Calculabilité et Complexité - Nicolas Ollinger

Alexandre Masson

15 Janvier 2013

Table des matières

1 Fondements de l'informatique

le But du cours est de nous donner quelques éléments culturels pour comprendre comment certaines choses seront toujours valable dans 50 ans et pourquoi c'est de la science.

Computability en Anglais , on s'intéresse à la propriété d'être calculable.

Qu'est ce qui est calculable ?

Complexité s'intéresse à la meilleur complexité que l'on peut obtenir avec un algorithme. Existe il des problèmes intrinsèquement plus complexes que d'autres (ex : tri , pas possible de faire mieux que O(nlog(n))).

Il est possible ici de faire un parallèle entre Programmation et Algorithmique. Ces deux choses travaillent sur : comment trouver de programme (ou comment construire un programme qui résous un problème) et calculabilité et complexité détermine l'existence d'un programme pour résoudre un problème.

thèse de Church-Turing(rien d'intuitif, quelque-soit le langage de prog, on ne pourra pas calculer plus de choses).

Les programmes qui calculent des fonctions partielles , peuvent ne pas répondre à des problèmes.

Un interprète, prend un programme et une donnée et renvoi le résultat du programme sur la donnée.Une fonction universelle existe dans tout modèle de calcul(Travaux de Alan Turing).

Un programme qui affiche son propre code dans son exécution, existe dans tous les langages de programmation. Théorème de récurrence, pour toute fonction , il existe une fonction auto-référente(qui se prend elle même en paramètre) qui prend e code en paramètre et affiche son code.

il n'existe pas de programme qui prend un autre programme en paramètre et dit si celui-ci s'arrête ou pas.

2 Calculabilité

Définition Études des modèles de calcul et des propriétés qui sont indépendantes du modèle.

On s'intéresse :
- aux programmes vus à la fois comme :
  - chaîne de caractères (code source) : un mot sur un alphabet fini
  - un objet qui code une fonction (soit des chaînes -> les chaînes, soit des Entiers -> les Entiers).

  Remarque on confondra allègrement et sans scrupules les ensemble E*(ensemble de tous les mots d'un alphabet), N*(ensemble des couples d'entiers de taille*) , N ,... qui sont codables bijectivement et récursivement (codage et décodage calculables par un programme) les uns en les autres.

- problèmes de décision : problème auquel on répond par oui ou non, c'est un langage (donc un sous ensemble de E*).

Intuitivement Une fonction f est : E* -> E* partielle est calculable si il existe un programme tel que : $\forall$ x $\in \sigma$ * .

Un problème de décision P $\subset \sigma$ * est décidable si fp : x -> oui si p est x $\in$ P est calculable , non sinon.

3 Motivation informelle(approche naïve)

Fixons un langage de programmation impératif IMP idéalisé. Code Source,entrée,sortie $\in$ UTF8*. Prenons eval(p,x) = y si les programme sur l'entrée x s'arrête en renvoyant y, $\perp$ sinon.

ϕ p : σ * -> σ *
ϕ (x) = eval(p,x).

Question : Est ce que toute fonction est IMP-calculable ? NON, il suffit de compter !
Prop : Toutes les fonctions ne sont pas IMP-calculable.
Démonstration : Comparons le nombre des fonctions et le nombre des programmes, σ * est un ensemble infini dénombrable.
σ * -> σ * n'est pas dénombrable.
donc il existe des fonctions non IMP-calculable. σ * est dénombrable : il suffit d'énumérer par taille croissante et ordre lexicographique à taille fixée.

σ * -> σ * n'est PAS dénombrable.

Procédé de Cantor :
par l'absurde , on suppose que σ * -> σ * est dénombrable : énumération(ϕ i) tel que ϕ i = σ * σ *

Remarque : l'entrée est <p,x> codage dans la paire(p,x).

Proposition : Il existe un programme universel U tel que $\forall$ p $\forall$ X U(<p,x>) = eval(p,x), il suffit de prendre pour U un interprète IMP codé en IMP.

Proposition : IMP-arrêt est IMP-indécidable.

Démonstration : Par l'absurde. Supposons que IMP-arrêt soit IMP-décidable, idem qu'il existe un programme débug tel que ϕ debug (<p,x>) = oui si ϕp(x) != $\perp$, NON sinon.
On construit un programme malicieux comme suit : on lit X, on execute debug, si debug dis oui boucle infinie while 1 , sinon return , masi alors , ϕ malicieux(malicieux) répond oui si imp-arrêt dis non et non si imp arrête dis oui, donc ce programme ontredit IMP-arrêt CONTRADICTOIRE!!! donc IMP-arrêt n'est pas IMP-calculable.
Comment utiliser ce résultats pour démontrer que d'autres problèmes sont indécidables

imp-arrêt entrée : code source P.
question : Est ce que $\phi(\epsilon)$!= $\perp$.

On aimerais montrer que IMP-arrêt0 est lui aussi IMP-indécidable en montrant que IMP-arrêt est plus simple que IMP-arrêt0. On va construire un programme qui utilise IMP-arrêt0 pour résoudre imp-arrêt.

Est ce que ϕ_p(x) != $\perp$

On a montré que IMP-arrêt est indécidable on va dire que IMP-arrêt <=_imp IMP-arrêt0.
On va construire un programme qui utilise un programme qui résout IMP-arrêt0 pour résoudre IMP-arrêt. Dans IMP-arrêt, on a en entrée P et x, on doit manipuler P et x, pour contruire un programme P' qui va nourrir imp-arrêt0.

Démonstration : Par l'absurde, on suppose qu'il existe un programme qui décide imp-arrêt0.

On construit alors le programme suivant :

Read Z ;
P := π_1(Z)
X := π_2(Z)
Q := le programme P où la première ligne [read Y] est remplacée par [read Y ;
Y := X] ;
R := IMP-arrêt0(Q) ;
write(R) ;

Ce programme résout imp-arrêt, qui est imp-indécidable : CONTRADICTION.
Donc IMP-arrêt0 est indécidable.

Cette preuve utilise une technique d'évaluation partielle.

Proposition : Il existe une fonction IMP-calculable s qui à <P,x> associe le code source d'un programme qui ignore son entrée et simule le programme P sur l'entrée x, c'est à dire : $\forall$ y, ϕs(<P,x>)(Y) = ϕp(x).

Proposition : Il existe une fonction IMP-calculable s qui à <P,x> associe le code source d'un programme qui sur une entrée Y, simule le programme P sur l'entrée <X,Y> : ϕs(<P,x>)(Y) = ϕp(<X,Y>). .

Démonstration : Idem, mais remplacer [read Y] par [read Y, Y := <X,Y>].

Prop : il existe un programme IMP qui retourne son propre programme code source.
Démonstration :
Soit R le programme suivante :
Read(Z) ;
X :=π1(Z)
Y := s'(<X,X>)
write Y
donc le programme s'(<R,R>) écrit son propre code source.
Bilan : Ce qui précède ne dépend pas vraiment de IMP.
Quelques principes sous-jacents :

- dénombre de manière calculable les programmes ;
- possibilités d'enchainer, concaténer, composer des programmes.
- existence d'un crochet <.,.> et de projection π1,π2 calculables.
- existence d'un programme universel ;
- existence de fonctions d'évaluation partielle : s,s',...

La calculabilité s'intéresse aux propriétés des fonctions calculables qui sont indépendantes du modèles de calcul considéré

4 Machines de Turing

Les MT sont le modèle de calcul de référence du cours. Il s'agit d'un modèle de machine mécanique abstrait.C'est le modèle de référence en complexité. A.M.Turing : On computable numbers with an application to the Entscheidungproblem.1936

4.1 Définitions

LE ruban est coupé en case, quantité finie d'information, on suppose qu'on a un alphabet fini qui nous permet d'écrire dans les cases du ruban, et le contenu d'un case c'est une lettre sur un alphabet fini.

L'opérateur est une flèche qui pointe sur une case, il a le droit de mémoriser une quantité fini d'informations et est dans un certain état, cet état est lui aussi dans un alphabet fini , l'ensemble Q des états, il a différentes opérations possibles : lire et écrire dans la case pointée, il peut aussi changer d'état, il peux déplacer le pointeur vers la gauche ou la droite. Le programme c'est : quoi faire quand on commence, que faire quand on a un couple état/lettre dans la case.

Définition : Une MT est un octuplet (ensemble d'états(Q),Σ(alphabet d'entrée/sortie),Γ,lettre blanche (B), état initial(Q0), un état fini succès(Qoui), un ensemble d'état finaux échec(Qnon) et δ : la fonction de transition).

Définition : Une configuration d'une MT M est un triplet (q,c,p) $\in$ Q * γ expZ x Z.

Définition : Il existe une transition de la configuration (q,v,p) à la configuration (q',c',p') de M, notée
(q,c,p |-(q',c',p')
siδ(q,c(p)) = (q',a,d)
où $\forall$k$\in$Z c'(k)= a si k = p, c(k) sinon, et p'= p-1 si d = <-, p si d= bas, et p+1 si d= ->

Remarque : Dans une utilisation normale d'une MT le ruban est B presque partout. On préféreras noter uqv une configuration (q,c,p) telle que : u : ce qu'il y a de pas blanc avant la case, q l'état de la MT, et V ce qu'il y a après sur la bande.

Définition : La configuration initiale de M sur l'entrée u $\in \Sigma^*$ est la configuration q0u.

Définition une configuration acceptante de M est une configuration UqouiV.

Définition : une configuration de refus de M ets une configuration UqnonV.

La machine s'arrete lorsqu'elle atteint un état d'arrêt , c'est a dire l'état acceptant qoui ou l'état de refus qnon.

Remarque : On noteras |-n n transitions successive, |-* = union des transitions(|-0 l'identité).

Définition : Le langage L(M) d'une MT m est l'ensemble des mots acceptés : L(M) = u∈ Σ* | q0 u |- * VQuoiW.

Définition : Un langage est récursivement énumérable s'il est le langage d'une MT M, ie L= L(M).

Définition : Un langage L est récursif s'il est le langage d'un MT qui s'arrete sur toute entité.

Exemple : Langage des palindrome : Lu ∈ a,b* | u est un palindrome