Calculabilité

L2 Informatique – Université Paul Sabatier

Documents

1. Introduction à la calculabilité, P Wolper, InterEditions 2001.
2. Introduction to Theory of Computation, M Sipser, PWS Publishing Company 1997.

1 Introduction

Nous présentons dans ce chapitre les principaux résultats de la calculabilité ("l'existence de méthode de calcul d'un résultat").

Dans le cadre d'une méthodologie de la programmation, nous aborderons, dans les chapitres suivants, quelques principes concernant la complexité ("Efficacité d'une méthode de calcul").

Nous cherchons à comprendre les limites des programmes informatiques (les problèmes que nous pouvons résoudre et ceux que nous ne pouvons absolument pas résoudre).

Il est essentiel aussi de comprendre, dans le cas de l'existence d'algorithmes pour résoudre un problème donnée, s'il existe un algorithme efficace (du point de vue des ressources nécessaires en temps et/ou en espace mémoire).

Pendant des siècles les mathématiciens ont décrit et utilisé des méthodes de calcul permettant de résoudre des problèmes. Dans d'autres sciences, il était plus simple de refuter une proposition qui ne respecte pas les principes. Par exemple, la construction d'un moteur basé sur le mouvement perpétuel, en présence de frottements et sans consommer de l'énergie, contredit les lois de la mécanique.

Des problèmes de mathématiques sont restés ouverts pendant des siècles avant qu'une solution soit trouvée. Il aura fallut attendre les années trente du siècle dernier pour que la notion de calcul soit définie d'une manière précise. Depuis, les mathématiciens et les logiciens possèdent les outils et les moyens permettant de savoir exactement quels sont les problèmes traitables et, avec grande surprise, d'identifier certains problèmes qui ne le sont pas dans l'absolu. Ces derniers sont désormais appelés problèmes indécidables. C'est dans ce cadre que la science de l'informatique s'est fondée.

2 Calculabilité et décidabilité

L'étude de la calculabilité et de la décidabilité va nous permettre de comprendre les limites de l'informatique à travers les concepts et les outils suivants:

1. algorithmes
2. fonctions calculables
3. problèmes décidables, non décidables,...

Les mathématiciens ont recherché et construit des algorithmes bien avant que le concept d'algorithme ne soit élucidé (Monsieur Jourdain faisait de la prose sans le savoir).

Définition 1 *Un problème est une question, souvent paramétrée. En affectant des valeurs aux paramètres nous parlons d'une instance du problème. Le problème existe indépendamment de toute méthode d'obtention de la réponse.*

Exemple 1

- *Un nombre naturel N est-il pair ? Ceci concerne tout $N \in \mathbb{N}$*
- *Est ce que le nombre 4 est pair ? Il s'agit d'une instance du problème ci-dessus*

Remarque 1 *Pour un ordinateur, le programme, les entrées et les sorties de tous types ne sont que des mots(chaînes) binaires.*

Un problème peut avoir:

- Au moins une solution (Il existe au moins une méthode de calcul de la réponse)
- 0 solution (il n'est pas possible d'obtenir la réponse)
- On ne sait pas encore s'il est possible ou non d'obtenir une réponse

Exemple 2

1. *Un nombre naturel N est-il premier ?* **Il existe des solutions**
2. *Est-ce que l'exécution d'un programme quelconque P se termine?* **0 solution**
3. *La conjecture de Fermat est-elle valide?* **solution trouvée après plus de trois siècles**
4. *la conjecture de Goldbach est-elle valide?* **on continue à chercher une solution**

Définition 2 *Un algorithme est la formalisation d'une méthode effective, complète et sans ambiguïté pour la manipulation de données (et qui termine après un temps fini).*

Une méthode effective de calcul doit satisfaire les obligations suivantes:

1. l'algorithme consiste en un ensemble fini (fixé) d'instructions simples et précises qui sont décrites avec un nombre fini de symboles
2. l'algorithme doit toujours produire le résultat en un nombre fini d'étapes
3. l'algorithme peut en principe être suivi par un humain avec seulement du papier et un crayon
4. l'exécution de l'algorithme ne requiert pas d'intelligence de l'humain sauf celle qui est nécessaire pour comprendre et exécuter les instructions

Exemple 3
Déterminer si un nombre naturel N est pair ou non
On sait que pour les instances suivantes:

- $N = 1$, *N n'est pas pair*
- $N = 4$, *N est pair*

D'une façon générale: On sait que pour déterminer que N est pair il faut qu'il soit divisible par 2. Dans le cas contraire N n'est pas pair. Il suffit de procéder de la manière suivante:

- *faire la division de N par 2 (en représentation binaire, c'est une décalage à droite)*
- *regarder le reste r de cette division*
- *si $r = 0$, il est $VRAI$ que "N est pair"*
- *sinon il est $FAUX$ que "N est pair"*

Cette méthode effective n'est qu'une définition intuitive assez claire, mais elle manque de précisions. Il faudra formaliser pour préciser les points suivants:

- instruction simple et précise
- intelligence requise pour exécuter les instructions

2.1 Thèse de Church

Il s'agit d'une thèse (qui n'a pas été démontrée ni rencontré de contradiction jusqu'à maintenant). Elle affirme qu'une méthode effective peut être exprimée par un ensemble donné de règles de calcul. Cette formalisation a été exprimée de différentes manières. On peut montrer que toutes ces formulations, bien que fondées sur des idées assez différentes, sont équivalentes. Elles décrivent exactement le même ensemble de fonctions.

- λ calcul (A. Church)
- fonctions récursives (K. Goëdel, N Herbrand, S. Kleene)
- machine de Turing (A. Turing)

Définition 3 *Une fonction est dite calculable s'il existe un algorithme permettant d'obtenir l'image de tout élément de son domaine. Intuitivement une fonction f est calculable sur un argument x, s'il existe un algorithme qui permet de calculer, pour tout x, la valeur $f(x)$.*

Remarque 2 *La machine de Turing est le modèle le plus simple et le plus proche des intuitions concernant la méthode effective, capturant de la façon la plus naturelle ce que l'humain peut calculer, a donné naissance à la thèse de Church ou la thèse Church-Turing.*
D'autres modèles et formalismes ont été aussi démontrés équivalents comme: RAM, POST, Caml, PYTHON, C, C++, Java...

Ceci est considéré comme acquis par la communauté scientifique en ce qui concerne les machines déterministes. Les portes ont été laissées ouvertes pour de nouvelles définitions concernant les notions d'algorithmes et de machines étendus à l'hypercalcul, l'aléatoire ou la physique quantique par exemple.

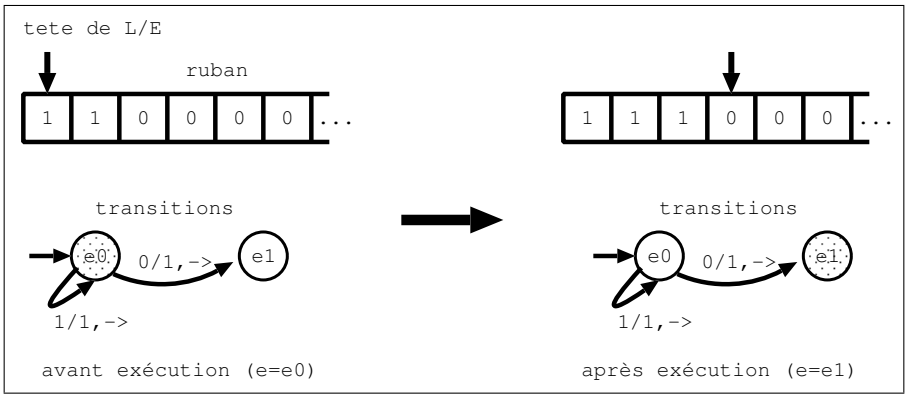
2.2 Machine de Turing: MT

Reste le modèle de calcul le plus proche des principes de la physique classique et d'une méthode effective et par conséquent la MT est devenue la référence pour toute proposition d'un formalisme ou d'un modèle de calcul.
Une machine de Turing consiste en:

- un ruban infini de cellules consécutives
- une tête de lecture/écriture ou curseur qui pointe sur une cellule et qui se déplace d'un pas, à gauche ou à droite, à partir de sa position courante
- les symboles utilisés appartiennent à un alphabet fini par exemple{0, 1}
- un ensemble fini d'états, parmi eux, on distingue un état initial
- la MT a un comportement automatique en fonction des transitions qui sont décrites dans une une table de transition (**le programme**)
- une transition exprime la lecture/écriture d'un symbole, le mouvement du curseur et le nouvel état de la machine **(e, a, a', direction, e')**:
  - e est l'état courant de la machine
  - a est le symbole lu: contenu de la cellule pointée par le curseur

- a' est le symbole écrit dans la cellule pointée par le curseur
- direction précise le mouvement du curseur d'un pas à gauche ou à droite
- e' est le nouvel état de la machine

- une transition est donc composée par une condition **(e, a)** et une action **(a', direction, e')**
- la MT s'arrête s'il n' y a pas de transition avec le couple (e,a)
- elle s'arrête aussi, si on est au début du ruban et la transition dirige vers la gauche
- la MT peut ne jamais s'arrêter



Exemple 4 calculer $n+1$, au départ $n=2$

- *L'alphabet utilisé est constitué par* $\{0,1\}$
- *Les cellules contiennent 0 pour exprimer qu'elles sont vides*
- *On représente un naturel* $n \geq 1$ *par une suite de* n 1 *(notation unaire)*
- *On considère un ensemble à deux états* $\{e_0, e_1\}$ *où* e_0 *est l'état initial de la machine*
- *Le mot d'entrée est 2 càd :*$\{1100....0.....\}$*, le curseur pointe sur le premier 1 à gauche*

Table de transitions

- *t1:*(e_0*,1, droite,1,*e_0*)*
- *t2:*(e_0*,0,droite,1,*e_1*)*

Le comportement automatique va être: t1; t1; t2 et on s'arrête car la table ne contient pas de transition avec $(e_1, 1, ...)$

Nous pouvons introduire et étudier les principes et les propriétés de la calculabilité à travers les problèmes de décision. Ceci pourrait s'étendre sur les autres problèmes.

- un problème de décision: pour $N \in \mathbb{N}$, répondre 1 si N est pair, 0 sinon.
- un problème qui n'est pas de décision: calculer la longueur du plus court chemin entre deux sommets d'un graphe.

Définition 4 *Décidabilité*

Un problème P est décidable, si et seulement si il existe un algorithme A qui résout P.

Remarque 3 *Les problèmes de décisions sont des problèmes binaires.*

L'algorithme répond, pour un problème de décision P, par OUI ou par NON à toute instance de P.

L'ensemble des problèmes de décisions n'est pas dénombrable.

Considérons, l'ensemble FD des fonctions $f : \mathbb{N} \to \{0, 1\}$ Cet ensemble n'est pas dénombrable. Utilisons le diagonale de Cantor (méthode utilisée pour montrer que les réels ne sont pas dénombrables).

Raisonnement par l'absurde: Si FD est dénombrable ,c.à.d $FD = \{f_1, f_2, ...f_i, ...\}$. Construisons la fonction $fd \in FD$ telle que pour tout $i \in \mathbb{N}, fd(i) = 1 - f_{i+1}(i)$. Cette fonction appartient bien à FD et est différente de tous les f_i appartenant à FD ce qui est contradictoire.

En sachant, par exemple, que l'ensemble des programmes en C est dénombrable, on peut déduire qu'il existe des fonctions non calculables.

Exemple 5

1. *Un nombre N est-il premier?* ***Problème décidable****. Un nombre naturel $N > 1$ est premier si et seulement s'il existe une fonction calculable* `prem`*, prenant en paramètre N et répondant (en un temps fini):*

   - *NON si on trouve un diviseur différent de 1 et de N*
   - *OUI si on ne trouve pas de diviseur autre que 1 et N*

   Il suffit de chercher un diviseur d parmi les valeurs $x : x^2 \leq n$. Pour n fixé, l'ensemble des x est fini ce qui permet, en parcourant les x un par un, d'aboutir toujours à la fin de cette recherche.

2. *Problème de l'arrêt*

   Existe-t-il un algorithme qui permet de répondre par $vrai$ si un programme quelconque s'arrête et par $faux$ sinon?

   Il y a des programmes qui s'arrêtent comme:

   "en partant de $x = 0$, incrémenter x tant que $x < N$" ($A1(N)$)

   et d'autres qui ne s'arrêtent jamais comme :

   "en partant de $x = 2$, tant que x est pair, ajouter 2 à x" ($A2$).

   Le problème de l'arrêt est indécidable: raisonnons par l'absurde

- *Supposons qu'il existe un algorithme $Halt(A, P)$ qui, pour tout programme A et tout paramètre P, répond par vrai si $A(P)$ termine et par $faux$ sinon.*
- *Construisons le programme mystere qui prend en entrée un programme X et utilise l'algorithme $Halt$, une boucle infinie dénotée $INFINI$ et un programme qui termine dénoté $STOP$. mystere est défini de la manière suivante:*

$$\text{mystere}(X) = \textbf{si } Halt(X, X) \textbf{ alors } INFINI \textbf{ sinon } STOP$$

- *Prenons* $X =$ mystere*:*
  - *si* $Halt(\text{mystere}, \text{mystere})$ *retourne vrai,* mystere(mystere) *devrait terminer, mais* mystere(mystere) $= INFINI$ *ne termine pas.*
  - *si* $Halt(\text{mystere}, \text{mystere})$ *retourne faux,* mystere(mystere) *devrait ne pas terminer, mais* mystere(mystere) $= STOP$ *termine.*

  Dans les deux cas, on arrive à une contradiction. Donc $Halt$ ne peut pas exister.

Remarque 4 *La démonstration de l'indécidabilté est un problème difficile à résoudre et souvent nous restons sans solution. Une démonstration par l'absurde ou par réduction s'impose. Elle pourrait être directe (exemple de l'indécidabilité du $Halt$) Souvent nous restons devant des énoncés de problèmes qui demeurent sans démonstration dans un sens ou dans un autre, d'autres ont été démontrés en consommant beaucoup d'énergie et beaucoup de temps.*

Exemple 6

1. *Problème des équations Diophantiennes(XVIIème siècle): (10ème problème d'Hilbert posé en 1900);*

   Existe-t-il un algorithme permettant de vérifier pour toute équation d'un nombre quelconque d'inconnues et de coefficients entiers, si elle possède des racines entières? En 1970 il a été demontré qu'il s'agit d'un problème indécidable.

   Ceci n'empêche pas l'existence des cas ou d'instances qui ont des solutions:

   - *La conjecture de Fermat: elle a été demontrée après plus de 3 siècles d'efforts,*

     Il n'existe pas de nombres entiers non nuls a, b, c tels que $a^n + b^n = c^n$, dès que $n > 2$

2. *Problème de Correspondance de Post PCP (1946)*

   Il s'agit d'une autre approche que celle de la machine de Turing (qui lui est équivalente). Considérons deux listes (U, V), de taille $N \geq 1$ de mots d'un alphabet, A, d'au moins deux symboles. Existe-t-il un algorithme permettant de répondre par OUI si on peut obtenir le même mot en juxtaposant un même nombre de mots, de la première liste et de la deuxième liste, sinon la réponse est NON.

   $U = [u_1, u_2, ...u_N]$

$V = [v_1, v_2, ...v_N]$

Existe-il une suite $I_1, I_2...I_K$ *dans l'intervalle* $[1, N]$*, avec* $K \geq 1$.

Pour $x \neq y$ *on peut avoir* $I_x = I_y$

$[u_{I_1}.u_{I_2}...u_{I_K}] = [v_{I_1}.v_{I_2}...v_{I_K}]$ *?*

Bien que ce problème paraisse simple il est, dans sa généralité, indécidable.

3. *Conjecture de Goldbach (1742) : tout nombre pair de* $\mathbb{N}$ *supérieur strictement à 3 est la somme de deux nombres premiers*

   $$(\forall N : N \textbf{ mod } 2 = 0 \wedge N > 3 \rightarrow (\exists X, Y : \text{Premier}(X) \ \wedge \ \text{Premier}(Y) \wedge N = X + Y))$$

   Nous supposons l'existence d'une fonction `Somme2PremiersDe(n)` *qui retourne* 1 *si on trouve deux nombres premiers dont la somme est égale à* n *sinon elle retourne* 0.

   Selon cette conejcture, la boucle suivante est infinie:

   ```
   n=4
   while ( Somme2PremiersDe(n))
     n=n+2;
   ```

   Personne n'a su demontrer cette conjecture pour pouvoir démontrer que c'est une boucle infinie. Rien ne garantit l'existence de la fonction `Somme2PremiersDe(n)`.

4. *Suite de Syracuse On construit une suite à partir d'un entier* > 0*: s'il est pair, on le divise par* 2*, sinon on le multiplie par* 3 *et on lui ajoute* 1.
   En répétant l'opération on obtient une suite.
   La conjecture de Syracuse est l'hypothèse selon laquelle la suite de Syracuse commançant par un Naturel > 0 *atteint* 1.

   ```
   while (x!=1)
     if (x % 2 == 0)
       x=x/2;
     else
       x=3*x+1;
   ```

   Personne n'a su démontrer que cette boucle s'arrête toujours à 1.

2.3 Conclusion

Nous pouvons remarquer qu'il y a plusieurs catégories de problèmes:

- Ceux qui possèdent une solution, càd. qu'il existe un programme qi se termine en donnant la réponse. (Problèmes décidables)

- Ceux qui ne possèdent pas de solution, càd. il n'existe pas d'algorithme (et par conséquence un programme) répondant à ces problèmes. (Problèmes indécidables)
- Ceux pour lesquels il existe un programme qui peut ne pas s'arrêter (Problèmes ouverts).

Dans la troisième catégorie, on se trouve souvent devant des problèmes ouverts (partiellement décidables, récursivement énumérables).

Dans le chapitre 3, nous introduisons le modèle récursif. Ensuite, dans les chapitres qui suivent, nous adoptons une méthodologie en quatre étapes pour développer un programme(en petit):

1. Analyser et comprendre le problème pour décrire l'objectif de l'algorithme à construire.
2. Spécification de l'algorithme: formaliser sa pré condition(proriétés en entrée) et sa post condition(propriétés de l'objectif/sortie).
3. Choisir un modèle de solution en fonction de contraintes de complexité et d'efficacité.
4. Développer le programme avec preuve.

2.4 Exercice

Compléter l'**exemple 4** pour que le curseur de la MT s'arrête à la case de départ.
Solution
Table de transitions

- t1:(e_0,1, droite,1,e_0)
- t2:(e_0,0,droite,1,e_1)
- t3:(e_1,0,gauche,0,e_1)
- t4:(e_1,1,gauche,1,e_1)