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 est 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.

Définitionun problème est décidable si son langage est récursif.
exemple : le problème de l'arret :
l'entrée : une MT M et u un mot u ∈ Σ^*. Question Est que $\Phi_M(U)! = \perp$?

Langage associé : K = wM,u | $\Phi_M(u)! = \perp$

Définition Une fonction partielle f : $\Sigma * -> \Sigma *$ est calculable si il existe une MT M qui calcule F, c'est a dire que $\forall u \in \Sigma *$
- $q_0 \vdash^* vq_{oui}f(u)sif(u)! = \perp$
- $\neg q_0 \vdash^* vq_{oui}f(u)sif(u) = \perp$

Proposition Un langage L ⊆ Σ^* tes récursif si et seulement si L est récursivement énumérable et L est complémentairement récursivement énumérable

Remarque : L est co-r.e signifie $\Sigma^* \backslash L$ est r.e.

Démonstration dans le sens ⇒ L est récursif donc il existe une MT M totale telle que L(M) = L.
L est r.e car M reconnaît L.
L est Co-r.e car M', copie de M où q_{oui} et q_{non} sont échangés, reconnais $\Sigma^* \backslash L$

Dans le sens ⇐ Soient M une MT qui reconnaît ML et N une MT qui reconnaît $\Sigma^* \backslash L$, on construit une MT Z qui , sur une entrée u, simule en parallèle M et N sur l'entrée u. Si M s'arrête alors Z répond oui. Si N s'arrête alors Z répond non. Z est totale et reconnais L.

Exercice : Montrons que L est récursif si et seulement si χ_L est calculable , où $\forall u \in \Sigma^* \chi_L(u)$ =OUI si u ∈ L et NON sinon

5 Thèse de Chruch-Turing-Kleene

- 1936 A.Turing définit le calcul/ par les MT. <- complètement mécanique
- 1936 A.Church définit le calcul par la $\lambda - definissibilité$, i.e le $\lambda - calcul$ <- modèle fonctionnel.

Thèse La notion intuitive de ce qui est calculable correspond exactement à la notion de fonction calculable par MT.

Idée : essayer de construire d'autre modèle de calcul "raisonnables" : la notion de calculables coïncide toujours (en tout cas de 1936 à 2013).

6 la machine universelle

Proposition IL existe une machine de Turing U telle que :
$\forall M \forall u \ \Phi_U(< M, u >) = \Phi_M(u)$

Proposition La fonction STEP suivant est calculable.

$\forall M \forall u \in \Sigma^* \forall k \in N$:
STEP(<M,u,K>) =

- TERMINÉ(q,v) si M sur l'entrée u arrête avant k étapes avec sortie v dans l'état q
- ENCOURS sinon.

Principe de fonctionnement de U Sur l'entrée <M,u> U doit simuler le calcul de M sur l'entrée u. Simuler M c'est lire le ruban , écrire, déplacer la tête , modifier l'état.
U organise sa mémoire, son ruban , en zones, pour coder

- l'état de M
- le ruban de M
- la table de transitions de M

U se contente de chercher l'état et la lettre correcte dans la table des transition, de les remplacer et de déplacer le marqueur.Ajouter des détails techniques pour initialiser et arrêter le calcul.
STEP fonctionne pareil en limitant le nombre de pas de simulation à k.

7 Indécidabilité

Proposition Il existe des langages non récursifs. Preuve par dénombrement : Il y a ω MT et 2^ω fonctions partielles de $\Sigma^* -> \Sigma^*$.

K = <M,u> | $\Phi_M(u)! = \perp$

Remarque : K est r.e il suffit d'utiliser U pour tester $\Phi_M(u)$.

Théorème de l'arrêt K n'est pas récursif, ie le pb de l'arrêt n'est pas décidable.

Démonstration Supposons qu'il existe une MT H telle que
$\forall M \forall u\ \Phi_H(< M, U >) =$
- OUI si $\Phi_M(u) \neq \perp$
- NON si $\Phi_M(u) = \perp$

On diagonalise H en construisant une MT qui s'appelle D(comme diagonale) qui sur l'entrée M (où M est le code d'une machine) :
- exécuter H sur l'entrée <M,M>
- si H à répondu OUI, on boucle
- si H a répondu NON , on s'arrête.

$\forall M, \Phi_D(M) = \perp$ si $\Phi_M(M) \neq \perp$, arrêt si $\Phi_M(M) = \perp$

Donc , si L= D : $\Phi_D(D) = \perp$ si $\Phi_D(D) \neq \perp$, arrêt si $\Phi_D(D) = \perp$: CONTRADICTION!!!
Donc H n'existe pas, K n'est pas récursif, K n'est pas co-r.e.

Théorème de Rice Soit P une propriété non triviale des fonctions partielles f : $\Sigma * -> \Sigma *$.
L'ensemble des machines de Turing qui calculent une fonction qui satisfait P n'est pas récursif. P sélectionne parmis les fonctions partielles un sous-ensemble, non-triviale, indique qu'il existe deux fonction f et g différentes telles que P(f) est vrai et P(g) est faux. M | P(Φ_M) n'est pas récursif.
exemple de propriété :
- l'ensemble des fonctions totales
- l'ensemble des fonctions définies nulle part
- etc...

5 Février 2013

Pour toute propriété non triviale sur les fonctions partielles savoir si une MT calcule une fonction qui satisfait est indécidable : cette fonction est triviale , si il existe une fonction f qui la satisfait, et une fonction g : $f \neq g$ qui ne la satisfait pas. Exemple : { M | P(Φ_M)} n'est pas récursif. { $M|\phi_M \neq \perp$}

$$K = \{ <M,x> \mid \Phi_M(x) \neq \perp \}$$

Démonstration Par l'absurde, supposons qu'il existe une propriété non-triviale P calculable par MT T_P. soit $\perp$ une fonction définie nulle part. Supposons sans perte de généralité, que $\perp \notin$ P et soit S une MT telle que $\Phi_S \in$ P.

On construit une machine de Turing F comme suit :
- Sur l'entrée <M,x>
  - Soit T le codage de la MT suivante :
    - Sur l'entrée y :
      - simuler M sur l'entrée x.
      - si M s'arrête , on simule S sur l'entrée y
      - si S s'arrête, on s'arrête.
  - Simule T_P sur l'entrée T
    - Si T_P répond OUI, accepter <M,x>
    - Si T_P répond NON, rejeter <M,x>

F est totale.

$\Phi_F(< M, x >) =$ { OUI si $T_P(T) =$ OUI, ie $si \Phi_M(x) \neq \perp$
et NON si $T_P(T)$, ie si $\Phi_M(x) = \perp$}
F résout le problème de l'arrêt ! CONTRADICTION

Les problèmes indécidables sont légion :
- le typage au second ordre du λ-calcul ;
- mortalité des produits de matrices carrées
- problème du mot pour les groupes finiment engendrés.
- exemple : existence d'une solution entière à une équation polynomiale à coefficient entiers ($10^{ème}$ problème de Hilbert).
- Pavabilité du plan par un jeu de tuiles de Wang (tuiles divisées en quater avec des couleurs au coins, pas le droit de les tourner , et on les collent avec deux couleurs identiques aux arrêtes adjacentes) ;
  - entrée : famille de tuiles de Wang T $\subseteq \Sigma^4$
  - Question : T pave-t-il le plan , ie, $\exists c : Z^2 -> T$ telle que les contraintes soient satisfaites.

Problème de correspondance de Post
- entrée : u_1, u_2 , ..., u_n, v_1, v_2,...,$v_n \in \Sigma^*$
- question : $\exists (ik)$ u_1, u_2 , ..., $u_n = v_1$, v_2,...,v_n ?

8 Interlude

Définition Une machine linéairement bornée (MTLB) est une MT qui s'interdit d'écrire sur B. $\forall q, q', a, d$ $\sigma(q, B) = (q', a, d)$ -> a = B car on ne peux pas écrire sur blanc.

Proposition L'arret des MTLB est décidable.

Démonstration On construit le graphe fini des configuration accessibles à partir de l'entrée x pour la machine M. Il y a environ $|Q| * (|x| + 2) * |\Gamma|^{|x|}$

$\exists_{MTLB}$
- entrée : M une MTLB
- question : Existe-t-il une entrée x acceptée par M ?

Proposition $\exists_{MTLB}$ est indécidable
Démonstration Par l'absurde, on suppose que $\exists_{MTLB}$ est décidable, et on construit un programme qui résoud l'arrêt des MT.
IL s'agit étant donnés une MT M et une donnée x, de calculer une MTLB f(<M,x>) qui accepte une entrée si et seulement si le MT M s'arrête sur l'entrée x. Notre MTLB est un vérifieur/parseur/accepteur de trace de MT : il considère son entrée comme une suite de configuration valide de M qui commence pas q_0^x et termine par une configuration d'arrêt :
BB#$q_0x1x2...xn$# ... # ...#BB

$$\Sigma' = Q_M \cup \Gamma_M \cup \{\#\}$$
$$\Gamma' = \Sigma' \cup \{\text{B}\} \cup \text{alphabet Entouré}$$

Principe de fonctionnement de la MTLB :
- vérifier que l'entrée commence par #$q_0x1x2...xn$#
- vérifier que l'entrée termine par # ... $q_{OUI}...$#
- pour chaque paire successive #u#v# avec $u, v \in \Sigma'^* \setminus \{\#\}7$
  - vérifier que $u| -_M v$ en faisant des aller retours en marquant les position avec l'alphabet entouré.

MPCP est la variante de PCP dnas laquelle on est obligé de commencer par le premier domino ($i_1 = 1$).
Théorème MPCP est indécidable

Démonstration Idée : comme pour $\exists_{MTLB}$ on va construire un reconnaisseur de trace de MT. <M,x> |—-> jeu de dominos T(M,x) tel que T(M,x) à une solution si et seulement si $\Phi(x) \neq \perp$ et même mieux la solution de T(<M,x>) écrit une trace de M sur l'entrée x.

Voici les dominos de T(<M,x>) INSERER ICI LES IMAGES DU TABLEAU T(<M,x>) admet pour solutions les traces acceptables de M sur l'entrée x. MPCP est indécidable

9 Evaluation

2 CC
- 1h30 calculabilite mer. 6 Mars
  - construire une MT

    - expliquer une MT
    - démontrer qu'un problème est décidable.
  - ensuite une correction
  - 1h30 de complexité
    - raisonner sur les classes
    - démontrer qu'un problème est NP-complet
  - ensuite une correction

et pour finir une CT de 2h sur tout

10 Corollaire

PCP est indécidable , c'est une sorte de dominos qui se collent les uns des autres.

Démonstration On va transformer récursivement une instance donnée de MPCP τ en une instance τ' de PCP de sorte que τ possède une solution si et seulement si τ' en possède une.
Dans MPCP, on est obligé de poser en premier le premier domino de la liste. Il existe donc des solution de PCP qui ne sont pas solution de MPCP.
$[\frac{aa}{a}]\ [\frac{bb}{ab}]\ [\frac{b}{bb}]\ [\frac{bb}{b}]$

$[\frac{aa}{a}]\ [\frac{bb}{ab}]\ [\frac{b}{bb}]$ est une solution MPCP

$[\frac{b}{bb}]\ [\frac{bb}{b}]$ solution de PCP , mais pas MPCP

idée : $[\frac{*a*a}{*a*}]\ [\frac{*b*b}{a*b*}]\ [\frac{*b}{b*b*}]\ [\frac{*b*b}{b*}]\ [\frac{*\$}{\$}]$ comme ça on est obligé de prendre la première tuile en premier et celle avec des dollars en dernier.

$\forall u \in \Sigma*$
- $\bullet u = *u1*u2...*un$
- $u\bullet = u1*u2*...un*$
- $\bullet u\bullet = *u\bullet = \bullet u*$

À partie de $\tau = (u1, v1), ..., (un, vn)$
On construit τ' qui contient les $n+2$ dominos :
- $(\bullet u_1, \bullet v_1\bullet)$,
- $\forall i \in 1, .., n(\bullet u_i, v_i\bullet)$,
- $(\bullet\$, \$)$

avec \$ et * des nouvelles lettres

Une solution PCP de τ' commence nécessairement par $(\bullet u_1, \bullet v_1\bullet)$ et se termine par $(\bullet\$, \$)$.
une plus courte solution contient entre ces tuiles uniquement $(\bullet u_i, v_i\bullet)$
Les lettres en position impaires sont des * , en position paires des lettres de u_i/v_i
τ possède une solution de MPCP si τ possède une solution de PCP.
La transformation de τ en τ' est totale et calculable donc PCP est indécidable.

Réductions pour montrer qu'un problème est indécidable on a souvent construit une transformation calculable totale d'une instance A en instance B qui préserve les solutions.

Définition Soient A et B deux problèmes/langages $A, B \subseteq \Sigma*$, on notera A $<=_m$B si il existe une fonction F : $\Sigma * -> \Sigma*$ calculable et totale telle que : $\forall x \in \Sigma* \; x \in A <-> f(x) \in B$ On dit que A est plus simple que B , que B est plus compliqué que A ou A est réductible à B.

Proposition $<=_m$ est un pré-ordre

Démonstration

- $<=_m$ est réflexive : $\forall A \subseteq \Sigma* \; A <=_m A$, en effet l'identité (a vers a) est calculable et totale.
- $<=_m$ est transitive : $\forall A, B, C \subseteq \Sigma* \; A <=_m B$ et $B <=_m C -> A <=_m C$
  - si $A <=_m B$ il existe F calc.tot. tq $\forall x \; \forall x \in \Sigma* \; x \in A <-> f(x) \in B$
  - si $B <=_m C$ il existe G calc.tot. tq $\forall x \; \forall x \in \Sigma* \; x \in B <-> f(x) \in C$
  - Donc $\forall x \in \Sigma* \; x \in A <-> f(x) \in B <-> f(x) \in C$.
    G o F qui est calculable totale réduit A à C.

EX : $ARRET <=_m MPCP <=_m PCP$ donc $ARRET <=_m PCP$

Proposition si $A <=_m B$ et A indécidable, alors B est indécidable

Démonstration Soient $A <=_m B$ avec A indécidable.
supposons B décidable. Alors il existe une MT μ totale qui reconnaît B.
Comme $A <=_m B$ il existe F calculable telle que $\forall x \in \Sigma* \; x \in A <-> f(x) \in B$
alors il existe ne MT F tot qui calcule f.
À partir de F et μ on construit une MT α qui reconnaît A :

- sur l'entrée x
- α applique F pour obtenir f(x)
- puis α appelle μ sur l'entrée f(x)

Par construction $\forall x \in \Sigma* \; x \in A <-> \Phi_A(x) = OUI$
α est totale, on en déduit que A est décidable, CONTRADICTION ! ! Donc B est indécidable.

Propostion si $A <=_m B$ alors $!A <=_m !B$.

Idée $\forall x \in \Sigma* \; x \in A <-> f(x) \in B$
donc $\forall x \notin \Sigma* \; x \in A <-> f(x) \notin B$
$\forall x \in \Sigma* \; x \in !A <-> f(x) \in !B$

Proposition si $!A <=_m !B$ et B est rec.enum, alors a est rec.enum

Proposition si $!A <=_m !B$ et A n'est pas rec.enum alors B n'est pas rec.enum
ARRET rec.enum et pas co.rec.enum.
!ARRET co.rec.enum et non-rec.enum

EQTM : égalité entre machines de Turing
- Entrée : <M1,M2> paire de MT
- question : Est ce que L(M1) = L(M2) ?

Théorème EQTM n'est ni rec.Enum , si co-rec.enum.

Proposition si $A <=_m B$ et A non rec.enum alors b non rec.enum

Corollaire si $ARRET <=_m !A$ alors A est non rec.enum.
Démonstration On sais que !ARRET n'est pas rec.enum , et si $!ARRET <=_m A$ alors a non rec.enum , on a $ARRET <=_m !A$ ssi $!ARRET <=_m A$

Démonstration du théorème
- montrons que $ARRET <=_m !EQTM$, i.e EQTM non rec.enum.
  À <M,x> on associe récursivement la paire
  - M1 : le MT qui calcule $\perp$
  - M2 :
    - sur l'entrée y
    - simuler M sur x
- Montrons que $ARRET <=_m EQTM$, i.e !EQTM non rec.enum.
  À <M,x> on associe récursivement la paire
  - M1 :la MT qui s'arrête tout le temps et répond CAKE.
  - M2 :
    - sur l'entrée y
    - simuler M sur x
    - répondre CAKE.