Eléments de calculabilité
Notes de cours - Licence

Christian Attiogbé

Faculté des sciences et des techniques
Université de Nantes
Christian.Attiogbe@irin.univ-nantes.fr

Juillet 1997, mai 2001

Table des matières

Avant-propos

La rédaction de ces notes de cours s'est faite avec le souci de pousser les étudiants à consulter les documents cités en références.
Ces notes doivent donc être prises en tant que telles, c'est à dire support des quelques heures données en amphi et non comme document de référence.

Dans le cadre de ce cours, Didier ROBBES encadre les étudiants avec moi depuis 1997.
Christian RETORÉ vient de nous rejoindre (2000/2001) pour continuer cet encadrement.

Cette équipe pédagogique est à votre disposition pour votre initiation aux outils fondamentaux d'analyse des algorithmes.

Chapitre 1

Notions de Calculabilité éffective (Thèse de Church)

Tous les problèmes ne sont pas solubles algorithmiquement. Un algorithme est une procédure effective de calcul.

– Quels sont alors les problèmes que l'on peut résoudre par l'approche algorithmique ou comment peut-on les caractériser ? (autrement dit, quels sont les problèmes calculables par algorithmes ?)

– Qu'est-ce qui est calculable ?

– Fonctions calculables, ensembles calculables.

1.1 Introduction : Thèse de CHURCH

La notion intuitive de calculabilité effective est exactement traduite par celle d'*ensemble et de fonction récursive*, et donc aussi par celle issue de l'une des autres définition équivalentes :
– calculabilité TURING,
– calculabilité HERBRAND-GODËL,
– calculabilité par programme de tout langage de haut niveau,
– ...
Une interprétation de la thèse de CHURCH : *tout algorithme peut être implanté sur une machine de* TURING.

1.1.1 Tout est mot

Un entier, un tableau, un son, une musique, tout est digitalisable, c'est à dire codable par des mots binaires et ce sans aucune perte d'information.

Pour la machine (ordinateur), le programme et les entrées et sorties de tous types ne sont autres choses que des mots binaires ; c'est l'utilisateur qui interprête ces suites de 0 et 1 comme des objets variés et

pourvus de structures particulières. Pour cela certaines correspondances simples entre mots, entiers, arbres, graphes, etcc sont utilisées.

L'objet de base de la théorie des algorithmes apparaît comme le *mot.*

1.1.2 Encodage

Pour résoudre un problème, un programme informatique doit disposer d'une représentation compréhensible par lui des instances du problème.

Un codage d'un ensemble d'objets abstraits A est une application $e : \mathcal{A} \rightarrow E$ (avec E l'ensemble des chaînes binaires).

Exemples :

- codage en binaire : $e : \mathbb{N} \rightarrow \{0,1\}^*$

- les données de toute sorte peuvent être vues comme des objets composés et par conséquent être codées par la composition des codages des données élémentaires. Ainsi les graphes, les fonctions, les programmes peuvent être codés sous forme de chaînes binaires.

Un algorithme qui resout un certain problème de décision prend en fait en entrée un codage d'une instance du problème. L'instance du problème apparait alors comme une donnée avec une taille relative au codage.

1.2 Formalisation de la calculabilité

Espaces des objets finitaires

Nous considérons pour espaces de base d'objets "finitaires" sur lesquels porte la notion de calculabilité effective (intuitive) les ensembles :
- $\mathbb{N}$ (ensemble des entiers),
- $Seq(\mathbb{N})$ (ensemble des suites finies d'entiers),
- Σ^* (le monoïde libre formé sur Σ), où Σ est un alphabet fini, ainsi que
- les produits de ces derniers.

Les notions d'*ensembles et de fonctions effectivement calculables,* fondées sur la notion intuitive et non formalisée d'algorithme, peuvent être traduites dans un cadre mathématique précis.

Les résultats de cette traduction mathématique des ensembles et fonctions calculables par algorithme assurent que l'on peut définir mathématiquement pour tous espaces X et Y qui sont des produits finis des espaces $\mathbb{N}$, Σ^* et $Seq(\mathbb{N})$,
- une classe de parties, dite classe des *ensembles récursifs* inclus dans X,
- une classe de fonctions partielles de X dans Y dont les domaines sont des parties récursives de X, dite classe des *fonctions récursives* de X dans Y,

telles que :
- toute partie récursive X soit effectivement calculable, toute fonction récursive de X dans Y soit

effectivement calculable ;
– tous les ensembles effectivement calculables connus à ce jour soient récursifs, toutes les fonctions effectivement calculables à ce jour soient récursives.

Exemples

1. Fonction partielle effectivement calculable :
   La division sur les entiers est une fonction partielle effectivement calculable de $\mathbb{N} \times \mathbb{N}\backslash\{0\} \rightarrow \mathbb{N}$
2. Partie effectivement calculable :
   L'ensemble des programmes Pascal syntaxiquement corrects (donc acceptés pas le compilateur Pascal) est une partie effectivement calculable de Σ^* où Σ est l'alphabet de Pascal.

Il y a de **nombreuses approches** pour la *formalisation des notions de fonction et d'ensemble calculables par algorithme.*
Ces approches sont toutes équivalentes :
– Machine de TURING,
– Approche algébrique,
– Approche par les langages de programmation
– Systèmes équationnels de Herbrand

1.2.1 Approche par Machine de TURING

Soient X et Y des produits finis d'espaces $\mathbb{N}$ ou Σ^*. Une fonction partielle f de X dans Y est dite *Turing-calculable* s'il existe une machine de TURING M telle que, pour tout $x \in X$, l'évolution de M, lancée dans l'état initial avec l'entrée x écrite (en binaire pour ce qui concerne les composantes qui sont des entiers) sur ses rubans d'entrée (i.e chaque composante de x est écrite sur les premières cases d'un ruban, toutes les autres cases étant marquées du symbole blanc), la conduit - après un nombre fini d'étapes- à un état final d'acceptation, avec $f(x)$ écrit sur le ruban de sortie si $x \in domaine(f)$, à un état final de rejet si $x \notin domaine(f)$.

1.2.2 Approche par les langages de programmation

Les boucles for et $while$ des langages de programmation permettent de traduire facilement les constructions par récurrence et minimisation. Inversement, ces constructions permettent de traduire les boucles for et $while$ (Exemple : la factorielle en récursif et en itérative).

La traduction du $if\ then\ else$ en termes de récurrence et des fonctions de base est un exercice simple (qui demande néanmoins quelques soins).

Définition

Une **fonction partielle f est calculable** dans un langage de programmation fixé s'il existe un programme π de ce langage qui, lancé sur une entrée u a le comportement suivant :

- π s'arrête en indiquant une erreur de domaine dans le cas où u n'est pas dans le domaine de f,
- π s'arrête en affichant la valeur de $f(u)$ si u est dans le domaine de f.

1.3 Machine de TURING

Qu'est-ce qu'un algorithme ?	Qu'est-ce qui est calculable ?

Plusieurs approches ont été proposées pour définir le concept d'algorithme. ALAN TURING (mathématicien anglais, 1912-1954) est parvenu à formaliser l'idée intuitive qu'un algorithme est une procédure que l'on peut exécuter mécaniquement, sans réfléchir.

TURING modélisa un processus de calcul par une *machine* simple et précise, capable d'effectuer quatre actions élémentaires : *lire, écrire, aller à gauche, aller à droite.*

Si une procédure est décomposable en une séquence des quatre (4) actions, alors on peut lui associer une machine : d'où l'idée de la *Machine de* TURING.

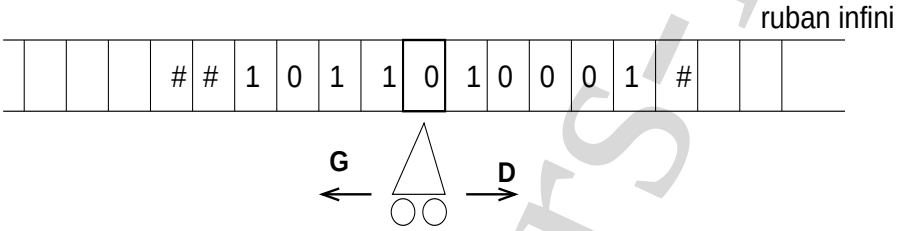


FIG. 1.1 – Machine de TURING

On peut comparer la machine de TURING à une machine à écrire performante ayant :
– une tête mobile qui se déplace **pas à pas** soit vers la gauche (**G**) soit vers la droite (**D**), le long d'un ruban,
– un ruban infini divisé en cases ; chaque case peut recevoir un caractère extrait d'un **alphabet fini** qui comprend un symbole spécial ϕ ou # représentant le blanc.

Quand la tête accède à une case, elle peut remplir deux autres fonctions :
– *lire* le contenu de la case et y
– *écrire* un caractère, en effaçant l'ancien.

Une Machine de TURING peut se trouver dans l'un quelconque d'*un nombre fini d'états internes.* La machine fonctionne en passant d'un état à un autre. La *transition* des états et la tâche que doit remplir la tête mobile sont *programmées à l'avance.*

Un *Turing-programme* ou *T-programme* est un tableau où chaque ligne contient une instruction.

Etat courant	Lecture	Ecriture	Mouvement	Etat suivant
q_0	a	b	D	q_2
q_1	c	ϕ	G	q_0
q_2	ϕ	ϕ	D	q_0
q_1				
q_1				
q_2				

Initialement la machine est supposée être dans l'état q_0. La tête est face à une case fixée d'un ruban presque partout blanc sauf un nombre fini de cases.

On distingue souvent des états spéciaux (q_Y ou q_{oui} et q_N ou q_{non}) pour l'acceptation ou la non acceptation.

Une instruction est un 5-uplet $(q,\ i,\ o,\ m,\ q')$ qui s'interprète comme suit : à l'instant t, si la machine se trouve dans l'état q et la tête lit le caractère i, alors elle imprime le caractère o sur la case courante pour aller ensuite à la case voisine (selon $m = D$ ou $m = G$) ; l'état de la machine à l'instant suivant $t+1$ est q'.

Déterminisme : Pour tout état q et un symbole i, il existe au plus une instruction $(q,\ i,\ o,\ m,\ q')$.

La machine s'arrête lorsque lorsqu'elle atteint un état q pour lequel le symbole lu est i, et il n'existe aucune instruction.

Dans le langage mathématique, une machine de TURING est un 5-uplet $(Q,\ V,\ \delta,\ q_o,\ q_f)$ avec :
– Q un ensemble fini appelé ensemble d'états,
– V un ensemble fini comprenant le symbole blanc ϕ,
– $\delta : Q \times V \rightarrow V \times \{G, D\} \times Q$, une fonction partielle dite fonction de transition,
– q_0 l'état initial et
– q_f l'état final.

TURING a montré qu'on peut combiner plusieurs machines simples pour obtenir une machine capable d'effectuer tous les calculs que l'on sait décrire explicitement.

TURING a montré aussi qu'on peut simuler les actions d'une machine de TURING quelconque par une autre machine appelée machine de TURING Universelle. Ce résultat s'est vu concrétiser par les ordinateurs actuels dits de VON NEWMANN (les programmes sont maintenant en mémoire alors qu'avant seules les données y étaient).

1.3.1 Utilisation de la machine de TURING

Elle est utilisée de différentes façons :
– pour *reconnaître un langage*
– calculer une fonction,
– ...

1.3.2 Exemples de machines de TURING

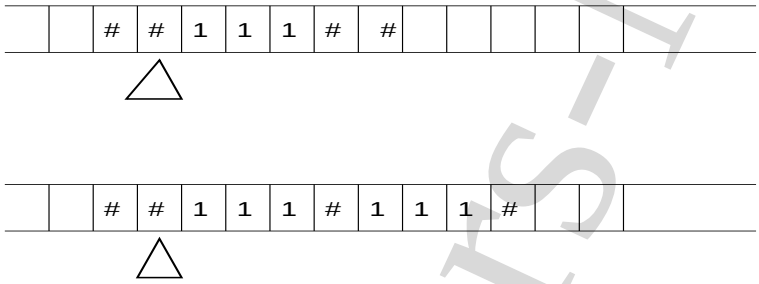
1. Une machine de TURING pour la parité binaire (nombre de 1 paire) :

Etat courant	Lecture	Ecriture	Mouvement	Etat suivant
q_0	0	0	D	q_0
q_0	1	1	D	q_1
q_0	ϕ	ϕ	G	q_Y
q_1	0	0	D	q_1
q_1	1	1	D	q_0
q_1	ϕ	ϕ	G	Q_N

2. Donner le contenu du ruban ainsi que l'état atteint par la machine suivante, avec l'entrée `abaa` :

Etat courant	Lecture	Ecriture	Mouvement	Etat suivant
q_0	a	a	D	q_0
q_0	b	c	D	q_1
q_0	c	b	D	q_0
q_0	ϕ	ϕ	D	q_f
q_1	a	b	D	q_1
q_1	b	a	D	q_0
q_1	c	c	D	q_0
q_1	ϕ	ϕ	D	Q_f

3. On désire concevoir une machine de TURING qui recopie une chaîne de caractères inscrite sur le ruban à l'instant $t = 0$.
Pour simplifier, soit l'alphabet $V = \{0, 1\}$. Supposons que la disposition initiale est de la forme : on souhaite arriver à :



1.3.3 Langage accepté par une Machine de TURING

Le langage $L(M)$ accepté par une machine de TURING M est l'ensemble des mots w tels qu'il existe une suite de dérivations (une suite de transitions) se terminant dans un état q_y.

Langage accepté et langage décidé

Une machine de TURING acceptant un langage ne décrit pas toujours une procédure effective pour reconnaître ce langage. En effet, si on considère la suite des configurations à partir de l'état initial, plusieurs cas peuvent se présenter :

1. cette suite contient une configuration où l'état est accepteur. Dans ce cas le mot est accepté dès que la configuration où apparait l'état accepteur est atteinte.
2. la suite de configuration se termine par une configuration pour laquelle il n'y a pas de configuration suivante parce que :
   – soit la fonction de transition n'est pas définie pour cette configuration,
   – soit la fonction de transition précise un déplacement à gauche de la tête de lecture alors que celle-ci se trouve sur la première case du ruban.
3. la suite de configurations ne passe jamais par un état final et est infinie.

Pour les deux premiers cas, la machine nous permet de déterminer si oui ou non le mot appartient au langage considéré (oui dans le premier, non dans le deuxième).

Dans le troisième cas, nous n'obtenons jamais de réponse ; en effet à aucun moment nous ne sommes sûrs que le mot ne sera pas accepté, car nous ne pouvons jamais affirmer que la machine ne s'arrêtera pas. Ce n'est pas parce que la machine est passée par de très nombreuses configurations sans atteindre d'état accepteur que cela ne se produira pas par la suite. C'est donc la présence d'exécutions infinies qui fait qu'une machine de TURING acceptant un langage ne définit pas une procédure effective (un algorithme) pour reconnaître ce langage.

1.3.4 Langage décidé par une Machine de TURING

Considérons l'exécution d'une machine de TURING sur un mot w comme la suite de configurations $C_0 \vdash C_1 \vdash C_2 \cdots$ maximale, c'est à dire telle que :
– soit elle est infinie,
– soit elle se termine dans une configuration dont l'état est accepteur,
– soit elle se termine par une configuration à partir de laquelle aucune configuration n'est dérivable.
Le langage décidé par une machine de TURING se définit comme suit :

Définition : un langage L est décidé par une machine de TURING M si
– M accepte L,
– M n'a pas d'exécution infinie (la machine atteint q_N ou q_Y).
Par conséquent un langage décidé par une machine de de TURING peut être reconnu par une procédure effective.

1.3.5 Langages récursifs et récursivement énumérables

Les **langages décidés** par les machines de TURING **sont appelés récursifs**.
Les **langages acceptés** par les machines de TURING sont appelés **récursivement énumérables**.

Définition : un langage est récursif s'il est décidé par une machine de TURING.

Définition : un langage est récursivement énumérable s'il est accepté par une machine de TURING.

1.3.6 Fonctions calculables par une machine de TURING

Définition : une machine de TURING calcule une fonction $f : \Sigma^* \rightarrow \Sigma^*$ si pour tout mot d'entrée w, elle s'arrête toujours dans une configuration où $f(w)$ se trouve sur le ruban.

Les fonctions calculables par une procédure effective sont les fonctions calculables par une machine de TURING.

1.3.7 Machines de TURING non-déterministe

1.4 Formalisation des problèmes

Si un problème doit être résolu par une procédure effective (algorithme), il faut que ses instances soient représentées par une fonction accessible à cette procédure effective.

Par exemple si la procédure effective est un programme Pascal, alors l'instance du problème (les données sur lesquelles le programme opérera) devra être représentée par une collection d'entiers, de chaînes, ···

Donc il faut avoir un alphabet (Σ) pour construire les mots (qui représentent l'instance du problème).

Un problème peut donc être caractérisé par l'ensemble des mots qui sont ses instances "oui" (décision positive). Un ensemble de mots est un langage.

Précisément, un problème est caractérisé par le langage des encodages de ses instances et résoudre un problème revient à reconnaître les instances positives de ses encodages.

Résoudre un problème = reconnaître le langage des instances "oui" du problème.

On établit que :
– un problème se formalise par un langage,
– une procédure effective se définit en termes de langages décidés par une machine de TURING.
Avec ces éléments, il est possible de démontrer que certains problèmes ne sont pas solubles par une procédure effective, c'est à dire ne sont pas décidés par une machine de TURING.

1.5 Extensions des machines de TURING

1.5.1 Machines à ruban multiples

1.5.2 Machines à mémoire à accès direct (RAM)

1.6 Sur la terminaison

La notion de machine de TURING est parfaitement spécifiée. Cependant la notion de machine de TURING qui s'arrête sur chacune des des entrées ne l'est pas. Par conséquent, le problème de l'arrêt des machines de TURING est indécidable.

De la même façon, les langages de programmation de haut niveau sont parfaitement spécifiés alors que la notion de programme qui s'arrête sur chacune de ses entrés n'est pas effective. Par conséquent, l'ensemble des programmes Pascal (par exemple) à zéro argument qui s'arrêtent est non récursif.

1.7 Conclusion

Les difficultés inhérentes à la notion de calculabilité viennent du fait que les procédures assez puissantes pour donner tout le champ de la calculabilité s'accompagnent d'un problème de la terminaison non décidable.

Bibliographie

[AA89] J. Ullman A. Aho, J. Hopcroft. *Structures de données et algorithmes*. Collection IIA. InterEditions, 1989.

[CF94] M. Soria C. Froidevaux, M-C. Gaudel. *Types de données et algorithmes*. Ediscience. McGrawHill, 1994.

[eCB84] B. Meyer et C. Baudoin. *Méthodes de programmation*. Eyrolles, 1984.

[eIG97] A. Arnold et I. Guessarian. *Mathématiques pour l'informatique*. Enseignement de l'informatique. Masson, 1997. 3e édition.

[SF96] Robert Sedgewick and Philippe Flajolet. *Introduction à l'analyse des algorithmes*. International Thomson Publishing, 1996.

[Ste90] Stern. *Fondements mathématiques de l'informatique*. MacGrawHill, 1990.

[Wil89] H.S. Wilf. *Algorithmes et complexité*. Collection logique mathématiques et Informatique. Masson - Prentice Hall, 1989.

[Xuo92] N.X. Xuong. *Mathématiques discrètes et Informatiques*. Collection logique mathématiques et Informatique. Masson, 1992.