Introduction

Théorie de la calculabilité et théorie de la complexité

Deux questions fondamentales

- Qu’est ce qui est calculable par un ordinateur ? C’est la question à laquelle s’attaque la théorie de la calculabilité.
- Qu’est ce qui est calculable efficacement par un ordinateur ? C’est la question à laquelle s’attaque la théorie de la complexité. L’impact pratique de cette théorie est évidemment très important.

Théorie de la Calculabilité

- Premières réponses scientifiques à partir de 1935 (Afonzo Church, Kurt Gödel, Alan Turing)
- Thèse de Church : La réponse à la question est indépendante de la technologie (tous les modèles de calculs raisonnables sont équivalents)
- Conséquence : « langage C comme modèle »

Exemple de problème non décidable

- Peut-on déterminer à la compilation si un programme P va boucler sur une donnée D ?

- La réponse à la question est NON (preuve dans la suite du cours)

Théorie de la Complexité

- Qu’est ce qui est calculable efficacement par un ordinateur ?
- Difficulté : définir la notion efficacement
- Les énoncés du type le problème A, peut-être programmé par le programme P dans le langage L et s’exécute en t secondes sur la donnée D sur l’ordinateur O ne sont que de peu d’intérêt sur le problème A.

Définition plus formelle d’efficacement

- Par convention on admet qu’un problème n’est pas soluble en pratique s’il n’existe pas d’algorithme pour le résoudre qui s’exécute en temps polynomial par rapport à la taille du problème (nombre de bits pour coder la donnée).

Pourquoi cette convention ?

- Si on a un algorithme polynomial alors si on multiplie la puissance d'un ordinateur on multiplie aussi la taille des problèmes que l'on peut traiter par unité de temps.
- Par exemple si on a un algorithme en $O(n^3)$, chaque fois que la puissance est multipliée par 8, la taille des données traitables en 1 seconde est multipliée par 2.

Pourquoi cette convention ?

- Par contre un ordinateur qui fonctionnerait à 2 GHz depuis le Big-Bang (si l'on admet cette théorie) aurait exécuté moins de 2^{90} instructions.
- Donc un algorithme en $O(2^n)$ est de peu d'intérêt pratique.

Théorie de la Complexité

- Un problème est facile s’il existe un algorithme polynomial par rapport à la taille de la donnée pour le résoudre et il est difficile sinon.
- Avec cette définition la notion d’efficacement calculable devient indépendante de la technologie.

Conclusion

Classification :

- un problème est-il calculable ou non ?
- S’il est calculable l’est-il efficacement ou pas ?

Le concept de Réduction

```
ProcP(D)
    début
    .......
    procQ(f(D))
    .......
    fin
```

- ProcP résout le problème P et ProcQ le problème Q.

Quelques constats

ProcP(D)

début ..... ProcQ(f(D)) ..... fin

- Q calculable => P calculable

  (sous les hypothèses f et .... calculable évidemment)
- Q facile => P facile

  (sous les hypothèses f et .... calculable efficacement évidemment)
- P non calculable
- P difficile

Quelques constats

ProcP(D)

début ..... ProcQ(f(D)) ..... end

- Q calculable => P calculable

  (sous les hypothèses f et .... calculable évidemment)

- Q facile => P facile

  (sous les hypothèses f et .... calculable efficacement évidemment)

- P non calculable => Q non calculable (par l'absurde supposons que ProcQ existe) (sous les hypothèses f et .... calculable)

- P difficile => Q difficile (par l'absurde)

  (sous les hypothèses f et .... calculable efficacement)

Attention de ne pas faire le raisonnement inverse (et faux). Il existe peut-être une façon plus efficace d'écrire une procédure qui résout P.

Pour la suite

- La première moitié du cours sur la théorie de la calculabilité : comprendre les limites de l'informatique, notions un peu théoriques mais qui doivent faire partie de la culture d'un informaticien. Vous n'en ferez en Master que si vous êtes intéressés.
- La fin du cours sur la théorie de la complexité. L'impact pratique est beaucoup plus fort. La plupart des cours de master utiliseront les notions abordées.

Chapitre 1. Numérotations

- Le premier principe sur lequel repose la théorie de la calculabilité est le suivant :

  Tout objet informatique peut-être codé en entiers naturels.

- Ce principe nous permet sans perte de généralités (spdg) de ne considérer que des fonctions de N dans N.

Enumérations.

- Nous allons dans ce chapitre étudier des objets informatiques classiques et donner des fonctions qui permettent de les coder et les fonctions de décodage associées. Il s'agit en général de bijections entre N et l'ensemble des objets à coder.
- Ces fonctions permettent par exemple de parler du $i^{ième}$ objet de l'ensemble ou d'afficher tous les objets de l'ensemble.

Les Entiers relatifs

- Cet ensemble peut être par exemple codé simplement

  Par exemple si $x \geq 0$ $c(x)=2x$ sinon $c(x)=-2x-1$

-

x	0	-1	1	-2	2	-3	3	-4	4
c(x)	0		2		4		6		8

Les Entiers relatifs

- Cet ensemble peut être par exemple codé simplement

  Par exemple si $x \geq 0$ $c(x)=2x$ sinon $c(x)=-2x-1$

-

x	0	-1	1	-2	2	-3	3	-4	4
c(x)	0	1	2	3	4	5	6	7	8

Décodage(z) : si z est impair alors retourner -(z+1)/2 sinon retourner z/2

Affichage des Entiers relatifs

x	0	-1	1	-2	2	-3	3	-4	4
c(x)	0	1	2	3	4	5	6	7	8

Décode(z) : si z est impair alors retourner -(z+1)/2 sinon retourner z/2

z=0; while(1) { écrire(decode(z) ; z++}

Permet d'écrire tous les entiers relatifs dans le sens où tout entier relatif sera écrit au bout d'un temps fini. On dira que ce programme énumère les entiers relatifs.

Autres ensembles

- Couples d’Entiers
- Généralisation au codage de triplets (ou de n-uplets)
- Mots
- Listes d’Entiers
- Procédures C

Echec du codage

- Réels

  Ils y en a trop : pas de bijection

  Représentation par des flottants
- Procédures C qui sont définies sur toutes les données (voir cours suivant)

Codage des couples

- $c(x,y) = (x+y)(x+y+1)/2 + y$

(0,0)	(1,0)	(0,1)	(2,0)	(1,1)	(0,2)	(3,0)	(2,1)	(1,2)	(0, 3)
0	1	2	3	4	5	6	7	8	9

Décode(z) : trouver le plus grand t tel que $t(t+1) \leq 2z$ (boucle tq)
y= z-t(t+1)/2
x= t-y

Exemple : Quel est le couple de code 63 ?
t= 10 (car 10*11=110 et 11*12=132) donc y=63-55=8 et x=2.
Vérification : c(2,8)=10*11/2 + 8 = 55 +8 = 63