5. Incalculabilité *explicite*

Bruno Grenet

Université Grenoble Alpes – IM²AG
L3 Informatique
UE Modèles de calcul – Machines de Turing

https://membres-ljk.imag.fr/Bruno.Grenet/MCAL-MT.html

Introduction

Rappel

- L'ensemble des algorithmes est dénombrable
- L'ensemble des fonctions est indénombrable

$\Rightarrow$ il existe des fonctions non calculables

Certes, mais où ?

- La preuve est *non constructive* : ne fournit aucune fonction non calculable
- Une construction possible : $f : \Sigma^* \to \{0,1\}$ définie *aléatoirement*
  - pour tout $w \in \Sigma^*$, on tire uniformément la valeur $f(w) \in \{0,1\}$
  - *on peut montrer que* f est incalculable avec probabilité 1

Objectifs

- Construire des fonctions *explicites* non calculables
- Réfléchir sur la distinction *décidable* / *reconnaissable*

Algorithmes, fonctions, langages, problèmes

$\Sigma = \{0,1\}$

Vocabulaire (rappel)

- Fonction (partielle) ou problème $f : \Sigma^* \to \Sigma^*$
- Langage : $L \subset \Sigma^*$
- Algorithme :
  - soit définition informelle et intuitive
  - soit objet dans une formalisation possible — machine de Turing, λ-calcul, …

Problèmes de décision et langages

- Problème de décision : fonction *totale* $f : \Sigma^* \to \{0,1\}$
- Équivalent à un langage : $f \leftrightarrow L_f = \{w : f(w) = 1\}$

Dans la suite

- On ne s'intéresse qu'à des *problèmes de décision*/*langages*
  - On utilise les deux formalismes en fonction de ce qui est le plus pratique
  - Machines de Turing avec un état « oui »

Table des matières

Programmes et données

Un programme est une donnée comme un autre !

- Programme dans un langage donné : fichier texte
- Avec un interpréteur : exécution du programme contenu dans le fichier
- Mais aussi : programmes qui modifient d'autres programmes

Machines de Turing

- Rappel : codage $\langle \mathcal{M} \rangle \in \{0,1\}^*$ d'une machine
- Une machine $\mathcal{N}$ peut prendre en entrée le code d'une machine $\langle \mathcal{M} \rangle$

Autres formalisations

- Exactement pareil avec des machines RAM, des automates cellulaires, …
- Dans toute formalisation d'algorithmes, un algorithme peut prendre en entrée la description d'un algorithme !

Machine universelle

Définition

Une machine de Turing $\mathcal{U}$ est dite **universelle** si sur l'entrée $\langle \mathcal{M}, x \rangle$, $\mathcal{U}$ *simule le comportement* de $\mathcal{M}$ sur l'entrée x :

- $\mathcal{U}$ termine sur l'entrée $\langle \mathcal{M}, x \rangle$ si et seulement si $\mathcal{M}$ termine sur l'entrée x
- Si $\mathcal{U}$ termine, elle renvoie la même valeur que $\mathcal{M}$

Théorème fondamental

Il existe une machine de Turing universelle

Idée de la preuve

- Rubans : 1. $\langle T_b \rangle$, 2. $\langle x \rangle$, 3. état de T_b, 4. ruban de T_b
- Simuler une transition $\delta(q, a) = (q', b, \leftrightarrow)$
  - -> Aller chercher (q, a) sur le ruban 1
  - -> Ecrire b sur le ruban 4 et déplacer la tête, remplacer q par q' sur le ruban 3.

Utilités théorique et pratique

Théorie

- Dans tout modèle Turing-complet, il existe un algorithme universel
  - Un algorithme universel permet de calculer toute fonction calculable
  - Un algorithme universel peut donc être dit Turing-complet
  - Exemple : le *jeu de la vie* est Turing-complet
- Un modèle est Turing-complet s'il peut simuler une machine universelle

- Une machine $\mathcal{N}$ peut prendre $\langle \mathcal{M} \rangle$ en entrée *et simuler* $\langle \mathcal{M} \rangle$ *sur l'entrée de son choix*

Pratique

- Tout langage de programmation admet un *programme universel*
  - interpréteur / compilateur
  - écrit dans le langage lui-même
- En quel langage est écrit GCC ?

Dans la suite de ce cours

Algorithme = machine de Turing (ou autre…)

- Formalisation implicite
- Description des algorithmes en *pseudo-code*
  - traduction possible en machine de Turing ou autre formalisation
  - pseudo-code extrêmement simple → pas de triche !
- Pour un algorithme A, $\langle A \rangle$ est sa description binaire

Conséquences de la machine universelle

- Étant donné $\langle A \rangle$ et w, un algorithme peut simuler $A(w)$
- Étant donné $\langle A \rangle$, un algorithme peut calculer une variante $\langle A' \rangle$ de $\langle A \rangle$
- Exemples :
  - si $A(w) = 1$, renvoyer 0 ; si $A(w) = 0$, rentrer dans une boucle infinie
  - étant donné $\langle A \rangle$ et w, calculer $\langle A_w \rangle$ qui ignore son entrée et exécute $A(w)$

→ Difficultés (éventuellement) techniques mais pas conceptuelles

Table des matières

Définition

Problème de l'arrêt (informel)

Entrées : un algorithme A
une entrée w
Sortie : 1 si A termine sur l'entrée w
0 si A ne termine pas sur l'entrée w

Formalisations

- Fonction H : $\{0,1\}^* \to \{0,1\}$

$$\langle A, w\rangle \mapsto \begin{cases} 1 & \text{si } A \text{ termine sur l'entrée } w \\ 0 & \text{sinon} \end{cases}$$

- Langage $K = \{\langle A, w\rangle : A \text{ termine sur l'entrée } w\} \subset \{0,1\}^*$

Incalculabilité

Théorème de l'arrêt

Le problème de l'arrêt n'est pas calculable

Preuve

- On suppose qu'il existe un algo H tq $H(\langle A, \omega \rangle) = \begin{cases} 1 \text{ si } A(\omega)\downarrow \\ 0 \text{ si } A(\omega)\uparrow \end{cases}$
- On construit D : sur l'entrée $\langle A \rangle$,
  1. Simule $H(\langle A, A \rangle)$
     - ↳ si $H(\langle A, A \rangle) = 1$, rentrer dans une boucle infinie
       Sinon renvoyer 1
- Que fait $D(\langle D \rangle)$?
  - → Si $H(\langle D, D \rangle) = 1$, $D(\langle D \rangle)\uparrow$: impossible
  - → Si $H(\langle D, D \rangle) = 0$, $D(\langle D \rangle)\downarrow$: impossible

D'autres problèmes incalculables

Corollaire

Les langages suivants sont indécidables :

- $\{\langle A \rangle : A \text{ termine sur l'entrée vide}\}$ (*)
- $\{\langle A \rangle : A \text{ termine sur au moins une entrée}\}$
- $\{\langle A \rangle : A \text{ termine sur toute entrée}\}$
- $\{\langle A \rangle : A \text{ renvoie toujours 1}\}$
- ...

versions fonction

est-ce que $A(\varepsilon)\downarrow$?
est-ce qu'il existe x t.q. $A(x)\downarrow$?
est-ce que $A(x)\downarrow$ pour tout x ?
est-ce que $A(x) = 1$ pour tout x ?

Preuve du premier

- Supp. il existe un algo T qui décide (*).
- On va utiliser T pour résoudre le pb de l'arrêt.
- On construit H : sur l'entrée $\langle A, w \rangle$
  1. Calculer $\langle A_w \rangle$
  2. Renvoyer $T(\langle A_w \rangle)$

$$H(\langle A, w \rangle) = T(\langle A_w \rangle) = \begin{cases} 1 \text{ si } A_w()\downarrow \Leftrightarrow A(w)\downarrow \\ 0 \text{ sinon} \end{cases}$$

Généralisation : théorème de Rice

Définition

- *Propriété non triviale* des langages reconnaissables :
  - sous-ensemble $\mathcal{P}$ des langages reconnaissables
  - $\mathcal{P}$ n'est ni vide, ni l'ensemble de tous les langages reconnaissables
  - (Intuitivement, $\mathcal{P}$ est l'ensemble des langages reconnaissables *qui satisfont la propriété*)

Exemple

- « Le langage ne contient que des palindromes » :
  - $\mathcal{P} = \{L \in \{0,1\}^* : \forall w \in L, w = \overleftarrow{w}\}$ où $\overleftarrow{w}$ est le miroir de w
  - $\mathcal{P}$ est non triviale car $L = \{w : w = \overleftarrow{w}\} \in \mathcal{P}$ mais $\{0,1\}^* \notin \mathcal{P}$

Théorème de Rice

Pour toute propriété non triviale $\mathcal{P}$ des langages reconnaissables, $\{\langle A \rangle : L(A) \in \mathcal{P}\}$ est indécidable

Conséquences

- Aucun algorithme ne peut décider si $L(A)$ ne contient que des palindromes
- Tous les langages de la page précédente sont indécidables !

Preuve du théorème de Rice

On suppose qu'il existe R tq $R(\langle A \rangle) = \begin{cases} 1 & \text{si } L(A) \in P \\ 0 & \text{sinon} \end{cases}$

Cas 1 . On suppose $\emptyset \notin P$, et on choisit $A^+ \in P$

. Pour A et ω, on définit $A_\omega(y)$: $\begin{cases} 1. \text{ Simule } A(\omega) \\ 2. \text{ Si } A(\omega) \downarrow, \text{ simule } A^+(y) \end{cases}$

Alors $L(A_\omega) = \begin{cases} L(A^+) \in P & \text{si } A(\omega) \downarrow \\ \emptyset \notin P & \text{sinon} \end{cases}$

. On construit H : sur l'entrée $\langle A, \omega \rangle$,
1. Construit $\langle A_\omega \rangle$
2. Appelle R sur $\langle A_\omega \rangle$

$H(\langle A, \omega \rangle) = R(\langle A_\omega \rangle) = \begin{cases} 1 \text{ si } L(A_\omega) \in P \\ 0 \text{ sinon} \end{cases}$
$= \begin{cases} 1 \text{ si } A(\omega) \downarrow \\ 0 \text{ sinon.} \end{cases}$

Contradiction car H ne peut pas exister.

Preuve du théorème de Rice

Cas 2 $\emptyset \in P$. On choisit $A^- \notin P$.

- $A_\omega(y)$: 1. Simule $A(\omega)$
  2. si $A(\omega)\downarrow$, simule $A^-(y)$

- $H(\langle A, \omega \rangle) =$ 1. Calcule $\langle A_\omega \rangle$
  2. Simule $R(\langle A_\omega \rangle)$
  3. Renvoie le contraire de R

$$H(\langle A,\omega\rangle) = 1 - R(\langle A_\omega\rangle) = \begin{cases} 0 & \text{si } L(A_\omega) \in P \\ 1 & \text{sinon} \end{cases} = \begin{cases} 0 & \text{si } A(\omega)\uparrow \\ 1 & \text{si } A(\omega)\downarrow \end{cases}$$ Contradiction.

Bilan

Théorèmes de l'arrêt et de Rice

- Aucun algorithme ne peut décider si une machine de Turing s'arrête sur une entrée w
- Aucune propriété sur le langage reconnu par une machine de Turing n'est décidable
  - Implique : aucune propriété sur la fonction calculée n'est décidable

Attention !

- On *sait* décider des propriétés sur la machine de Turing elle-même !
  - « la machine possède 12 états »
  - « sur l'entrée w, la machine effectue 100 étapes de calcul »

Généralisation

- Théorèmes valables pour toute formalisation des algorithmes
- Version langage de programmation :
  - on ne sait *rien* décider sur ce que fait un programme donné
  - on sait décider des caractéristiques du programme lui-même

Table des matières

Petit rappel

Cours précédent

Puisque l'ensemble des algorithmes est dénombrable,

- il existe des langages non reconnaissables
- il existe des langages non décidables
- il existe des fonctions non calculables

Partie précédente de ce cours

- Le problème de l'arrêt H est non calculable / le langage K est indécidable
- Toutes les propriétés non triviales de $L(A)$ sont indécidables

Et les langages non reconnaissables alors ? Où sont-ils ?

Un premier résultat positif

Proposition

Le langage $K = \{\langle A, w\rangle : A \text{ termine sur l'entrée } w\}$ est reconnaissable

Preuve

1. Sur l'entrée $\langle A, w\rangle$, simuler $A(w)$
2. Si $A(w)$ boucle, renvoyer 1.

Complémentaire et co-reconnaissable

Définition

- Complémentaire d'un langage $L \subset \{0,1\}^* : \overline{L} = \{w : w \notin L\}$
- Un langage est co-reconnaissable si son complémentaire est reconnaissable
  - Reconnaissable : il existe un algorithme tel que $A(w) = 1 \iff w \in L$
  - Co-reconnaissable : il existe un algorithme tel que $A(w) = 1 \iff w \notin L$

Exemple

- Problème de l'arrêt :
  - $K = \{\langle A, w\rangle : A \text{ termine sur l'entrée } w\}$
  - $\overline{K} = \{\langle A, w\rangle : A \text{ boucle sur l'entrée } w\}$
  - K est reconnaissable, donc $\overline{K}$ est co-reconnaissable

Version « problème de décision »

- Complémentaire de $f : \{0,1\}^* \to \{0,1\}$:

$$\overline{f} : \{0,1\}^* \to \{0,1\}$$

$$w \mapsto \begin{cases} 1 & \text{si } f(w) = 0 \\ 0 & \text{si } f(w) = 1 \end{cases}$$

Un deuxième résultat positif

Théorème

Un langage est décidable si et seulement s'il est reconnaissable et co-reconnaissable

Preuve

($\Rightarrow$) (quasi) trivial : si A décide L, $A(w)=1 \Leftrightarrow w \in L$

l'algo $\bar{A}(w) = 1 - A(w)$: $\bar{A}(w)=1 \Leftrightarrow w \notin L$.

($\Leftarrow$) On a A^+ tq $w \in L \Leftrightarrow A^+(w)=1$

A^- tq $w \notin L \Leftrightarrow A^-(w)=1$

Idée On simule A^+ et A^- en parallèle et on répond dès que l'un répond 1.

-> Forcément l'un des deux va répondre 1

-> Simuler en parallèle : exécuter une étape de A^+, puis une étape de A^-, puis ...

Une conséquence négative

Corollaire

Le langage $\overline{K} = \{\langle A, w\rangle : A \text{ boucle sur l'entrée } w\}$ n'est pas reconnaissable

Preuve

- Supposons $\overline{K}$ est reconnaissable : alors K est co-reconnaissable
- Comme K est reconnaissable, il est donc décidable
- Contradiction : $\overline{K}$ ne peut pas être reconnaissable

Généralisation

- Si L est reconnaissable mais pas décidable, $\overline{L}$ n'est pas reconnaissable

OK mais ni reconnaissable ni co-reconnaissable ?

Qu'est-ce qu'on a pour l'instant ?

- Problème de l'arrêt (langage K, fonction H) : reconnaissable / non décidable
- Complémentaire $\overline{K}$, $\overline{H}$: co-reconnaissable / non décidable
- Il existe des langages ni reconnaissables, ni co-reconnaissables
  - Ensemble des algorithmes dénombrable
  - Ensemble des langages indénombrable

Peut-on construire un langage ni reconnaissable, ni co-reconnaissable ?

Théorème

Le langage $\{\langle A, w, 1\rangle : A(w)\downarrow\} \cup \{\langle A, w, 0\rangle : A(w)\uparrow\}$ n'est ni reconnaissable ni co-reconnaissable

Preuve du théorème

1. Non reconnaissable : supp qu'il existe R tq

$$R(\langle A, w, \varepsilon \rangle) = 1 \iff \begin{cases} A(w)\downarrow \text{ et } \varepsilon = 1 \\ \text{ou} \\ A(w)\uparrow \text{ et } \varepsilon = 0 \end{cases}$$

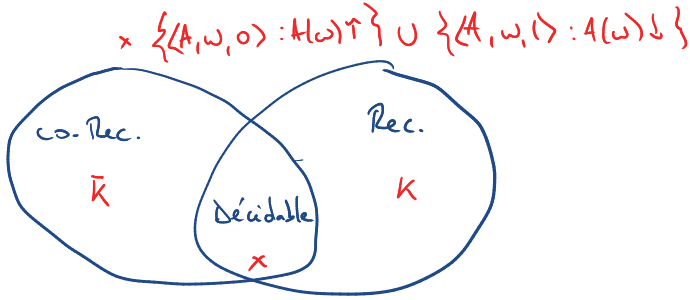
Soit $\bar{H}(\langle A, w \rangle)$ qui simule $R(\langle A, w, 0 \rangle)$.

Alors $\bar{H}(\langle A, w \rangle) = 1 \iff A(w)\uparrow$. Donc $\bar{H}$ reconnaît $\bar{K}$.

2. Non co-reconnaissable : idem en simulant $R(\langle A, w, 1 \rangle)$

Bilan

Un dessin



Un théorème de Rice ?

- Il existe une théorème similaire pour les langages non reconnaissables
  - Théorème de Rice-McNaughton-Myhill-Shapiro
  - Trop technique pour ce cours

Table des matières

Reconnaissable = récursivement énumérable

Définition

- *Récursivement énumérable* : il existe un algorithme qui *énumère* tous les mots de L
  - l'énumération dure un temps infini mais il ne s'écoule qu'un temps fini entre deux mots
  - chaque mot peut être affiché plusieurs fois, et aucun ordre d'énumération n'est fixé
- Formalisation avec une machine de Turing avec un *ruban d'affichage*

Théorème

Un langage est récursivement énumérable si et seulement s'il est reconnaissable

Idée de la preuve

$(\Rightarrow)$ Trivial : on énumère tous les mots jusqu'à tomber sur celui qu'on veut.

$(\Leftarrow)$ On énumère tous les couples $(x, t) \in \Sigma^* \times \mathbb{N}$ dans un ordre qcq.

R : algo de reconnaissance. Pour chaque couple on simule $R(x)$ pendant t étapes. Si $R(x) = 1$ en t étapes, on affiche x.

Incalculabilité des fonctions

Cas des problèmes de décision

- $f : \Sigma^* \to \{0,1\}$ est
  - calculable s'il existe un algorithme A tel que $A(w) = f(w)$ pour tout w
  - reconnaissable s'il existe un algorithme A tel que $A(w) = 1 \iff f(w) = 1$

Et pour les problèmes généraux ?

- $f : \Sigma^* \to \Sigma^*$ est
  - calculable s'il existe un algorithme A tel que $A(w) = f(w)$ pour tout w
  - pas de notion équivalente à *reconnaissable*

Trouver des fonctions incalculables

- Dénombrable / indénombrable : il existe des fonctions $f : \Sigma^* \to \Sigma^*$ non calculables
- Construction : similaire au cas des problèmes de décision
- Exemple : $f(\langle A \rangle) =$ *taille* du plus petit algorithme qui reconnaît $L(\langle A \rangle)$

Réductions

À quoi ressemblent les preuves de non-décidabilité/reconnaissabilité ?

- Preuves par diagonalisation :
  - On suppose que L est décidé par un algorithme A
  - On construit A' à partir de A qui est contradictoire
- *Réduction* à un problème indécidable / irreconnaissable :
  - Si L est décidable, alors K également
  - Comme K ne l'est pas, L ne peut pas l'être

Définition

- Une *réduction* (calculable, fonctionnelle) d'un langage L_1 à un langage L_2 est une fonction calculable $f : \Sigma^* \to \Sigma^*$ telle que $w \in L_1 \iff f(w) \in L_2$
- On dit que L_1 est *réductible à* L_2, noté $L_1 \leq_m L_2$

Propriétés

Soit L_1, L_2 tel que $L_1 \leq_m L_2$

- Si L_1 est indécidable, L_2 est indécidable
- Si L_1 est (co-)irreconnaissable, L_2 est (co-)irreconnaissable

Conclusion

Constructions explicites

- Langages non décidables / fonctions non calculables
- Langages non reconnaissables
- Langages ni reconnaissables ni co-reconnaissables

Théorèmes de l'arrêt et de Rice

- Aucun algorithme ne peut *expliquer* ce que fait un programme
- Attention !
  - Pas d'algorithme = pas de méthode générale qui marche à tous les coups
  - Mais un humain peut être capable de répondre pour *certains* programmes
  - D'ailleurs, un algorithme aussi ! *Vérification*

Incalculabilité hors des algorithmes / machines de Turing ?

- Mêmes résultats quel que soit le modèle (automates cellulaires, λ-calcul, …)
  - Même en croisant : une machine de Turing ne peut décider l'arrêt d'une machine RAM
- Et au delà : prochain cours !

Bonus 1 : preuve directe d'un langage non-reconnaissable

Théorème

Le langage $D = \{\langle A \rangle : A \text{ n'accepte pas le mot } \langle A \rangle\}$ n'est pas reconnaissable

Preuve

Non traité en cours

Bonus 2 : fonctions totales

Théorème

Le langage $T = \{\langle A \rangle : A \text{ s'arrête sur toute entrée}\}$ n'est ni reconnaissable, ni co-reconnaissable

Preuve

Non traité en cours