Calculabilité et Décidabilité

claire.lefevre@univ-angers.fr

- Notions de fonctions calculables,
  de problèmes décidables / indécidables
- Fonction calculable = il existe une méthode qui permet d'en calculer toutes les valeurs (il existe une méthode effective, un algorithme)
- Ce sont des notions essentielles en mathématiques, et dont les applications en informatique sont tout aussi essentielles

Petite Histoire

- Les mathématiciens ont toujours utilisé des méthodes de calcul sans que la notion ne soit définie précisément
- Ils cherchaient des méthodes générales pour résoudre des problèmes mathématiques
  ex : validité d'une formule du 1er ordre
- Ils en sont venus à formuler, formaliser cette notion de méthode de calcul,
  procédure effective,
  algorithme

Petite Histoire (suite)

- Entre 1931 et 1936, divers définitions et résultats apparaissent (Church, Kleene, Turing, Gödel)
- Plusieurs définitions de la notion d'algorithme, et de ce qu'on allait appeler les fonctions calculables :
  - Logique de 1er ordre avec les entiers (déclaratif)
  - Fonctions récursives partielles (+ proche de prg° fctionnelle)
  - Machines de Turing (on modélise l'outil de calcul lui-même)
- Toutes les définitions se sont révélées équivalentes (elles décrivent les mêmes fonctions)
- On utilise maintenant la plus « pratique », celle de Turing (1936)

Petite Histoire (suite)

- Cela a permis d'établir que de nombreux problèmes n'étaient pas « traitables »,
  i.e., non seulement on ne connaît pas de méthode pour les résoudre,
  mais il ne pourra jamais exister de telle méthode
- Ce sont les problèmes dits indécidables : on a démontré qu'il ne peut pas exister d'algorithme, de méthode effective de calcul, pour les résoudre
- Les conséquences en informatique sont très importantes : si on a établi qu'un problème est indécidable, on ne pourra jamais écrire de programme capable de le résoudre dans tous les cas

Problème ≠ Instance de problème

- Exemple de problème :
  soient deux entiers n et m, n est-il multiple de m ?
  On connaît une méthode générale (diviser n par m, et regarder si le reste = 0) qui permet de résoudre le problème dans tous les cas.
  Le problème est donc décidable.
- Instance du problème :
  10 est-il multiple de 3 ?

⇒ Un problème représente une infinité d'instances

Décidable = il existe une méthode générale pour résoudre **toutes** les instances

Indécidable = il n'existe pas de telle méthode générale mais il peut exister des méthodes pour résoudre certaines instances

Quand on parle de décidabilité / calculabilité,
il s'agit de savoir s'il existe ou non un algorithme pour un problème

Mais on ne parle pas de l'efficacité de cet algorithme (domaine, lié, de la complexité des algorithmes, non abordé ici)

Vue informelle de l'indécidabilité :
les pbs que les ordinateurs ne résoudront jamais

Ex de problème = déterminer si un programme va ou non afficher « coucou »

- Idée : on fait tourner le prg et on attend le résultat
- Pb : le prg peut mettre un temps extrêmement long avant d'afficher un résultat,
  il peut aussi ne jamais rien afficher (« boucle »)

=> Ne pas savoir **quand** quelque chose arrive est la cause de notre incapacité à (pré)dire ce qu'un prg fait

```
int main() {
        cout << "coucou" <<endl;          (*1)
}
```



```
int main()
{ int n, total, x, y, z;                                    (*2)
  cin >> n;
  total = 3;
  while(1)
  {       for (x=1; x<=total-2; ++x)
              for (y=1; y<=total-x-1; ++y)
              {     z = total-x-y;
                    if (exp(x,n)+exp(y,n) == exp(z,n))
                            cout << "coucou" << endl;
              }
          ++total;
  }
}
```

Recherche des solutions entières de l'équation $x^n+y^n=z^n$ (dernier théorème de Fermat)

- On prend en entrée un entier n
- On cherche des valeurs pour x, y et z t.q. $x^n+y^n=z^n$
- Si on trouve, on affiche « coucou »
  sinon, on continue indéfiniment à chercher sans rien afficher

Aujourd'hui, on sait que :

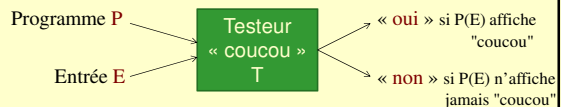
- Si n=2, il y a des solutions (ex: total=12, x=3, y=4, z=5) , donc le prg affiche « coucou »
- Mais ∀n>2, il n'y a pas de solution, donc le prg ne s'arrêtera jamais

Fermat avait émis l'hypothèse il y a 300 ans, mais la preuve n'a été établie que récemment

⇒ Il a fallu 300 ans aux mathématiciens pour déterminer si le prg (*2) affichait ou non « coucou »

⇒ On peut donc imaginer qu'il est difficile de déterminer ce que va faire un prg quelconque (avec 1entrée qcq)

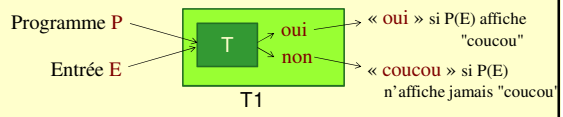
Le testeur « coucou »



Question : est-ce qu'un testeur « coucou » existe ?

Preuve par l'absurde :

- On suppose que le testeur T existe
- On le transforme, en modifiant l'affichage

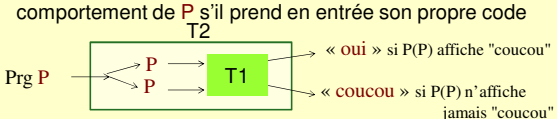


si T existe, le nouveau testeur T1 existe aussi

Le testeur « coucou » (suite)

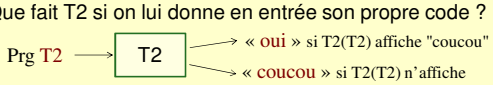
- On transforme T1, en modifiant l'entrée :

On ne prend en entrée que P (pas E), et on s'intéresse au comportement de P s'il prend en entrée son propre code



si T1 existe, le nouveau testeur T2 existe aussi

- Que fait T2 si on lui donne en entrée son propre code ?



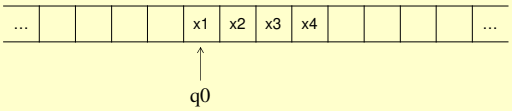
Absurde. Donc T2 n'existe pas, ni T1, ni T

=> Un testeur « coucou » ne peut pas exister

Les machines de Turing
(Alan Turing, 1936)

- Une formalisation de la notion de procédure effective
- « machine » très simple et très puissante

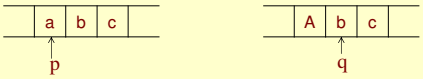
- Une machine de Turing (MT) déterministe :
  - Une mémoire infinie = un ruban divisé en cases, chaque case contient un symbole
  - Une tête de lecture qui se déplace sur le ruban
  - Un ensemble d'états



fonctionnement

Initialement,
- On écrit sur le ruban le mot d'entrée (les données) toutes les autres cases contiennent des blancs (#)
- La tête de lecture est sur le 1er symbole de l'entrée
- La machine est dans son état initial

À chaque étape, la machine
- Lit le symbole courant
- Remplace ce symbole par un autre
- Déplace la tête de lecture d'1 case à dr. ou à gauche
- Change d'état



Un mot est accepté si la MT atteint un état acceptant

Définition formelle

Une MT est un 7-uplet $M = (\Sigma, \Gamma, B, Q, q_0, F, \delta)$ où
- Σ : ens. fini de symboles d'entrée (alphabet d'entrée)
- Γ : ens. fini des symboles de ruban, avec $\Sigma \subseteq \Gamma$
- B : symbole blanc, $B \in \Gamma \setminus \Sigma$
- Q : ensemble fini d'états
- q_0 : état initial, $q_0 \in Q$
- F : ensemble des états acceptants, $F \subseteq Q$
- δ : fonction de transition, $Q \times \Gamma \rightarrow Q \times \Gamma \times \{\rightarrow, \leftarrow\}$

$\delta(p, a) = (q, A, \leftarrow)$ signifie

« si on est en état p et que le symbole courant est a,
alors, - on passe en l'état q
- on remplace a par A
- on déplace la tête de lecture à gauche »

exemple

- $M = (\Sigma, \Gamma, B, Q, q_0, F, \delta)$ avec
  $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \#\}$, $B = \#$,
  $Q = \{q, f\}$, $q_0 = q$, $F = \{f\}$
  $\delta(q, 0) = (q, 0, \rightarrow)$
  $\delta(q, 1) = (f, 0, \rightarrow)$

Les autres valeurs de δ ne sont pas définies

- C'est une MT qui
  - Se déplace vers la droite jusqu'à trouver le premier 1
  - Remplace ce 1 par 0, et s'arrête en état final

  S'il n'y a pas de 1, la machine s'arrête en état q (non final)

Configuration d'une MT

Une configuration est donnée par :
- L'état courant
- La partie de la chaîne à gauche de la tête de lecture
- La partie de la chaîne, depuis la tête de lecture, jusqu'à la fin (hors blancs)

Exemple (suite du précédent)

q001101	ou encore	q **0**01101
|- 0q01101		q 0**0**1101
|- 00q1101		q 00**1**101
|- 000f101		f 000**1**01

Chaînes et langages acceptés par une MT

- Une chaîne ω est acceptée par une MT M si
  $q_0\, w \mid\!-_M^*\, \alpha\, q\, \beta$ avec $q \in F$

i.e., *partant de sa configuration initiale, la machine atteint (un jour) une configuration dont l'état est acceptant*

- Le langage accepté par une MT M est l'ensemble des chaines acceptées par M

  $L(M) = \{w \in \Sigma^* \mid M \text{ accepte } w\}$

exemple

Une MT qui accepte le langage $\{a^n b^n \mid n > 0\}$

M = ({a,b}, {a,b,X,Y,#}, #, {q_0,q_1,q_2,q_3,q_4}, q_0, {q_4}, δ)

Principe (voir au tableau pour δ) :

(q_0) on remplace le 1er a par X

(q_1) on avance jusqu'à trouver le 1er b (sinon échec)
et on le remplace par Y

(q_2) on recule jusqu'à trouver un X (donc le a suivant)
et on retourne en q_0

(q_0) s'il n'y a plus de a

(q_3) on « passe » tous les Y jusqu'à trouver un blanc
(succès, même nbr de a et b) sinon échec

21

Langage accepté / décidé

On exécute une MT M pour un mot w, 3 cas possibles :

- On passe par un état acceptant, w est accepté
- M s'arrête (config. où δ non définie) sans être passée par un état acceptant, w est refusé
- M ne s'arrête jamais, et ne passe jamais par un état acceptant (exécution infinie, « bouclage ») , w devrait être refusé

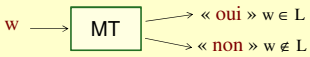
Pb : Si une exécution ne s'arrête pas, on ne sait pas si elle ne s'arrêtera jamais
ou si l'on n'a simplement pas attendu assez longtemps

=> Une MT qui a des exécutions infinies ne nous fournit pas une définition de procédure effective, d'algorithme

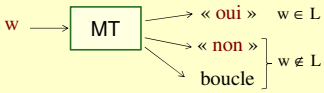
22

Langage accepté / décidé

- Un langage est décidé par une MT M si :
  - M accepte L
  - M n'a pas d'exécution infinie
- Un langage est récursif (décidable) s'il est *décidé* par une MT



- Un langage est récursivement énumérable (semi-décidable) s'il est *accepté* par une MT



23

Thèse de Church (ou Turing/Church)

Les MT (sans exécution infinie) formalisent bien la notion (floue) de méthode de calcul, d'algorithme, de procédure effective

Ou, plus précisément,

Les langages reconnaissables par une procédure effective sont exactement ceux décidés par une MT

Rem : il n'y a pas de démonstration possible, slmt des arguments :

- Toute extension des MT n'ajoute pas d'expressivité
- On peut simuler un ordinateur avec une MT
- Toute autre formalisation s'est révélée équivalente

24

Les fonctions

Une MT peut calculer une fonction f :

- on donne comme mot d'entrée l'argument w de la fct
- qd la machine s'arrête, f(w) est inscrit sur le ruban

On dit qu'une fonction est calculable par une MT si, pour tout mot d'entrée w, la MT s'arrête toujours dans une configuration où f(w) est sur le ruban.

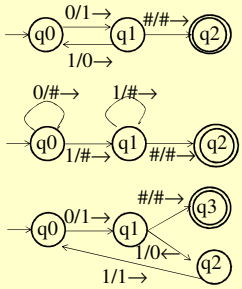
On peut ré-exprimer la thèse de Turing/Church :

Les fonctions calculables par une procédure effective sont les fonctions calculables par une MT

25

exercices

- Dire quel est le langage accepté par ces MT et ce que contient le ruban à la fin :



- Donner une MT qui, étant donné un entier n en binaire, calcule n+1

26

Exemple : soustraction

Fonction n,m → n-m

On code les entiers par des suites de 1

Entrée : 1^n01^m

Sortie : 1^{n-m} si n > m 01^{m-n} si n ≤ m

Principe (voir la MT au tableau) :

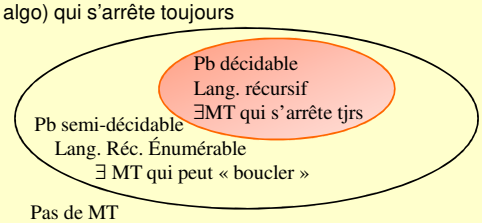
(q_0) on supprime le 1er 1 (remplacé par #)
(q_1) on avance jusqu'à la fin de l'entrée (le # à droite)
(q_2) si le dernier car est un 1, on le supprime
et (q_3) on recule jusqu'au début de l'entrée (# g)
et on retourne en q_0
si le dernier car est un 0 (m=0),
on le remplace par 1 (succès q_4)
(q_0) s'il n'y a plus de 1 (n=0, cas où n ≤ m), succès q_4

Indécidabilité

MT permettent de modéliser ce qu'est un « outil de calcul » indépendamment des limites des ordinateurs

=> S'il n'existe pas de MT pour un pb, alors aucun ordinateur du futur ne résoudra jamais ce pb

Rappel : Un problème P est décidable s'il existe une MT (un algo) qui s'arrête toujours



- Les langages indécidables incluent :
  - Les langages non r.e. (il n'existe pas de MT)
  - Les langages r.e. qui ne sont pas récursifs (MT avec exécutions infinies)

Ex : le problème de l'arrêt

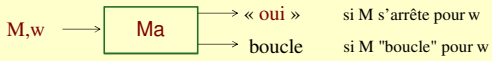
Intuition : existe-t-il un prg qui prend en entrée (P,E) et détermine si P s'arrête pour l'entrée E ?

Le problème de l'arrêt (Halting Problem)

La = { (M,w) | la MT M s'arrête pour l'entrée w }

- La est récursivement énumérable (semi-décidable : il existe une MT qui accepte La) :

(admis) on peut construire une MT Ma qui prend en entrée M,w,
simule le comportemt de la machine M avec l'entrée w,
et répond "oui" si M s'arrête pour w, et boucle sinon

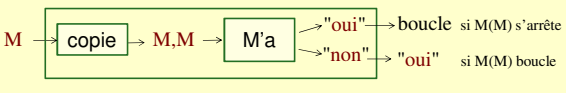


La n'est pas récursif (pas décidable)

(il n'existe pas de MT qui s'arrête toujours pour La)

Idée de la preuve

- Supposons que La est décidable (récursif)
  alors, il existe une MT qui s'arrête toujours pour La
- Soit M'a cette MT
- On peut construire une MT M_B



- si M'a existe, M_B existe (on peut la construire formellement)
- Si on donne en entrée à M_B son propre codage :
  - $M_B(M_B)$ boucle si $M_B(M_B)$ s'arrête
  - $M_B(M_B)$ s'arrête (répond « oui ») si $M_B(M_B)$ boucle

absurde, donc M_B n'existe pas, et M'a non plus

Le problème de l'arrêt est donc indécidable

Le théorème de Rice

(problèmes indécidables concernant les programmes)

Toute propriété non triviale qui porte sur la fonction calculée par une MT est indécidable

(non triviale = ni toujours vraie, ni toujours fausse)

Exemples : savoir si le langage accepté par une MT

- est vide
- est le langage « plein » Σ*
- est fini
- est régulier / non contextuel / récursif / r.e. / indécidable

Même les questions qui paraissent les plus simples, par exemple savoir si une MT - accepte l'ens. des mots de longueur paire
- calcule la fct n → n+1
sont indécidables

Conséquence importante

Tout problème non trivial concernant les programmes est indécidable

Exemples

- Équivalence de programmes

Étant donnés 2 programmes, calculent-ils la même chose ?

- Morceau de code inutile

Étant donné un programme, contient-il un morceau de code qui ne sera jamais exécuté ?

- Utilisation d'une ressource

Étant donné un programme, va-t-il utiliser l'imprimante (ou le capteur, ou …)

42

beaucoup d'autres problèmes indécidables

- Validité dans le calcul des prédicats

Déterminer si une formule logique de 1er ordre est valide

- Grammaires non contextuelles
  - L(G) = Σ^* ? (universalité du langage)
  - L(G1) ∩ L(G2) = ∅ ?
  - G est ambigüe ?
- 10ème problème de Hilbert

Déterminer si une équation p(x1, …, xn) = 0, où p(x1, …, xn) est un polynôme à coefficients entiers, possède des solutions entières

Ex : $x^2 - 4 = 0$ a une solution entière, $x^2 - 2 = 0$ n'en a pas

- Etc …

43

Pour conclure

Si un problème est indécidable,

- il n'y a pas de solution algorithmique totalement générale (qui fonctionne pour toute instance du problème)
- mais il peut y avoir des programmes qui traitent certaines instances, ou qui fonctionnent sans garantie de résultat

Exemple : démonstration de théorème (indécidable)

Il existe de nombreux démonstrateurs qui rendent bien des services même s'ils ne fonctionnent pas dans tous les cas

44