulier récursivement énumérable, i.e $\mathfrak{R}_e \subset \mathfrak{R}$. $\mathfrak{R}$ est fermé par complément : si $L \in \mathfrak{R}$, il existe une MT qui le reconnaît, et s'arrête sur toute entrée. Donc les entrées sur lesquelles elle s'arrête et ne donne pas de "reconnaissance" sont $\bar{L}$. On peut même construire une machine qui reconnaît $\bar{L}$ via la machine universelle, ou plus simplement en échangeant les états d'acceptation (ce n'est pas exactement une "nouvelle machine", la définition d'acceptation ne fait pas partie de la déifnition de MT, donc il s'agit simplement de réinterpréter la même machine).

Définition : si T est une MT avec une bande de sortie (une bande sur laquelle la machine se contente d'écrire et de se déplacer à droite) qui écrit des mots sur $\sum^*$ séparés par des cases vides. Dans ce cas là, on appelle "langage énuméré" par T l'ensemble de ces mots. On peut caractériser $\mathfrak{R}$ et $\mathfrak{R}_e$ via cette notion.

Proposition : $L \in \mathfrak{R}_e \Leftrightarrow$ il existe une MT qui énumère L, et $L \in \mathfrak{R} \Leftrightarrow$ il existe une MT qui énumère L dans l'ordre lexicographique.

Prouver ça (triv, au pire regarder le cours de Monin).

4 Machines RAM, RASP

Voir Ram machines sur moodle, et ce cours https://www.irif.fr/ carton/Enseignement/Complexite/MasterInfo/Cours/ram.html (vu que le prof a décidé d'expliquer avec les mains au lieu de donner les définitions formelles). Idem pour les RASP
Toute machine RAM peut être simulée par une machine RASP.

5 Complexité