Chapitre II

Décidabilité, calculabilité, réductibilité

Introduction

Ce chapitre se décompose de la manière suivante :

- **le problème de la décidabilité :**
  Les problèmes de décision peuvent-ils toujours être décidé ?
  Si non, lesquels peuvent l'être ?
- **le problème de la calculabilité :**
  Les problèmes d'évaluation peuvent-ils toujours être calculé ?
- **le principe de réductibilité :**
  Principe de démonstration permettant de déduire les propriétés d'un problème à partir des propriétés d'un autre problème (utilisé aussi lors des calculs de complexité).

Deux autres résultats importants seront également présentés :

- **le théorème de récursion :**
  Il affirme la possibilité pour une MT de se dupliquer, ce qui conduit à sa possibilité de disposer de son propre code.
- **le problème de la décidabilité des théories logiques**
  Est-il possible toujours possible de déterminer si un énoncé logique est vrai ou faux ?

1 Décidabilité

1.1 Classification

On voudrait maintenant classifier l'ensemble des programmes d'une machine de Turing qui peuvent servir à quelque chose.

On peut s'entendre sur les points suivants :

1) un programme qui ne s'arrête jamais quelque soit l'entrée ne sert à rien.
2) un programme qui s'arrête toujours quelque soit l'entrée fait quelque chose (peu importe quoi).

Si on veut considérer l'ensemble des programmes possibles sur une MT, il faut aussi considérer la catégorie suivante :

3) un programme qui s'arrête pour certaines entrées (celles pour lesquelles le programme a un sens ; trouve une solution par exemple), mais qui bouclent pour d'autres (n'a pas de sens ; ne trouve pas de solution).

Certes, si on ne connaît pas les entrées pour lesquelles le programme boucle, la machine ne s'arrête jamais dans ce cas.

Ces programmes sont, pour l'instant, même s'ils sont problématiques, impossible à ignorer car on ne peut pas savoir s'il est *a priori* possible de détecter qu'un programme boucle.
Renversons la perspective : regardons maintenant ce problème sous l'angle des langages.

Cet angle est plus intéressant car un même langage peut être reconnu par des programmes différents.

On veut connaître la puissance effective des MTs, en s'intéressant à ce qu'elles **savent faire** et non à la **façon de le faire**.

On définit un langage L acceptés par une machine de Turing M comme étant l'ensemble des mots $w \in \Sigma^*$ tels que $M(w)$ accepte. On note $L = \mathcal{L}(M)$.

Conséquence de la définition d'un langage : si une MT M boucle sur un mot, alors elle n'accepte pas ce mot. Boucler sur un mot est équivalent à le rejeter.

En ne présumant rien sur les capacités des MTs, on veut savoir si, pour n'importe quel langage L, on est dans l'un des cas suivants :

1. il existe une MT qui reconnaît L et qui ne boucle pas.
2. toute MT qui reconnaît L boucle nécessairement sur au moins un mot.
3. il n'existe aucune MT qui reconnaît L.

Formalisons maintenant ces notions.

1.1.1 Langage récursivement énumérable

Définition II.1 (langage récursivement énumérable)

Un langage L est dit (récursivement) énumérable s'il existe une machine de Turing qui l'accepte.

Notes : Initialement, la définition d'un langage énumérable était "qui peut être énuméré par un énumèrateur". Voir la définition d'un énumèrateur plus loin.

Définition II.2 (Classe des langages récursivement énumérables)

La classe des langages récursivement énumérables est constituée de l'ensemble des langages reconnus par une machine de Turing.
Notation : On note $\mathcal{RE}$ cette classe.

Pour un langage L récursivement énumérable, une MT M qui reconnaît L ne peut faire que trois choses sur un mot d'entrée w,

- l'accepter ($w \in L$),
- le rejeter ($w \notin L$),
- ne pas s'arréter (boucle infinie, $w \notin L$).

Si $w \notin L$, il est possible que M ne donne jamais de réponse.

C'est la définition la plus faible qui puisse être donnée au fait qu'un langage est reconnu par une MT.

1.1.2 Langage décidable

En même temps, ceci est problématique, car on s'attendrait à ce qu'une machine qui reconnaît un langage donne une réponse définitive pour toute entrée.

On introduit alors la notion de décidabilité :

Définition II.3 (langage décidable)

Un langage L est décidable s'il est récursivement énumérable (*i.e.* il existe une machine de Turing M qui accepte L) **et** que M s'arrête pour toutes les entrées.
On dit alors que la MT M décide L.

Autrement dit, si L est décidable, alors il existe une MT M telle que :

- si $w \in L$, alors M accepte w.
- si $w \notin L$ alors M rejette w.

i.e. M s'arrête dans tous les cas.

Définition II.4 (Classe des langages décidables)

La classe des langages décidables est constituée de l'ensemble des langages décidables.
Notation : On note $\mathcal{R}$ cette classe.

Cette notion est approfondie maintenant.

1.2 Lien entre énumérabilité et décidabilité

Donc : un langage L est :

- **énumérable** $\stackrel{\Delta}{=}$ s'il existe une MT M qui accepte tout $w \in L$.
  i.e. si $w \notin L$, alors M rejette ou boucle.
- **décidable** $\stackrel{\Delta}{=}$ s'il existe une MT M telle que M qui accepte tout $w \in L$ et s'arrête toujours ; *i.e.* si $w \notin L$, alors M rejette et ne boucle jamais.

La décidabilité est une notion plus forte que l'énumérabilité.

Proposition II.1

Si un langage L est décidable, alors il est énumérable.

Démonstration:

Par définition, une MT est décidable si elle est énumérable et ne boucle jamais. □

Proposition II.2

Soit un langage L et $\overline{L}$ son complémentaire.
$$(L \text{ est décidable}) \Leftrightarrow \left(\overline{L} \text{ est décidable}\right)$$

DÉMONSTRATION:

$\Rightarrow$ soit M une MT qui décide L.
Pour tout $w \in \Sigma^*$, M s'arrête toujours soit dans l'état acceptant q_a, soit dans l'état rejetant q_r.
Soit $\overline{M}$ la MT construite à partir de M en inversant q_a et q_r. Alors, trivialement, $\overline{M}$ s'arrête toujours et décide $\overline{L}$.

Même argument avec un code : soit M la MT qui décide L. Soit la machine $\overline{M}$ suivante construite à partir de M :

$\overline{M}(w) =$

```
EXÉCUTER M(w)              // s'arrête toujours car M décidable
DÉCIDER l'opposé de M      // = accepter si rejette , rejeter si accepte
```

Clairement, $\overline{M}$ décidable et décide $\overline{L}$
$\Leftarrow$ soit $\overline{M}$ une MT qui décide $\overline{L}$.
Même idée en utilisant $\overline{L}$ pour définir M. □

REMARQUES 9: Notations

- $\mathcal{RE} \stackrel{\Delta}{=}$ classe des langages énumérables.
- $co\mathcal{RE} \stackrel{\Delta}{=}$ classe des langages dont le complément est énumérable.
- $\mathcal{R} \stackrel{\Delta}{=}$ classe des langages décidables.

Attention : à bien comprendre la co-énumérabilité. La définition n'implique que les assertions suivantes :

- $(L \in co\mathcal{RE}) \Leftrightarrow (\bar{L} \in \mathcal{RE})$ = un langage est co-énumérable si est seulement si son complémentaire est énumérable.
- $(L \in co\mathcal{RE})$ n'implique pas $(L \in \mathcal{RE})$ = un langage co-énumérable n'est pas nécessairement énumérable.
- $(L \in \mathcal{RE})$ n'implique pas $(L \in co\mathcal{RE})$ = un langage énumérable n'est pas nécessairement co-énumérable.

Avec ces notations, les constatations deviennent :

- si un langage L est décidable, alors il est énumérable : $\mathcal{R} \subseteq \mathcal{RE}$.
- pour un langage L, si $\bar{L}$ est énumérable (*i.e.* $\bar{L} \in \mathcal{RE}$), alors $L \in co\mathcal{RE}$ (par définition).
- si L est décidable, alors $\bar{L}$ aussi ($\Rightarrow \bar{L} \in \mathcal{R}$). D'où $\mathcal{R} \subseteq co\mathcal{RE}$.
- donc $\mathcal{R} \subseteq \mathcal{RE} \cap co\mathcal{RE}$

Théorème II.1 (lien entre décidabilité et énumérabilité)

$\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$

DÉMONSTRATION:

- $\mathcal{R} \subseteq \mathcal{RE} \cap co\mathcal{RE}$: déjà démontré.
- $\mathcal{R} \supseteq \mathcal{RE} \cap co\mathcal{RE}$

  $\Leftrightarrow$ si $L \in \mathcal{RE} \cap co\mathcal{RE}$, alors $L \in \mathcal{R}$
  $\Leftrightarrow$ si L et son complément sont énumérables, alors L est décidable.

  Soit M une MT qui accepte L.
  Soit $\overline{M}$ une MT qui accepte $\bar{L}$.
  Construisons une MT D qui exécute M et $\overline{M}$ en parallèle :

- si M accepte, alors M accepte.
- si $\overline{M}$ accepte, alors M rejette.

Comment exécuter M et $\overline{M}$ en parallèle : prendre 2 bandes (une pour chaque) et alterner entre les 2 machines (et éviter les boucles d'une des 2 machines).

Montrons que D décide L.
Toute chaîne w est soit dans L, soit dans $\bar{L} \Rightarrow$ soit M, soit $\overline{M}$ accepte w.
D s'arrête chaque fois que M ou $\overline{M}$ accepte $\Rightarrow D$ s'arrête pour tout w.
De plus par construction, D accepte les chaînes de L et rejette les chaînes de $\bar{L}$. Donc D décide L, et L est décidable. □

Démonstration: (idem avec un code)

Considérons la MT D suivante :

$D(w) =$

```
RÉPÉTER à l'infini
  EXÉCUTER M sur une transition
  SI M ACCEPTE ALORS ACCEPTER   // M accepte si w ∈ L
  EXÉCUTER M̄ sur une transition
  SI M̄ ACCEPTE ALORS REJETER    // M̄ accepte si w ∉ L
```

D s'arrête toujours car $w \in L \cup \overline{L} = \Sigma^*$, et n'accepte que si $w \in L$. Donc D décide L et L décidable. □

Théorème II.2 ((le même reformulé))

Un langage est décidable si et seulement si il est à la fois énumérable et co-énumérable.

Exercice 2 (Fermeture de $\mathcal{R}$)**.** *Montrer que $\mathcal{R}$ est :*

1. *fermé par complémentarité, à savoir $L \in \mathcal{R} \Leftrightarrow \overline{L} \in \mathcal{R}$*
2. *fermé par union, à savoir $(L_1, L_2) \in \mathcal{R}^2 \Leftrightarrow L_1 \cup L_2 \in \mathcal{R}$*
3. *fermé par intersection (modifier la démonstration précédente)*
4. *fermé par concaténation, $(L_1, L_2) \in \mathcal{R}^2 \Leftrightarrow L_1 \circ L_2 \in \mathcal{R}$,*
   où $\circ$ est l'opérateur de concaténation défini par $L_1 \circ L_2 = \{w_1 w_2 \mid w_1 \in L_1 \text{ et } w_2 \in L_2\}$; à savoir l'ensemble des mots obtenus par concaténation d'un mot de L_1 et d'un mot de L_2.
5. *fermé par l'opérateur de Kleene $*$: $L \in \mathcal{R} \Leftrightarrow L^* \in \mathcal{R}^*$.*
   $w \in L^* \Leftrightarrow w = \varepsilon$ ou $\exists k \geq 1$ tel que $w = w_1 w_2 \dots w_k$ et $\forall k, w_k \in L$.

Exercice 3 (Fermeture de $\mathcal{RE}$)**.** *Montrer que $\mathcal{RE}$ est :*

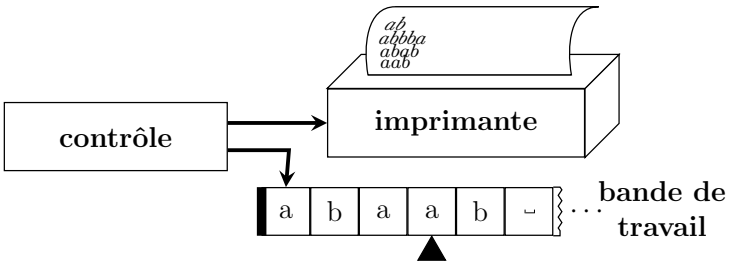
1. *fermé par union.*
2. *fermé par concaténation.*
3. *fermé par l'opérateur de Kleene.*

Attention : *si $L \in \mathcal{RE}$, alors toute machine de Turing M_L telle que $\mathcal{M} = L$ peut boucler sur n'importe quelle entrée.*

1.3 Énumèrateur

Énumèrateur : type particulier de MT avec une imprimante attachée servant à écrire l'ensemble des chaînes contenues dans le langage reconnu par cette MT.

- énumération dans n'importe quel ordre, éventuellement avec répétition.
- ne possède pas d'entrée (mais possède une bande pour travailler).



Le code d'un énumérateur E pour un langage L est schématiquement le suivant :

$E_L() =$
RÉPÉTER
  CALCULER le mot w suivant du langage L
  SI tous les mots ont déjà été tous calculés
  ALORS ARRÊTER
  SINON AFFICHER w

EXEMPLE 16: énumèrateur Le code de l'énumèrateur E_{MULTIPLE} qui écrit tout les couples de la forme (n, p) où $n \in \mathbb{N}$, $p \in \mathbb{N}$ et n est un multiple de p est le suivant :

$E_{\text{MULTIPLE}}() =$
POUR $n = 1, 2, \ldots$
  POUR $1 \leq p \leq n$
    SI $n\%p = 0$ **ALORS AFFICHER** (n, p)

REMARQUE 10:
Évidemment, si un énumèrateur E reconnaît un langage L, et que $\#L$ n'est pas fini, l'énumérateur ne s'arrête jamais.

Théorème II.3 (lien énumèrateur-MT)
Un langage est accepté par une MT si et seulement si il existe un énumèrateur qui l'énumère.

DÉMONSTRATION:
$\Leftarrow$: $\exists$ énumèrateur E qui énumère $L \Rightarrow \exists$ MT M qui reconnaît L

on construit la MT $M(w)$ comme : exécuter l'énumérateur pour obtenir l'ensemble des mots v, et accepter si $v = w$.

$M(w) =$
RÉPÉTER
  CALCULER le mot v suivant du langage L
  SI tous les mots ont été calculé
  ALORS REJETER
  SINON SI $w = v$ **ALORS ACCEPTER**

Clairement, M accepte les sorties de E, donc M reconnaît L.

$\Rightarrow$: $\exists$ MT M qui reconnaît $L \Rightarrow \exists$ énumèrateur E qui énumère L

Soit $s_1, s_2, s_3, \ldots$ l'ensemble des chaînes possibles de Σ^*.

On construit E de la manière suivante :

$E_L() =$
POUR i=1,2,3,...
  POUR $s_j = s_1, s_2, \ldots, s_i$
    EXÉCUTER $M(s_j)$ sur i transitions
    SI $M(s_j)$ **ACCEPTE**
    ALORS AFFICHER s_j

La limitation de l'exécution à i transitions permet d'éviter d'être bloqué au cas où un s_j ferait boucler M.

Inconvénient : la même sortie peut apparaître de très nombreuses fois.

□

1.4 Existence de langages non énumérables

Commençons par montrer l'existence de langages non énumérables, à savoir de langages reconnus par aucune machine de Turing.

Pour ce faire, nous avons besoin de quelques outils théoriques supplémentaires.

1.4.1 Compléments théorique

Comment comparer la taille de deux ensembles de tailles finies ?

Facile ! Il suffit de les compter.
Exemple : soit $n_A = \#A$ et $n_B = \#B$
si $n_A \neq n_B$, alors ils n'ont pas la même taille.

Définition II.5 (bijection (rappel))

une bijection de A dans B est une fonction f tq :

- si $(a_1, a_2) \in A^2$ et $a_1 \neq a_2$ alors $f(a_1) \neq f(a_2)$
- si $\forall b \in B$ alors $\exists! a \in A$ tel que $f(a) = b$

Autrement dit, tout élément de A est associé à un élément de B unique et vice-versa.

Autre méthode : s'il existe une bijection entre A et B
alors ils ont la même taille.

Comment comparer la taille de deux ensembles de tailles infinies ?

Il n'est plus possible de les compter.
Par contre, il est possible de définir une bijection entre 2 ensembles infinis.
On prend donc la **définition** suivante pour comparer la taille d'ensembles infinis.

Définition II.6 (Comparaison de la taille de deux ensembles)

Deux ensembles A et B ont la même taille s'il existe une bijection entre A et B.

Quelle conséquence pour des ensembles infinis ?

Première affirmation :
L'ensemble des entiers naturels $\mathbb{N}$ a la même taille que l'ensemble $\mathbb{E}$ des nombres pairs.
Démonstration : prendre la bijection $f(i) = 2i$.
$\mathbb{E}$ est un sous-ensemble de $\mathbb{N}$, mais ils ont la même taille !

Définition II.7 (Ensemble dénombrable)

Un ensemble A est dénombrable si :

- soit A est un ensemble fini.
- soit A a la même taille que $\mathbb{N}$ (ensemble des entiers naturels).

Notation : $\aleph_0 = \#\mathbb{N}$

Deuxième affirmation :

Proposition II.3

L'ensemble des entiers relatifs $\mathbb{Z}$ est dénombrable.

Démonstration:

prendre la bijection :

$$f(i) = \begin{cases} i/2 & \text{si } i \text{ est pair} \\ -(i+1)/2 & \text{si } i \text{ est impair} \end{cases}$$

□

Cette fois ci, $\mathbb{N}$ est un sous-ensemble de $\mathbb{Z}$.
Pourtant, ils ont encore la même taille.

Tous les ensembles sont-ils dénombrables ?

Ensemble des rationnels positifs $\mathbb{Q}^+ = \left\{ m/n \mid (m,n) \in \mathbb{N}^{*2} \right\}$:
Troisième affirmation :

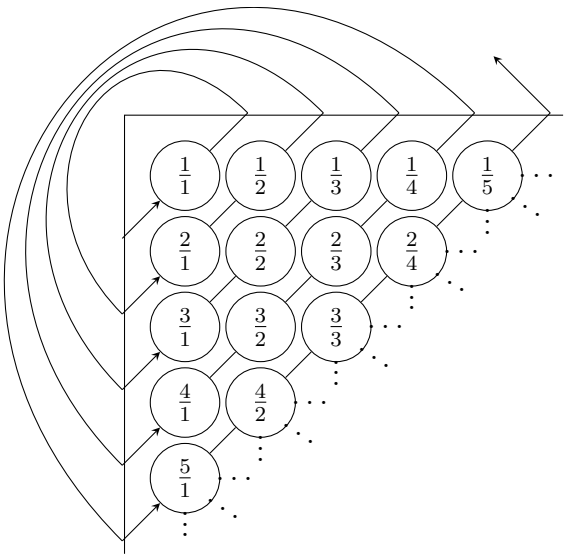
Proposition II.4

L'ensemble des entiers rationnels positifs $\mathbb{Q}^+$ est dénombrable.

Plus difficile à démontrer. **Démonstration:**

Première approche : On considère $\mathbb{Q}$ comme un tableau à 2 dimensions.
Commençons par compter la première ligne ...
Zut, elle est infinie !

Seconde approche : On compte le long des diagonales et en sautant les doubles comme ceci :



C'est une union dénombrable (= autant de diagonales que $\mathbb{N}$) d'ensembles (= éléments sur la diagonales) qui sont chacun fini (= la taille de chaque diagonale est finie).

Or, toute une union dénombrable d'ensembles finis est un ensemble dénombrable. Démontrons-le dans ce cas particulier.

La $n^{\text{ème}}$ diagonale contient n éléments. Par ailleurs, il y a $N_n = n(n-1)/2 = 1 + 2 + \cdots + n - 1$ éléments sur toutes les diagonales précédentes. En conséquence, on peut construire une bijection qui affecte les éléments $\{1, \ldots, n\}$ de la $n^{\text{ème}}$ diagonales aux élements $\{N_n + 1, \ldots N_n + n\}$ de $\mathbb{N}$.

Cette propriété est vraie dans le cas général.

Donc $\mathbb{Q}^+$ est dénombrable. □

Ensemble des nombres réels $\mathbb{R}$:

Théorème II.4

$\mathbb{R}$ n'est pas dénombrable.

Notation : $\aleph_1 = \#\mathbb{R}$

Remarque 11: Rappel

sur les nombres réels Tout nombre réel a une représentation décimale.
Exemple : $\pi = 3.1415926\ldots$, $\sqrt{2} = 1.4142136\ldots$
$0 = 0.\underline{0}$ où $\underline{0} = 0$ suivi d'une infinité de 0.

Équivalences : $1.6\underline{9} = 1.7\underline{0}$

Démonstration: par l'absurde

Hypothèse : supposons qu'il existe une bijection entre $\mathbb{N}$ et $\mathbb{R}$.
Il est donc possible de construire une table :

n	$f(n)$
1	$3.14159\ldots$
2	$55.55555\ldots$
3	$40.18642\ldots$
4	$15.20601\ldots$

Montrons que l'on peut trouver un réel x qui n'est pas dans cette liste.

Construisons un x, tel que $0 \leq x \leq 1$ qui soit différent de $f(n)$, $\forall n$.

La $n^{\text{ème}}$ décimale de x est construite à partir de la $n^{\text{ème}}$ décimale de $f(n)$.

n	$f(n)$
1	$3.\mathbf{1}4159\ldots$
2	$55.5\mathbf{5}555\ldots$
3	$40.18\mathbf{6}42\ldots$
4	$15.206\mathbf{0}1\ldots$

(méthode de diagonalisation)

La $n^{\text{ème}}$ décimale de x est choisie comme :

- différente de la $n^{\text{ème}}$ décimale de $f(n)$
- différente de 0 ou 9 (pour éviter les équivalences).

Ainsi construit, x a toujours au moins une décimale différente de $f(n)$ pour tout n.

Donc, il n'existe pas de bijection entre $\mathbb{N}$ et $\mathbb{R}$.
Donc, $\mathbb{R}$ n'est pas dénombrable. □

1.4.2 Preuve d'existence de langages non énumérables

Montrons maintenant qu'il existe des langages non énumérables.

On rappelle que si un langages n'est pas énumérable, alors il n'existe aucune machine de Turing qui accepte le langage associé à ce problème.

Lemme II.1
si Σ un ensemble fini, alors Σ^* est dénombrable.

Démonstration:
Notons Σ_n les chaînes de longueurs n de Σ^*. Il est clair que $\Sigma^* = \bigcup_{i\in\mathbb{N}} \Sigma_i$.
Or, tous ces ensembles sont finis : $\Sigma_n = (\#\Sigma)^n$.
Donc Σ^* est une union dénombrable d'ensembles finis.
Or, une union dénombrable d'ensembles finis est aussi dénombrable. □

Lemme II.2
L'ensemble des MTs est dénombrable.

Démonstration:
Chaque MT M est codée comme une chaîne $\langle M \rangle$.
Or, cette chaîne s'écrit à partir de symboles d'un alphabet Σ.
Donc, il existe une injection de l'ensemble des MTs dans Σ^* (i.e. toutes les MTs sont codées dans Σ^*, mais toutes les chaînes de Σ^* ne représentent pas des MTs).
Or, Σ^* est dénombrable. Donc l'ensemble des MTs aussi. □

Soit $\mathbb{B}$ l'ensemble des suites binaires de longueur infinie.

Lemme II.3
$\mathbb{B}$ n'est pas dénombrable.

Démonstration:
Réutiliser l'argument de diagonalisation pour construire une suite binaire de longueur infinie qui n'est dans aucun ensemble dénombrable d'éléments de $\mathbb{B}$. **Exercice :** faire la démonstration complète. □

Lemme II.4
L'ensemble des langages $\mathcal{L}$ sur un alphabet Σ fini n'est pas dénombrable.

Démonstration:
Soit $\Sigma^* = \{s_1, s_2, s_3, \dots\}$ l'ensemble (dénombrable) des mots reconnus par L. Pour $b \in \mathbb{B}$, on note b_i le $i^{\text{ème}}$ bit de b.
Soit $\chi : \mathcal{L} \to \mathbb{B}$ qui associe à tout langage $L \in \mathcal{L}$ une suite caractéristique unique $b \in \mathbb{B}$, définie par $b_i = 1$ si et seulement si $s_i \in L$ (et 0 sinon).
Or, l'application χ est une bijection (par construction).
Comme $\mathbb{B}$ n'est pas dénombrable, $\mathcal{L}$ n'est pas non plus dénombrable. □

Pour résumer :

- L'ensemble des MTs est dénombrable.
- l'ensemble $\mathcal{L}$ de tous les langages sur un alphabet fini Σ n'est pas dénombrable.

Corollaire II.1
Il existe des langages qui ne sont acceptés par aucune MT.

DÉMONSTRATION:

Une MT ne reconnaît qu'un seul langage.
Il y a infiniment plus de langages que de MT.
Donc, certains langages ne peuvent pas être reconnus par une MT. □

On a donc la preuve de l'existence de langages non énumérables (sans en exhiber un seul).

En terme informatique, cela signifie qu'**aucun programme** n'est capable de reconnaître de tels langages.

Pire : l'essentiel des langages ne sont pas énumérables (il y a infiniment plus de langages que de programmes qu'il est possible d'écrire).

Conséquence : le modèle de la machine de Turing est très limité sur le nombre de langages qu'il est capable de reconnaître.
Néanmoins, tout problème de décision ayant un sens pour nous se formule sous forme d'une chaîne de longueur finie avec un nombre fini de symboles (par exemple : langage constitué de l'ensemble des nombres premiers). Le nombre de ces problèmes est donc fini.

Est-il possible de formuler un problème de décision non énumérable ?

Nous n'avons pas de réponse à cette question pour le moment. Restreignons un peu nos ambitions, et regardons s'il est possible de trouver des problèmes énumérables mais pas décidables.

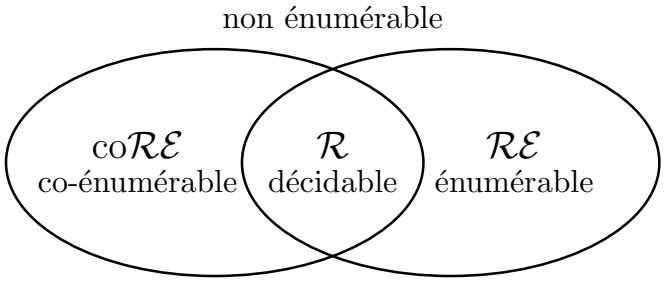
Exercice 4. *Prouver qu'il existe un sous-ensemble non récursivement énumérable de* $\{1\}^*$.

1.4.3 Première synthèse sur les classes de langages

Structuration des langages

- Tous les ensembles $\mathcal{RE}$, *co*$\mathcal{RE}$ et $\mathcal{R}$ sont dénombrables (car ils ne contiennent que des MTs).
- Il manque tous les langages non énumérables.

Les langages s'organisent donc comme suit :



1.5 Problèmes de décidabilité

On s'intéresse maintenant aux problèmes pour lesquels ils est possible d'écrire une machine de Turing (= au moins énumérables).

La décidabilité d'une problème (et donc de son algorithme) est une question fondamentale, en effet :

- si un problème est décidable, alors il existe **nécessairement** un programme qui décide le problème (= accepte ou rejette) et s'arrête pour n'importe quelle entrée.
  = pour tout mot d'entrée, si j'attends assez longtemps, je suis sûr que le programme me donnera la réponse.
- si le problème est indécidable, alors pour n'importe quel programme M qui accepte le langage L associé, il existe au moins un mot qui n'appartient pas au langage pour lequel M boucle.
  = pour tout mot d'entrée, si je n'ai pas de réponse, il n'y a aucun moyen de savoir si le programme est en train de calculer ou de boucler.

Clairement, on ne souhaite exécuter sur un ordinateur que des programmes correspondant à des problèmes décidables.

1.5.1 Premier exemple de problème indécidable

Au début du XX$^{\text{ème}}$ siècle, on pensait que tout problème mathématique correctement posé devait pouvoir être résolu, et que le seul obstacle à cette résolution était la complexité du problème.

Le 10$^{\text{ème}}$ problème de Hilbert :

En 1900, David Hilbert présente comme défi pour le 20$^{\text{ème}}$ siècle de résoudre 23 problèmes mathématiques centraux.

Le 10$^{\text{ème}}$ problème concerne directement l'algorithmique : "trouver un algorithme qui détermine si un polynôme multivarié à coefficients entiers possède une racine entière".

Clairement, la formulation de Hilbert induit que :

1. un tel algorithme existe.
2. il suffit de le trouver.

Exemples :

Soit $P_1(x, y, z) = 6x^3yzz^2 + 3xy^2 - x^3 - 10$
Existe-t-il $(x, y, z) \in \mathbb{Z}^3$ tel que $P_1(x, y, z) = 0$
Oui, prendre $(x, y, z) = (5, 3, 0)$.

Soit $P_2(x, y) = 3x^2 + 5y^2 + 7$.
Existe-t-il $(x, y) \in \mathbb{Z}^2$ tel que $P_2(x, y) = 0$
Non, toujours positif.

Soit $P_3(x, y) = x^2 - 991y^2 - 1$ (équation de Pell pour $d = 991$).
La plus petite solution entière est :

$$\begin{cases} x = 379516400906811930638014896080 \\ y = 12055735790331359447442538767 \end{cases}$$

Cas des polynômes à une variable (=univarié) :

Soit le langage :
$D = \{\langle p(x)\rangle \mid p(x)$ est un polynôme univarié avec une racine entière$\}$

On peut trouver une MT M qui reconnaît D :

$M(\langle p(x)\rangle) =$

POUR $x = 0, 1, -1, 2, -2, \ldots$
  ÉVALUER $p(x)$
  SI $p(x) = 0$ **ALORS ACCEPTER**

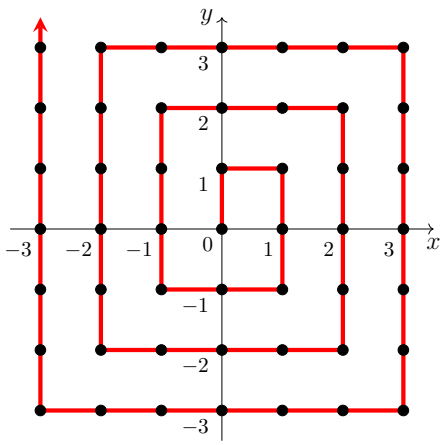
Si $p(x)$ a une racine entière, alors M finira par la trouver et l'accepter, sinon M ne s'arrêtera jamais.

Donc, ce langage est bien énumérable (= reconnu par une MT).

Cas des polynômes multivariés :

Une construction similaire est possible.

Par exemple en 2D pour les polynômes bivariés, on peut parcourir tous les couples entiers (x, y) ainsi :



Ceci est généralisable en dimension n.

Donc ce langage est aussi énumérable pour les polynômes multivariés.

Ces langages sont énumérables, mais sont-il décidables ?
Cas des polynômes à une variable

Théorème II.5 (Racines d'un polynôme univarié)

toutes les racines réelles de $p(x)$ vérifient $|x| < |kc_{\max}/c_1|$ où k = nombre de termes du polynôme, $c_{\max}$ = coefficient maximum et c_1 = coefficient d'ordre le plus élevé.

Modifions la MT M de la manière suivante :

$D(\langle p(x)\rangle) =$

$A = [-|kc_{\max}/c_1|, +|kc_{\max}/c_1|]$
POUR tous les x entiers dans A
  ÉVALUER $p(x)$
  SI $p(x) = 0$ **ALORS ACCEPTER**
REJETER

D s'arrête toujours en acceptant ou rejetant car A contient un nombre fini d'entiers. Donc, D **est décidable**.

Cas des polynômes multivariés :

C'est le $10^{\text{ème}}$ **problème de Hilbert**.

Ce problème n'est **pas décidable** (prouvé en 1970, par Yuri Matijasevic).

1.5.2 Problème de l'acceptation

Cas général

Le problème de l'acceptation consiste à construire un langage A constitué de couples $\langle M, w\rangle$ constitués :

- de l'ensemble des descriptions des machines M (qui reconnaisse chacune un certain langage),
- de l'ensemble des mots $w \in \Sigma^*$

tels que l'exécution de la machine M sur le mot w accepte.

A savoir :

$$A = \{\langle M, w\rangle |\ M \text{ est une machine qui accepte } w\}$$

Bien lire l'expression ci-dessus : A ne dépend pas d'un M ou un w particulier,

Le problème de l'acceptation est de savoir si A est décidable.

si A est décidable, cela signifie qu'il existe un programme D tel que, pour tout $\langle M, w\rangle \in A$, D peut décider si M accepte w.

Si A est décidable, le décideur D de A ne boucle jamais.

Proposition II.5

Si un décideur D de A existe, alors il existe un décideur pour toute machine T.

Démonstration:

Pour tout mot $w \in \Sigma^*$,

- si D accepte $\langle T, w\rangle$, alors $w \in \mathcal{L}(T)$.
- si D rejette $\langle T, w\rangle$, alors $w \notin \mathcal{L}(T)$.

Autrement dit, pour chaque machine T fixée, il est possible d'écrire le décideur D_T suivant :

$D_T(w) =$
construire $\langle T, w\rangle$ à partir de $\langle T\rangle$ **et** w
exécuter $D(\langle T, w\rangle)$
décider comme D // = accepter si accepte, rejeter si rejette

Chaque ligne du décideur D_T ne boucle jamais, et par conséquent, D_T ne boucle jamais. □

Cas des automates finis

Étudions maintenant le problème de l'acceptation dans le cas des automates finis.

Soient :

$$\begin{aligned} A_{\text{ADF}} &= \{< M, w > |\ M \text{ est un Automate Déterministe Fini qui accepte } w\} \\ A_{\text{ANF}} &= \{< M, w > |\ M \text{ est un Automate Non Déterministe Fini qui accepte } w\} \end{aligned}$$

Proposition II.6 (décidabilité des automates finis)

1. A_{ADF} est décidable.
2. A_{ANF} est décidable.

Intuitivement, cette proposition parait assez évidente.

La démonstration consiste à écrire une machine de Turing qui détermine ce que décide l'automate fini sur son entrée. Celle-ci doit donc simuler le fonctionnement de l'automate à partir de sa description $\langle M \rangle$ sur l'entrée w, et s'arrêter dans tous les cas.

Commençons partiellement la démonstration avec un exercice, que nous complétons plus tard.

Démonstration:

1. Le décideur doit d'abord vérifier que $\langle M \rangle$ est l'encodage d'un ADF, et que $w \in \Sigma^*$ où Σ est l'alphabet donné dans l'encodage de la description formelle de M.

   Il s'agit d'une suite de vérifications syntaxiques simples sur des listes de symboles.

   Par exemple, avec $\langle M \rangle = \{\langle \Sigma \rangle, \langle Q \rangle, q_0, \langle F \rangle, \langle \delta \rangle\}$, on doit vérifier que : $\langle \Sigma \rangle$ est liste de symboles distincts, $\langle Q \rangle$ est liste d'états distincts, $q_0 \in \langle Q \rangle$, $\langle F \rangle$ est liste d'états q_i distincts tels que $\forall i, q_i \in Q$, $\langle \delta \rangle$ est une fonction de transition (à savoir une liste de triplet (u, s, v) tel qu'il existe un seul triplet pour l'ensemble $(u, s) \in Q \times \Sigma$, et que $v \in Q$.

   Puis, il faut simuler l'ADF (voir exercice) avec le décideur D de l'exercice pour obtenir la décision. D'où A_{ADF} est décidable.

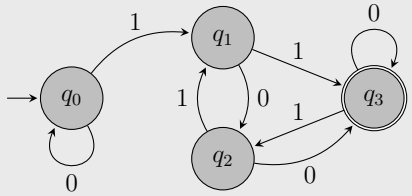
2. pour les ANFs, on peut démontrer (non fait) que tout ANF peut être converti en un ADF équivalent avec un algorithme T qui s'arrête toujours. Puis, on réutilise le décideur D de l'ADF.

   Le décideur D' d'un ANF M' est donc : $D'(\langle M', w \rangle) = D(\langle T(M'), w \rangle)$, et A_{ANF} est décidable.

□

Exercice 5 (Décidabilité des automates déterministes finis (ADF))**.**

1. *Pourquoi un ADF s'arrête-t-il toujours pour tout mot d'entrée w ?*
2. *Donner la définition formelle de l'ADF M suivant :*



3. *En déduire un codage $\langle M \rangle$ de M.*
4. *Avec des transformations simples, expliquer comment obtenir une machine de Turing qui reconnaît exactement le même langage que M.*
5. *Écrire une machine de Turing D qui prend $\langle M, w \rangle$ en entrée, et qui simule le fonctionnement de l'ADF M sur son entrée w. Indice : utiliser une machine multibande pour simplifier l'accès aux différentes caractéristiques de l'automate.*
6. *Montrer que D s'arrête toujours.*
7. *Peut-on déduire des questions précédentes que A_{ADF} est décidable ?*

Cas des grammaires algébriques

Passons maintenant aux grammaires algébriques :

Proposition II.7 (décidabilité des automates finis et des automates à pile)

1. $A_{\text{GA}} = \{< M, w > \mid M \text{ est une Grammaire Algébrique qui accepte } w\}$ est décidable.
2. $A_{\text{AP}} = \{< M, w > \mid M \text{ est une Automate à Pile qui accepte } w\}$ est décidable.

Démonstration: (grandes lignes)

1. savoir si un mot w est accepté par une GA peut être difficile en raison des règles du type $U \to \varepsilon|UU|0$ car il y a une infinité de façon d'obtenir une chaîne de longueur n.

   Mais, toute GA peut être mise sous forme normale de Chomsky = sous la forme $(S \to UU|a, U \to UV|a)$ où le symbole de départ S est le seul donnant ε, et toute variable est transformée en deux variables (hors S) ou en un terminal.

   En conséquence, pour savoir si un mot w de longueur n est généré par la GA, il suffit d'appliquer $n-1$ fois des règles de type $U \to UV$ et n fois des règles du type $U \to a$. Le nombre de combinaisons est alors fini (car le nombre de règles de chaque type est fini), et il suffit de les tester une à une.

   Si aucun mot généré ainsi n'est w, alors il ne peut pas être généré par la GA. Sinon, on accepte w. Cet algorithme s'arrête toujours, donc toute grammaire algébrique peut être décidée pour toute entrée. Donc A_{GA} est décidable.

2. tout AP peut être transformée en une GA équivalente avec un algorithme T qui s'arrête toujours.

□

Exercice 6 (Automate Linéaire Borné)**.** *Un automate linéaire borné (ABL) est une MT avec une bande de taille fixe (bornée à droite). Un déplacement à droite en fin de bande ne déplace pas le curseur de lecture.*

Soit M, un ABL à q états, g symboles de bande, et une bande de taille n.

1. *Montrer que le nombre de configurations différentes dans lequel un ABL peut se trouver est fini. On le calculera.*
2. *Soit $A_{ABL} = \{\langle M, w\rangle \mid M \text{ est un ABL qui accepte } w\}$. Montrer que A_{ABL} est récursivement énumérable.*
3. *Comment détecter si un ABL est en train de boucler ?*
4. *Montrer que le problème d'acceptation pour un ABL est décidable.*
5. *Est-il possible d'utiliser le résultat précédent pour détecter qu'un ordinateur limité (par exemple, avec 64 états, et 1Mo de mémoire) est en train de boucler ?*

Cas des machines de Turing

Et enfin, le cas de la machine de Turing :

$$A_{\text{MT}} = \{< M, w > \mid M \text{ est une Machine de Turing qui accepte } w\}$$

Théorème II.6 (décidabilité des machines de Turing)

1. A_{MT} est récursivement énumérable.
2. A_{MT} est indécidable.

Démonstration:

1. Une MT universelle U est une MT qui, $\forall M$, $\forall w$:
   - accepte $\langle M, w\rangle$ si M accepte w.
   - rejette ou boucle $\langle M, w\rangle$ si M rejette ou boucle sur w.

   Le langage reconnu par la MT universelle U est A_{MT}.

   A_{MT} est donc énumérable.

2. Montrons maintenant que A_{MT} n'est pas décidable.

Supposons qu'il existe un MT H qui décide A_{MT}.

Par définition, H doit donc avoir le comportement suivant pour toute machine M et chaîne w :

$$H(\langle M, w\rangle) = \begin{cases} \text{accepte} & \text{si } M \text{ accepte } w \\ \text{rejette} & \text{si } M \text{ n'accepte pas } w \end{cases}$$

Rappel sémantique : rejeter (= arrêt) ≠ ne pas accepter (= arrêt OU boucle).
Construisons maintenant une autre MT D qui utilise H de la façon suivante :

$D(\langle M\rangle) =$ **EXÉCUTER** $H(\langle M, \langle M\rangle\rangle)$
RETOURNER l'opposé de la décision de H.

Ainsi,

- D accepte $\langle M\rangle$ si H rejette $\langle M, \langle M\rangle\rangle$.
- D rejette $\langle M\rangle$ si H accepte $\langle M, \langle M\rangle\rangle$.

D s'arrête toujours (n'utilise que des opérations qui s'arrêtent toujours).

Note : exécuter une machine M sur sa propre description n'est pas aberrant. On peut écrire un compilateur C en C.

En reprenant la définition de H, D peut s'écrire :

$$D(\langle M\rangle) = \begin{cases} \text{accepte} & \text{si } M \text{ n'accepte pas } \langle M\rangle \\ \text{rejette} & \text{si } M \text{ accepte } \langle M\rangle \end{cases}$$

Que se passe-t-il maintenant si l'on utilise D sur lui-même ?

$$D(\langle D\rangle) = \begin{cases} \text{accepte} & \text{si } D \text{ n'accepte pas } \langle D\rangle \\ \text{rejette} & \text{si } D \text{ accepte } \langle D\rangle \end{cases}$$

Contradiction : D rejette $\langle D\rangle$ quand D accepte $\langle D\rangle$.

Donc, ni D, ni H n'existent.
Comme H n'existe pas, A_{MT} n'est pas décidable. □

On remarquera qu'il s'agit là d'une impossibilité logique, et non technique.
Une deuxième preuve de l'indécidabilité par la méthode dite de "diagonalisation".

DÉMONSTRATION:

(par diagonalisation)

Table $M(\langle M\rangle)$
si M_i accepte $\langle M_j\rangle$
alors (i, j) est "accept"

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	…
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					…
M_4	accept	accept			
⋮			⋮		

Table $H(\langle M, \langle M\rangle\rangle)$
si H accepte $\langle M_i, \langle M_j\rangle\rangle$
alors (i, j) est "accept"
sinon (i, j) est "reject"

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	…
M_1	accept	reject	accept	reject	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	…
M_4	accept	accept	reject	reject	
⋮			⋮		

Diagonalisation :
Ajout de $D(\langle M \rangle)$:

- opposé de la diagonale.
- problème pour $D(\langle D \rangle)$
- contradiction

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...	$\langle D \rangle$	...
M_1	accept	reject	accept	reject		accept	
M_2	accept	accept	accept	accept		accept	
M_3	reject	reject	reject	reject	...	reject	...
M_4	accept	accept	reject	reject		accept	
$\vdots$			$\vdots$		$\ddots$		
D	reject	reject	accept	accept	...	???	
$\vdots$			$\vdots$				$\ddots$

□

Conséquences de l'indécidabilité du problème de l'acceptation

L'indécidabilité de A_{MT} a la conséquence suivante pour l'informatique :

> Il n'existe pas de MT (*i.e.* programme) permettant de décider si n'importe quelle MT (*i.e.* autre programme) accepte n'importe quelle entrée.

Autrement dit, je ne peux pas écrire de programme :

- auquel je passe un programme de décision M et une entrée w,
- capable de dire comment M décide w,
- et qui fonctionne dans tous les cas.

Ce programme contiendra nécessairement au moins une contradiction logique qui l'empêchera de fonctionner correctement.

Constat :

- Les notions intuitives sont suffisantes pour construire des algorithmes simples.
- des notions formelles sont nécessaires pour démontrer que certains problèmes ne sont pas solvables par un ordinateur.

Attention :

- il est évidemment possible d'écrire un algorithme indécidable pour un problème décidable (exemple : prendre un algorithme décidable et le faire boucler dans certains cas ou l'algorithme rejette).
- si un problème est indécidable, cela ne signifie pas qu'il ne sera pas décidable sur un sous-ensemble du problème (= il est peut-être décidable **dans certains cas**, à condition d'avoir un algorithme **décidable** permettant de déterminer si l'on est sur ce sous-ensemble).
- ne pas confondre indécidable avec non récursivement énumérable (= aucun algorithme n'existe, tout court, pour résoudre le problème).

Premier exemple de langage non-énumérable

Donnons maintenant un exemple de langage non-énumérable.

Rappels :

- un langage co-énumérable si son complément est énumérable.
- **théorème :** un langage est décidable si et seulement si il est énumérable et co-énumérable.

Corollaire II.2

si un langage n'est pas décidable, alors soit le langage lui-même soit son complément n'est pas énumérable.

Démonstration:
proposition logique inverse du théorème dans les rappels ci-dessus. □

Théorème II.7
$\overline{A_{\text{MT}}}$ n'est pas énumérable.

$\overline{A_{\text{MT}}}$ est l'ensemble des couples $\langle M, w\rangle$ tels que $M(w)$ rejette ou boucle (par défaut, une expression $\langle M, w\rangle$ mal formée rejette).

Démonstration:
On a vu que A_{MT} est énumérable, mais qu'il n'est pas décidable.
Donc, d'après le corollaire précédent $\overline{A_{\text{MT}}}$ n'est nécessairement pas énumérable. □

1.5.3 Autres problèmes de décidabilité

Nous allons maintenant étudier d'autres problèmes de décidabilité.

Est-il possible de construire un programme permettant de décider :

- **le problème de l'arrêt :** si un autre programme s'arrête sur une entrée ?

  $$H_{\text{MT}} = \{\langle M, w\rangle |\ M \text{ est une MT qui s'arrête sur } w\}$$

  contient toutes les paires (MT,mot) telle que la MT s'arrête sur le mot.
- **le problème du vide :** si un autre programme rejette toutes ses entrées ?

  $$E_{\text{MT}} = \{\langle M\rangle |\ M \text{ est une MT et } \mathcal{L}(M) = \emptyset\}$$

  contient toutes les MTs qui n'acceptent aucun mot.
- **le problème de l'égalité :** si deux autres programmes prennent les mêmes décisions ?

  $$EQ_{\text{MT}} = \{\langle M_1, M_2\rangle |\ M_1 \text{ et } M_2 \text{ sont des MTs telles que } \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$$

  contient tous les couples de MTs qui acceptent le même langage.

Problème de l'arrêt

Théorème II.8
H_{MT} est récursivement énumérable.

Démonstration:
Reprendre la MT universelle U, et remplacer rejet par accepte, de façon à ce que U accepte tout le temps.
Ou de manière équivalente, en appelant M_H la MT qui énumère H_{MT} :

$M_H(\langle M, w\rangle) =$
```
SIMULER M(w)
// on arrive ici seulement si M ne boucle pas sur w
ACCEPTER
```

M_H accepte trivialement H_{MT}. □

Théorème II.9
H_{MT} est indécidable.

Démonstration:

Même type de preuve que pour A_{MT}.
On suppose qu'il existe une machine Halt($\langle M, w\rangle$) qui décide H_{MT}.
Considérons le programme D qui boucle si :

$D(\langle M\rangle) =$ **si** Halt($\langle M, M\rangle$) **alors boucler**() **sinon accepter**

$D(\langle M\rangle)$ = si Halt($\langle M, M\rangle$) alors boucler() sinon accepter

où *boucler*() est une MT qui boucle (par exemple, qui va à gauche puis à droite sur n'importe quel symbole, et recommence).
Alors, $D(\langle D\rangle)$ donne le résultat suivant :

- si Halt($\langle D, D\rangle$) = rejette, alors $D(\langle D\rangle)$ accepte
  $\Rightarrow$ Halt($\langle D, D\rangle$) devrait accepter.
- si Halt($\langle D, D\rangle$) = accepte, alors $D(\langle D\rangle)$ boucle
  $\Rightarrow$ Halt($\langle D, D\rangle$) devrait rejeter.
- si Halt($\langle D, D\rangle$) = boucle, alors il n'est pas décidable.

Dans les 3 cas, Halt($\langle M, M\rangle$) ne fonctionne pas, donc n'existe pas. □

Autre façon de faire cette démonstration :

Utilisons l'indécidabilité de A_{MT} pour prouver l'indécidabilité de H_{MT}.

Démonstration: ($2^{\text{ème}}$ version)

Supposons qu'il existe une MT R qui décide H_{MT}.
Utilisons R pour construire une MT S qui décide A_{MT} :

$S(\langle M, w\rangle) =$

```
// exécution du décideur de l'arrêt
EXÉCUTER R(⟨M, w⟩).
// R nous dit si M s'arrête sur w
// cas 1: R rejette = M(w) boucle.
SI R REJETTE ALORS REJETER
// cas 2: R accepte = M(w) ne boucle pas.
// on peut simuler M(w) sans crainte
SI R ACCEPTE ALORS SIMULER M(⟨w⟩).
DÉCIDER comme M
```

Clairement, si R décide H_{MT}, alors S décide A_{MT}.
Comme A_{MT} est indécidable, H_{MT} ne peut pas être décidable. □

Cette méthode est dite de **réduction** : consiste à réduire la résolution d'un problème inconnu à la résolution d'un problème déjà connu.

Problème du vide

On rappelle que : $E_{\text{MT}} = \{< M > \mid M \text{ est une MT et } \mathcal{L}(M) = \emptyset\}$

Théorème II.10

E_{MT} est indécidable.

Démonstration: (par réduction)

Idée de la démonstration : construire une MT à partir de $M(\langle w \rangle)$ qui réduit le problème A_{MT} au problème E_{MT}.

Soit la MT M_1 construite à partir d'une MT M et d'une entrée w comme suit :

$M_1(x) =$
```
SI x ≠ w
ALORS REJETER
SINON // exécuté seulement si x = w
  SIMULER M(⟨w⟩) // peut boucler
  DÉCIDER comme M // accepte ou rejette
```

Seul le mot w est accepté par M_1 si seulement si $w \in \mathcal{L}(M)$.
Donc, $\mathcal{L}(M_1) = \emptyset$ si $M(w)$ rejette ou boucle, et $\mathcal{L}(M_1) = \{w\}$ si $M(w)$ accepte.

Autrement dit : $\mathcal{L}(M_1) = \begin{cases} \emptyset & \text{si } w \notin \mathcal{L}(M) \\ \{w\} & \text{si } w \in \mathcal{L}(M) \end{cases}$

Supposons qu'il existe un décideur R de E_{MT}.
Construisons une MT S qui décide A_{MT} :

$S(\langle M, w \rangle) =$
```
CONSTRUIRE ⟨M1⟩ à partir de ⟨M⟩ ET w
EXÉCUTER R(⟨M1⟩)
SI R ACCEPTE
ALORS REJETER   // L(M1) = ∅ ⇒ M(w) rejette ou boucle
SINON ACCEPTER  // L(M1) = {w} ⇒ w ∈ L(M)
```

Le comportement de $R(\langle M_1 \rangle)$ est le suivant :

$$R(\langle M_1 \rangle) = \begin{cases} \text{accepte} & \text{si } \mathcal{L}(M_1) = \emptyset \Rightarrow w \notin \mathcal{L}(M) \\ \text{rejette} & \text{si } \mathcal{L}(M_1) = \{w\} \Rightarrow w \in \mathcal{L}(M) \end{cases}$$

Si R décide E_{MT}, alors S décide A_{MT}.
Comme A_{MT} est indécidable, E_{MT} n'est pas décidable. □

Remarque : ce problème n'est même pas énumérable (impossibilité d'assurer pour n'importe quel code M et par la seule analyse que l'état acceptant n'est jamais atteint). Évidemment, tester si aucun mot n'est accepté n'est pas envisageable non plus.

Problème de l'égalité

Soit $EQ_{\text{MT}} = \{< M_1, M_2 > \mid M_1 \text{ et } M_2 \text{ sont des MTs telles que } \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$

Théorème II.11

EQ_{MT} est indécidable.

Démonstration: (par réduction)

Montrons que la décidabilité de EQ_{MT} se réduit à la décidabilité de E_{MT}.
Supposons qu'il existe une MT R qui décide EQ_{MT}.
Soit $M_\emptyset$ une MT qui rejette toutes les entrées ($\mathcal{L}(M_\emptyset) = \emptyset$) :

$$M_\emptyset(w) = \boxed{\textbf{REJETER}}$$

Construisons une MT S qui décide E_{MT} (problème du vide) :

$$S(\langle M\rangle) = \begin{array}{|l} \textbf{CONSTRUIRE } \langle M_\emptyset\rangle \text{ // } \mathcal{L}(M_\emptyset) = \emptyset \\ \textbf{EXÉCUTER } R(\langle M, M_\emptyset\rangle) \\ \text{// décide si } \mathcal{L}(M) = \mathcal{L}(M_\emptyset) = \emptyset \\ \textbf{DÉCIDER comme } R \end{array}$$

Si R décide EQ_{MT}, alors S décide E_{MT}.
Comme E_{MT} est indécidable, EQ_{MT} n'est pas décidable. □

Remarques 12:

- EQ_{MT} n'est pas récursivement énumérable non plus.
- Plus on résout de problèmes (par réduction), plus on a de choix de réduction.

Exercices

Exercice 7 (Indécidabilité du langage $USELESS_{\text{MT}}$)**.** *Soit :*

$$USELESS_{MT} = \{\langle M, q\rangle \mid q \text{ est un état jamais atteint lors de l'exécution de la MT } M \text{ sur n'importe laquelle de ses entrées}\}$$

Montrer que $USELESS_{MT}$ est indécidable en utilisant une réduction de $USELESS_{MT}$ à E_{MT}.

Exercice 8 (Indécidabilité du langage $A_{\varepsilon-\text{MT}}$)**.** *Soit :*

$$A_{\varepsilon-MT} = \{\langle M\rangle \mid M \text{ est une MT qui accepte } \varepsilon\}$$

Montrer que $A_{\varepsilon-MT}$ est indécidable en utilisant une réduction de $A_{\varepsilon-MT}$ à A_{MT}.

Exercice 9. *Soit $T = \{\langle M\rangle | M$ est une machine de Turing qui accepte w^R chaque fois qu'elle accepte $w\}$ où w^R est l'inverse de la chaîne w. Montrer par réduction que T est indécidable.*

1.5.4 Théorème de Rice

Les problèmes de décidabilité étudiés d'une propriété non triviale d'un code ont tous eu la même conclusion : le problème est indécidable.

Nous voulons savoir s'il existe un sous-ensemble de $\mathcal{RE}$ qui est décidable.

Voyons comment définir un sous-ensemble de langages à partir d'une propriété.

Définition II.8

- Une **propriété** P est un ensemble particulier de langages inclus dans $\mathcal{RE}$ dont le langage possède caractéristique particulière.
  Exemple : $P = \{\langle M \rangle \mid \mathcal{L}(M)$ est un langage décidable$\}$.
- Un langage L **possède la propriété** P ssi $L \in P$.
- Une propriété P est dite **triviale** ssi
  - $P = \emptyset$ (la propriété n'est vérifiée par aucun langage de $\mathcal{RE}$),
  - $P = \mathcal{RE}$ (la propriété n'est vérifiée par tous les langages de $\mathcal{RE}$).
- Une propriété P est **dite non-triviale** ssi il existe :
  - au moins une MT qui a la propriété P.
  - au moins une MT qui a la propriété $\overline{P}$.

Est-il possible de décider un ensemble de langages ayant une certaine propriété ? (n'importe laquelle)

Cas des propriétés triviales

Théorème II.12 (décidabilité des propriétés triviales)

1. La propriété $P_\emptyset = \emptyset$ qui n'est vérifiée par aucun langage est décidable.
2. La propriété $P_{\text{all}} = \mathcal{RE}$ qui est vérifiée par tous les langages est décidable.

Démonstration:

1. Construisons une MT $M_\emptyset$ qui décide $P_\emptyset$.
   Soit la MT $M_\emptyset$:
   $M_\emptyset(w) =$ | **REJETER**
   Il est clair que $M_\emptyset$ reconnaît $P_\emptyset$ (ou $\mathcal{L}(M_\emptyset) = P_\emptyset$, donc si $\mathcal{L}(M) = \emptyset$) et s'arrête toujours (car ne dépend pas de l'entrée).
   Donc, $M_\emptyset$ décide $P_\emptyset$ et $P_\emptyset$ est décidable.
2. Construisons une MT M_{all} qui décide P_{all}.
   Soit la MT M_{all} :
   $M_{\text{all}}(w) =$ | **ACCEPTER**
   Il est clair que M_{all} reconnaît P_{all} (ou $\mathcal{L}(M_{\text{all}}) = P_{\text{all}}$). et s'arrête toujours (car ne dépend pas de l'entrée).
   Donc, M_{all} décide P_{all} et P_{all} est décidable.

□

Théorème II.13 (Rice)

Soit P une propriété non-triviale des langages $\mathcal{RE}$. Alors P est indécidable.

Démonstration: (par réduction)

Sans perte de généralité, supposons que $\emptyset \notin P$

nécessité de la démonstration - sinon effectuer la démonstration avec $\bar{P}$, énumérable lui-aussi si P est décidable).

Comme P n'est pas vide (sinon la propriété n'existe pas), il existe un langage $L \in P$.

Supposons que P est décidable, notons D_P son décideur.

On veut construire une réduction de A_{MT} vers P, afin de prouver que si P est décidable, alors A_{MT} est lui-aussi décidable.

Pour cela, on veut construire une MT M_w à partir de :

- un couple $\langle M, w\rangle$ quelconque dont on veut vérifier s'il appartient à A_{MT}.
- la MT M_L accepte le langage L (ci-dessus) qui vérifie la propriété P.

telle que $(M_w \in P) \Leftrightarrow (\langle M, w\rangle \in A_{\text{MT}})$.

Soit la machine suivante :

$M_w(\langle x\rangle) =$

```
SIMULER M(w)
SI M ACCEPTE
ALORS EXÉCUTER M_L(⟨x⟩)
        SI M_L ACCEPTE
        ALORS ACCEPTER
        SINON REJETER
SINON REJETER
```

Examinons le comportement de M_w :
si $\langle M, w\rangle \in A_{\text{MT}}$
alors M_w exécute $M_L(\langle x\rangle)$ $\quad \mathcal{L}(M_w) = L$, par construction $L \in P$
sinon rejeter $\quad \mathcal{L}(M_w) = \emptyset$, par définition de P, $\emptyset \notin P$

On a donc bien une MT construite à partir de $\langle M, w\rangle$ et de L qui vérifie la propriété P si et seulement si M accepte w.

Construisons alors un décideur D de A_{MT} :

$D(\langle M, w\rangle) =$

```
CONSTRUIRE ⟨M_w⟩ à partir de ⟨M⟩, w ET ⟨M_L⟩
SIMULER D_L(⟨M_w⟩)
DÉCIDER comme D_L
```

Notons la machine M_w ne pose aucune difficulté à être construite. C'est la concaténation de :

- une MT universelle qui s'exécute sur $M(w)$ (M et w variables dans D)
- la MT M_L (toujours la même)

et qu'il s'agit bien d'une réduction de A_{MT} à P.

Donc D est un décideur de A_{MT}.

Or, A_{MT} est indécidable. Donc D n'existe pas.

Comme l'existence de D est basé sur celle de D_P, D_P n'existe pas non plus.

En conséquence, P est indécidable.

Comme la démonstration ci-dessus peut-être effectuée pour toute propriété P (ou $\overline{P}$), on en conclut que toute propriété non triviale est indécidable.

□

Nous verrons plus loin comment ce type de méthodes de réduction peut être formalisée.

Conséquences du théorème de Rice

Exemple : Est-ce qu'un programme trie un tableau d'entiers ?

Attention : il ne s'agit pas de trier un tableau d'entiers, mais de vérifier qu'un programme effectue véritablement cette tâche.

A savoir, **s'il a la propriété** de trier un tableau d'entiers.

Le problème est bien défini : les spécifications exprimées sous forme d'objets mathématiques précis.

Prouver qu'un programme respecte une spécification ne devrait pas être plus difficile que de savoir si un triangle ressemble à un autre.

Le théorème de Rice montre que ce n'est PAS DU TOUT le cas : ce problème est indécidable.

Interprétation en termes de programme :

Il n'existe pas de programme permettant de décider si une propriété non triviale **quelconque** d'un autre programme est vraie.
Ceci est vrai pour **TOUTE** propriété non triviale.

Par exemple :

- savoir si un programme fait ce qu'on lui demande.
- savoir si un programme s'arrête (=ne boucle pas).
- savoir si deux programmes font la même chose.
- ...

Ceci constitue une limite importante **à toute tentative de vérification sur programme**.

Par contre :

Cela ne veut pas dire que le problème ne peut pas être résolu dans certains cas particuliers.

Mais qu'il ne le sera **jamais** par une machine de Turing sur la classe complète des langages de $\mathcal{RE}$.

Attention : à bien comprendre le théorème de Rice

Il dit : pour une MT M, les questions suivantes sur $\mathcal{L}(M)$ sont non-décidables.

Est-ce que $\mathcal{L}(M)$:

- est vide ?
- est fini ?
- est infini ?
- est décidable ?
- est régulier ?
- $= \Sigma^*$?
- = {Hello, World} ?
- contient un palindrome ?
- contient une chaîne de longueur paire ?

La vérification de toute propriété sur le langage engendré par une machine de Turing est indécidable.
Attention : ne sont pas concernées par le théorème de Rice :

- Les propriétés concernant l'encodage d'une MT
  Exemple : "$\langle M \rangle$ a un nombre pair d'états ?" est décidable.
- Les propriétés concernant les étapes intermédiaires de l'exécution d'une MT :
  — si on passe par un état particulier.
  — si on passe par une configuration particulière (état de la bande).
  — le nombre de transitions ...

  Exemple : "M passe par q_6 pour l'entrée vide"
  Non décidable, mais théorème de Rice ne s'applique pas.

Exercice 10. *Utiliser le théorème de Rice pour prouver l'indécidabilité de* $\{\langle M \rangle | M$ *est une machine de Turing telle que* $1011 \in \mathcal{L}(M)\}$.

Exercice 11. *Utiliser le théorème de Rice pour prouver l'indécidabilité de* $INFINITE_{TM} = \{\langle M \rangle | M$ *is a TM and* $\mathcal{L}(M)$ *is an infinite language*$\}$.

Exercice 12. *Soit* $T = \{\langle M \rangle | M$ *est une machine de Turing qui accepte* w^R *chaque fois qu'elle accepte* $w\}$ *où* w^R *est l'inverse de la chaîne* w. *Montrer avec le théorème de Rice que* T *est indécidable.*

1.6 Conclusion

Nous avons vu qu'il existait plusieurs types de problèmes qui sont toujours indécidables :

- Les problèmes non-énumérables,
- Les problèmes devant décider des propriétés sur des programmes (th. de Rice)

Ce ne sont pas les seuls.

- Le $10^{\text{ème}}$ problème de Hilbert,
- La pavage de Wang (impossibilité de pavage automatique d'un plan),
- Le calcul de la complexité de Kolmogorov (description minimale d'un bloc de données),
- ...

Tant qu'il n'a pas été démontré qu'un problème est décidable, on ne doit pas supposer qu'il l'est.

Rappel : pour démontrer qu'un problème est décidable, il suffit de montrer qu'il **ne boucle jamais**, et qu'il **s'arrête toujours**.

2 Calculabilité

2.1 Lien entre calculabilité et décidabilité

Rappelons les définitions d'une fonction calculable et partiellement calculable.

Définition II.9 (fonction (totalement) calculable)

Une MT M calcule totalement une fonction $f : \Sigma^* \to \Sigma^*$ si M

- commence avec l'entrée w sur sa bande.
- s'arrête pour tout w et avec seulement $f(w)$ écrit sur sa bande.

Définition II.10 (fonction partiellement calculable)

Une MT M calcule partiellement une fonction $f : \Sigma \to \Sigma \cup \perp$ si M :

- commence avec l'entrée w sur sa bande.
- si $f(w)$ est défini, alors M s'arrête avec seulement $f(w)$ écrit sur sa bande.
- si $f(w)$ est indéfini, alors M ne s'arrête pas.

On peut effectuer un lien direct entre calculabilité et complexité.

Exemple 17:

- **Formulation du problème :** Soit trois entiers x, y, z. Est-ce que $z = x \times y$?
- **Problème de décision :**
  $L = \{\langle x, y, z\rangle \mid x, y, z$ sont des entiers et $z = x \times y\}$.
  Ce problème est **décidable**.
- **Problème d'évaluation :**
  $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tel que $f(x, y) = x \times y$.
  Cette fonction est **calculable**.

Exemple 18:

- **Formulation du problème :** Soit une machine de Turing M. Est-ce que M a un nombre pair d'états ?
- **Problème de décision :**
  $L = \{\langle M\rangle \mid$ la MT M a un nombre pair d'états $\}$.
  Ce problème est **décidable**.
- **Problème d'évaluation :**
  $f : \{\langle M\rangle\} \to$ booléen tel que $f(\langle M\rangle) =$ est vrai si la MT $\langle M\rangle$ a un nombre d'états pairs et faux sinon.$\}$
  Cette fonction est **calculable**.

Exemple 19:

- **Formulation du problème :** Soit une machine de Turing M et une chaîne w. Est-ce que $M(w)$ s'arrête ?
- **Problème de décision :**
  $L = \{\langle M, w\rangle \mid$ la MT $M(w)$ s'arrête$\}$.
  Ce problème est **récursivement énumérable** (c'est le problème de l'arrêt).
- **Problème d'évaluation :**
  $f : \{\langle M, w\rangle\} \to \mathbb{N}$ tel que $f(\langle M, w\rangle) =$ le nombre de transitions au bout duquel M s'arrête sur w, sinon la réponse est indéfinie.
  Cette fonction est **partiellement calculable**.

EXEMPLE 20:

- **Formulation du problème :** Soit une machine de Turing M. Est-ce que M s'arrête sur toutes les entrées en moins de n transitions ?
- **Problème de décision :**
  $L = \{\langle M, n\rangle \mid$ la MT M s'arrête sur tous les mots de Σ^* en moins de n transitions $\}$.
  Ce problème est **indécidable**.
- **Problème d'évaluation :**
  $f : \{\langle M\rangle\} \rightarrow \mathbb{N}$ tel que $f(\langle M, w\rangle) =$ le nombre maximum de transitions au bout duquel M s'arrête sur n'importe quel mot de Σ^*, sinon la réponse est indéfinie.
  Cette fonction n'est pas **partiellement calculable** (on exclut les machines qui ne tiennent pas compte de la chaîne d'entrée).

2.2 Exemple de fonction non calculable

Donnons maintenant un exemple célèbre de fonction non calculable.
Le castor affairé est la fonction $\mathcal{S}(n)$ définie de la manière suivante.

- Soit l'ensemble de toutes les MTs à une bande avec $\Gamma = \{\sqcup, 1\}$, la bande infinie des deux côtées, initialisée avec $\sqcup$.
- Soit le sous-ensemble S_n des MTs M à n états tel que $M(\epsilon)$ s'arrête.

  L'ensemble S_n est fini (nombre fini d'états et de transitions possibles). Par définition, $M \in S_n$ s'exécute en un nombre fini de pas (il doit s'arrêter).
- Soit $\mathcal{S}(n)$ le nombre maximum de transitions exécutées par une machine de S_n sur l'entrée ϵ.

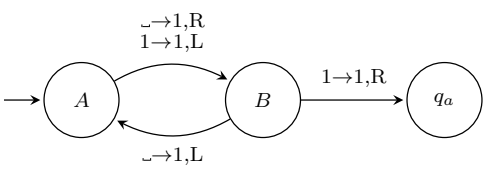
  De ce qui précède, $\mathcal{S}(n)$ est une fonction complètement définie.

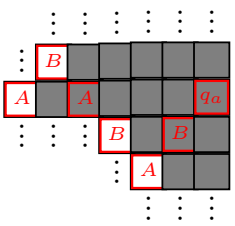
Cette fonction a les propriétés suivantes :

- Cette fonction est celle qui croît le plus rapidement

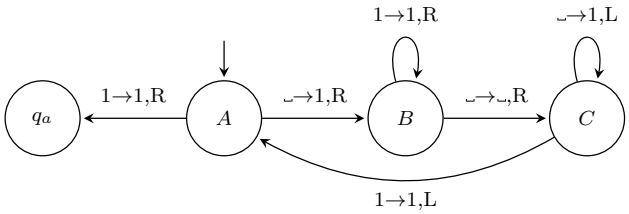
  $\mathcal{S}(1) = 1$, $\mathcal{S}(2) = 6$, $\mathcal{S}(3) = 14$, $\mathcal{S}(4) = 107$, $\mathcal{S}(5) = 47.176.870$, $\mathcal{S}(6) = 7.412.10^{36534}$, $\mathcal{S}(12) > 4096$ mis 166 fois à la puissance 4096.
- les premières MTs associées à $\mathcal{S}(n)$ peuvent être trouvées :

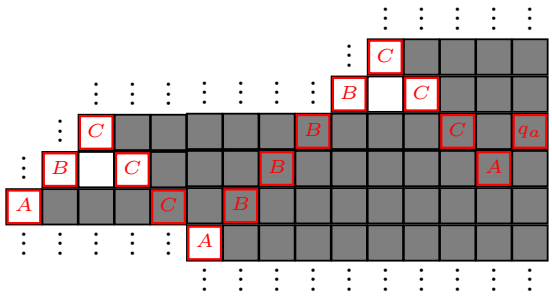
cas $n = 2$: $\mathcal{S}(2) = 6$



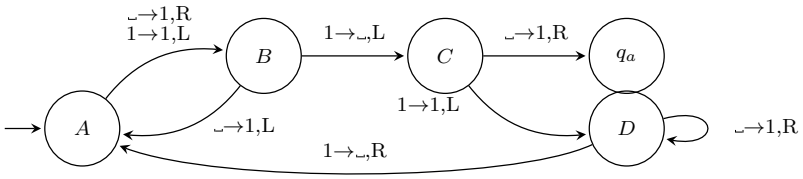


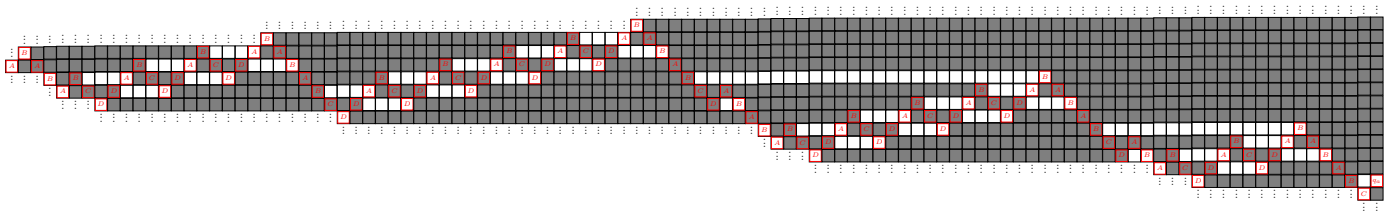
cas $n = 3$: $\mathcal{S}(3) = 14$





cas $n = 4$: $\mathcal{S}(4) = 107$





Montrons que cette fonction n'est pas calculable.

Théorème II.14

la fonction $\mathcal{S}(n)$ du castor affairé n'est pas calculable.

Démonstration:

Supposons que $\mathcal{S}(n)$ soit calculable.
Comme $\mathcal{S}(n)$ est le nombre maximum de transitions pour qu'une MT M à n états s'arrête, M ne s'arrêtera pas si elle dépasse ce nombre de transition.
On peut alors construire la MT S qui vérifie si un programme M s'arrête sur ε :

$S(\langle M \rangle)$ = calculer le nombre n d'états de M.
calculer $\mathcal{S}(n)$.
exécuter au plus $\mathcal{S}(n) + 1$ transitions de M.
si M s'est arrêté, alors accepter.
si M ne s'est pas arrêté alors rejeter.

Si un tel programme existait, H_{MT} serait décidable.
Comme H_{MT} n'est pas décidable, $\mathcal{S}(n)$ n'est pas calculable. □

Paradoxe :

On peut calculer $\mathcal{S}(n)$ pour certaines valeurs de n, mais $\mathcal{S}(n)$ n'est pas calculable ?

- Tout segment fini d'une suite de fonctions non calculables est calculable :
  pour tout n, il existe un algorithme qui calcule la suite $\mathcal{S}_n = \{\mathcal{S}(0), \mathcal{S}(1), \ldots, \mathcal{S}(n)\}$, ou pour une valeur de n particulière.
- mais il n'existe aucun algorithme qui calcule la suite entière de $\mathcal{S}(n)$ (*i.e.* $\mathcal{S}(n), \forall n$).

Toujours pas ? Cela tient à la définition d'un algorithme :

- Un algorithme est une méthode permettant de résoudre une **classe** de problème.
- $\mathcal{S}(n)$ ne se calcule que pour des valeurs de n déterminées.
- Comme il n'existe pas d'algorithme calculant $\mathcal{S}(n)$ pour tout n, cette fonction n'est donc pas calculable.

Cela revient à la différence entre être capable de résoudre un problème dans un cas particulier et dans le cas général.

3 Réductibilité

Lors de plusieurs démonstrations de décidabilité, nous avons utilisé le principe de réductibilité.

Ce modèle de démonstration est schématiquement le suivant :

- on connaît la propriété d'un certain langage A,
- on voudrait savoir si un autre langage B possède la même propriété.
- on construit une fonction dite "de réduction" capable de transformer tout mot de B en un mot de A.
- on en déduit que B a la même propriété que A.

Donnons-en maintenant une définition formelle.

3.1 Définition de la réductibilité

Définition II.11 (application réductive)

Soit A et B deux langages. Soit f une fonction **calculable**.

f est une application réductive (ou réduction) de A vers B s'il existe une fonction calculable $f : \Sigma^* \to \Sigma^*$ telle que pour tout w,

$$(w \in A) \Leftrightarrow (f(w) \in B)$$

Notation : $A \leq_m B$

Rappel : on a : $(u \Rightarrow v) \Leftrightarrow (\overline{v} \Rightarrow \overline{u})$. Donc :

$((w \in A) \Rightarrow (f(w) \in B)) \Rightarrow ((f(w) \notin B) \Rightarrow (w \notin A))$

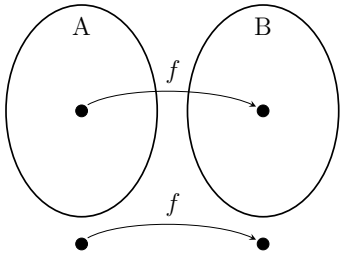
$((f(w) \in B) \Rightarrow (w \in A)) \Rightarrow ((w \notin A) \Rightarrow (f(w) \notin B))$

D'où on déduit $(w \notin A) \Leftrightarrow (f(w) \notin B)$

Remarque 13:

Une réduction de A vers B permet de convertir une question sur l'appartenance à A en une question sur l'appartenance à B.
A savoir :

- si $f(w) \in B$ alors $w \in A$.
- si $f(w) \notin B$ alors $w \notin A$.



Interprétation intuitive :

Comprendre $A \leq_m B$ comme :

- tout problème sur A peut être transformé en un problème sur B.

- tout problème de A n'est pas plus compliqué qu'un problème de B.
- tout problème de B est au moins aussi difficile qu'un problème de A

Donnons maintenant des exemples d'applications.

3.2 Lien avec la décidabilité

Théorème II.15 (décidabilité par réduction)

si $A \leq_m B$ et B est décidable alors A est décidable.

Démonstration:

Soit M_B une MT qui décide pour B, et f une réduction de A vers B.
Alors, la MT M_A suivante décide pour A :

$M_A(\langle w \rangle) =$

```
CALCULER ⟨v⟩ = f(⟨w⟩)      // ⇒ f(w) ∈ B
EXÉCUTER M_B(⟨v⟩)          // décide si f(w) ∈ B (ne boucle pas)
DÉCIDER comme M_B          // (f(w) ∈ B) ⇔ (w ∈ A)
```

f s'arrête toujours (fonction calculable), tout comme M_B (décideur de B). Donc M_A ne boucle jamais.
Ce code décide si $f(w) \in B$.
Or, $(f(w) \in B) \Leftrightarrow (w \in A)$.
Donc, si $f(w) \in B$ alors $w \in A$. Si $f(w) \notin B$, alors $w \notin A$.
Ainsi, M_A est un décideur de A. D'où A décidable. □

Corollaire II.3 (indécidabilité par réduction)

Si $A \leq_m B$ et A est indécidable alors B est indécidable.

Démonstration:

Supposons que B soit décidable. On se retrouve alors dans le cadre du théorème précédent : $A \leq_m B$ et B décidable implique que A est décidable. Or, A n'est pas décidable, donc B non plus. □

Lemme II.5

$A \leq_m B$ implique $\bar{A} \leq_m \bar{B}$

Démonstration:

Simple contraposée logique (rappel : $A \Rightarrow B \equiv \neg B \Rightarrow \neg A$). Donc $(w \in A) \Leftrightarrow (f(w) \in B)$ est équivalent à $(w \notin A) \Leftrightarrow (f(w) \notin B)$ □

Remarque 14:

utile pour démontrer les propriétés sur les ensembles complémentaires.

3.3 Application à l'énumérabilité

Théorème II.16 (énumérabilité par réduction)

Si $A \leq_m B$ et B est récursivement énumérable alors A est récursivement énumérable.

Démonstration:

La démonstration est quasiment identique à la même proposition sur la décidabilité.
Soit M_B la MT qui énumère B.
Soit f la fonction calculable associée à la réduction de A à B.
Alors, la MT suivante énumère A :

$$M_A(\langle w\rangle) = \begin{array}{|ll|l} \textsc{calculer} & \langle v\rangle = f(\langle w\rangle) & // \Rightarrow f(w) \in B \\ \textsc{exécuter} & M_B(\langle v\rangle) & // \text{ accepte , rejette ou boucle} \\ \textsc{décider} & \text{comme } M_B & // (f(w) \in B) \Leftrightarrow (w \in A) \end{array}$$

$M_B(f(\langle w\rangle))$ peut faire 3 choses :

- **accepter :** comme $(f(w) \in B) \Leftrightarrow (w \in A)$. Donc M_A accepte quand $w \in A$.
- **rejeter :** comme $(f(w) \notin B) \Leftrightarrow (w \notin A)$. Donc M_A rejette quand $w \notin A$.
- **boucler :** dans ce cas $f(w) \notin B$, donc $(w \notin A)$. M_A boucle aussi, mais signifie que $w \notin A$, ce qui est bien le comportement attendu.

En conséquence, M_A énumère A, et A est énumérable. □

Corollaire II.4

si $A \leq_m B$ alors

- si A n'est pas $\mathcal{RE}$ alors B n'est pas $\mathcal{RE}$.
- si A n'est pas $co\mathcal{RE}$ alors B n'est pas $co\mathcal{RE}$.

Démonstration:

- Si B était $\mathcal{RE}$, le théorème ci-dessus impliquerait que A est $\mathcal{RE}$. Contradiction.
- Même démonstration avec les ensembles complémentaires, et grâce au lemme impliquant $\bar{A} \leq_m \bar{B}$.

□

3.4 Exercices

Exercice 13. *Montrer que A est récursivement énumérable si $A \leq_m A_{TM}$.*

Exercice 14. *Donnez un exemple de langage indécidable B tel que $B \leq_m \overline{B}$.*

Exercice 15 (Indécidabilité de $\text{USELESS}_{\text{MT}}$ par réduction formelle)**.** *Soit :*

$$USELESS_{MT} = \{\langle M, q\rangle \mid q \text{ est un état jamais atteint lors de l'exécution de la MT } M \text{ sur n'importe laquelle de ses entrées}\}$$

*Montrer que $USELESS_{MT}$ est indécidable en utilisant une **réduction formelle** de $USELESS_{MT}$ à E_{MT}.*

Exercice 16 (Indécidabilité de $A_{\varepsilon-\text{MT}}$ par réduction formelle)**.** *Soit :*

$$A_{\varepsilon-MT} = \{\langle M\rangle \mid M \text{ est une MT qui accepte } \varepsilon\}$$

*Montrer que $A_{\varepsilon-MT}$ est indécidable en utilisant une **réduction formelle** de $A_{\varepsilon-MT}$ à A_{MT}.*

Exercice 17. *Considérons le problème de déterminer si une machine de Turing M sur une entrée w tente jamais de déplacer sa tête vers la gauche lorsque sa tête est sur la première cellule de la bande. Formulez ce problème sous forme de langage et montrez par réduction qu'il est indécidable.*

Exercice 18. *Considérons le problème consistant à déterminer si une machine de Turing à une bande n'écrit à aucun moment un symbole vide sur un symbole non vide au cours de son exécution sur une chaîne d'entrée quelconque. Formulez ce problème sous la forme d'un langage et montrez par réduction qu'il est indécidable.*

3.5 Fonction à exécution contrôlée

Une classe intéressante de fonctions :
fonction qui modifie la machine qu'elle exécute.

Exemple 21: fonction à exécution contrôlée La machine F suivante termine l'exécution de la machine M si M essaie d'aller à gauche alors que le pointeur de lecture est déjà au début de la bande.

$F(\langle M, w\rangle) =$

```
exécution pas à pas de M sur w
  trouver la transition suivante ( état ,symbole, direction )
  SI on est en début de bande
  ALORS
    SI direction = L
    ALORS ÉCRIRE ϵ
      ARRÊTER
  appliquer la transition
  SI état = q_a ALORS ARRÊTER
```

3.6 Exécution contrôlée

Réduction par exécution contrôlée :

Technique de réduction qui consiste à construire la MT pas à pas.

Prenons par exemple, $L_\infty = \{\langle M\rangle \mid \mathcal{L}(M) \text{ est infini}\}$.
Par le théorème de Rice, on sait que $L_\infty \notin \mathcal{R}$.
Montrons aussi que $L_\infty \notin \mathcal{RE}$ en utilisant une réduction de $\overline{H_{\text{MT}}}$.
Rappel : $\overline{H_{\text{MT}}}$ = complémentaire de H_{MT} = ensemble des MTs qui ne s'arrêtent pas.

Lemme II.6

$\overline{H_{\text{MT}}} \notin \mathcal{RE}$

Démonstration:

On sait que $H_{\text{MT}} \notin \mathcal{R}$, mais $H_{\text{MT}} \in \mathcal{RE}$. Donc, nécessairement $H_{\text{MT}} \notin co\mathcal{RE}$, d'où $\overline{H_{\text{MT}}} \notin \mathcal{RE}$. □

Théorème II.17

$L_\infty \notin \mathcal{RE}$

DÉMONSTRATION:

La ligne de preuve est la suivante :

- on sait que $\overline{H_{\text{MT}}}$ n'est pas récursivement énumérable.
- si on trouve une réduction telle que : $\overline{H_{\text{MT}}} \leq_m L_\infty$
- on en déduit que L_∞ n'est pas récursivement énumérable.

Cherchons une réduction f : $\overline{H_{\text{MT}}} \to L_\infty$ telle que : $\langle M, w\rangle \mapsto \langle M_0\rangle$

- si M s'arrête sur w alors $\mathcal{L}(M_0)$ est fini.
- si M ne s'arrête pas sur w alors $\mathcal{L}(M_0)$ est infini.

On rappelle que $|w|$ retourne la longueur du mot w.

Soit la MT M_0 suivante construite à partir de $\langle M, w\rangle$ et une MT universelle U.
Si $M(w)$ ne boucle pas, notons $k_{M,w}$ le nombre de transitions nécessaire à $M(w)$ pour s'arrêter.

$M_0(\langle v\rangle) =$

```
EXÉCUTER |v| transition de U(⟨M, w⟩)
SI U s'est arrêtée
ALORS REJETER    // M(w) s'arrête, rejette si |v| > k_{M,w}
SINON ACCEPTER   // M(w) ne s'est pas arrêtée ou boucle, accepte tout
```

Examinons le comportement de $\mathcal{L}(M_0)$:

- si M **boucle** sur w, alors M_0 accepte v en entier, donc $\mathcal{L}(M_0) = \Sigma^*$, et $\langle M_0\rangle \in L_\infty$.
- si M **s'arrête** sur w après k transitions, alors M_0 accepte tous les mots $v \in \Sigma^*$ de longueur inférieure ou égale à k donc $\mathcal{L}(M_0)$ est fini, et $\langle M_0\rangle \notin L_\infty$.

Pour résumer, le comportement de $L(M_0)$:

- $\langle M, w\rangle \in \overline{H_{\text{MT}}} \Rightarrow \langle M_0\rangle \in L_\infty$.
- $\langle M, w\rangle \notin \overline{H_{\text{MT}}} \Rightarrow \langle M_0\rangle \notin L_\infty$.

D'où $(\langle M, w\rangle \in \overline{H_{\text{MT}}}) \Leftrightarrow (\langle M_0\rangle \in L_\infty)$
Soit la fonction de réduction f définie par :

$f(\langle M, w\rangle) =$

```
CONSTRUIRE ⟨M_0⟩ à partir de ⟨M, w⟩ ET U
RETOURNER ⟨M_0⟩
```

f est bien une fonction calculable (MTU exécutant partiellement $M(w)$).
Donc, f est une réduction de $\overline{H_{\text{MT}}}$ dans L_∞ et $\overline{H_{\text{MT}}} \leq_m L_\infty$. Or $\overline{H_{\text{MT}}} \notin \mathcal{RE}$, ce qui implique $L_\infty \notin \mathcal{RE}$ (cf corollaire réductibilité).

□

REMARQUES 15: technique utilisée pour des preuves comme

- tester l'existence de certains objets.
- est-ce qu'une MT bornée linéairement accepte le langage vide ?
- est-ce qu'une GLC génère Σ^* ?

4 Théorème de récursion

4.1 Principe

Nous abordons dans cette partie la possibilité de créer des machines capables de construire des répliques d'elles-mêmes.

Une première approche naïve serait de dire :

- si une machine A construit une machine B,
  alors A doit être plus complexe que B.
- Mais une machine ne peut être plus complexe qu'elle-même.
- Donc, aucune machine ne peut se construire elle-même, et donc l'auto-reproduction est impossible.

Les assertions ci-dessus sont fausses : créer une machine qui se reproduit elle-même est possible.

Ce fait est affirmé par le théorème de récursion, qui nécessite le lemme suivant.

Lemme II.7

Il est existe une fonction calculable $q : \Sigma^* \to \Sigma^*$ telle que pour toute chaine $w \in \Sigma^*$, $q(w)$ est la description de la machine de Turing P_w qui écrit w sur la bande et s'arrête.

DÉMONSTRATION:

La construction de P_w et l'écriture de $\langle P_w \rangle$ peuvent être trivialement effectuée.

Soit la fonction suivante qui remplace le contenu de la bande par w est clairement calculable :

$$P_w(\langle x \rangle) = \begin{array}{|l} \textsc{effacer } \langle x \rangle \text{ de la bande} \\ \textsc{écrire } \langle w \rangle \end{array}$$

On considère la fonction calculable Q suivante qui calcule $q(w)$:

$$Q(\langle w \rangle) = \begin{array}{|l} \textsc{construire } \langle P_w \rangle \\ \textsc{écrire } \langle P_w \rangle \end{array}$$

La MT Q s'arrête donc bien dans tous les cas avec $\langle P_w \rangle$ écrit sur sa bande.
Donc, q existe et est bien une fonction calculable.

□

On cherche maintenant à écrire une MT `SELF` capable de se répliquer (= de s'exécuter et de produire en fin d'exécution son propre code sur la bande).

Pour ce faire, `SELF` doit nécessairement être constituée d'au moins deux parties A et B (à savoir $\langle \texttt{SELF} \rangle = \langle AB \rangle$) telles que :

- la partie A qui écrit une description de B :
  on utilise la machine $P_{\langle B \rangle}$ (= remplace le contenu de la bande par $\langle B \rangle$).
  A nécessite donc d'avoir la description de B : on ne peut donc pas décrire A avant d'avoir construit B.
- la partie B qui écrit une description de A :
  On ne peut pas utiliser $q(\langle A \rangle)$ car A est lui-même défini en fonction de B (= définition circulaire = transgression logique).
  Autre stratégie : B calcule A à partir de la sortie que A devrait produire.

Si B obtient $\langle B\rangle$ (= sa propre description), il peut calculer $\langle A\rangle = q(\langle B\rangle)$.
mais comment obtenir $\langle B\rangle$?
Puisque A (après son exécution) écrit $\langle B\rangle$, il suffit donc pour B de lire la bande.
Donc, B lit $\langle B\rangle$ sur la bande, calcule $q(\langle B\rangle) = A$, combine A et B en une seule machine, et écrit la description $\langle AB\rangle$ sur la bande.

En résumé :

A = $Q(\langle B\rangle)$ = écrit $\langle B\rangle$ sur la bande
$B(\langle M\rangle)$ =

```
                                          // avec M = B
CALCULER Q(⟨M⟩)                           // = Q(⟨B⟩)
concaténer le résultat avec ⟨M⟩ // = Q(⟨B⟩)⟨B⟩ = ⟨A⟩⟨B⟩
ÉCRIRE Q(⟨M⟩)⟨M⟩
```

SELF = AB

Le code de SELF est $\langle \texttt{SELF}\rangle = \langle A\rangle\langle B\rangle = Q(\langle B\rangle)\langle B\rangle$.

L'exécution de SELF donne donc :

1. l'exécution de A ecrit $< B >$ sur la bande.
2. l'exécution de B lit $< B >$ sur la bande, calcule $q(\langle B\rangle)$, puis y réécrit la concaténation $Q(\langle B\rangle)\langle B\rangle$.

A la fin de l'exécution, le contenu de la bande est $Q(\langle B\rangle)\langle B\rangle$,
soit le code de SELF.

Exercice 19. *Donnez un exemple d'un programme dans un langage de programmation réel (ou une approximation raisonnable de celui-ci) qui s'affiche lui-même.*

Le théorème de récursion fournit la possibilité d'implémenter l'auto-référence *this* à toute MT (= à tout langage de programmation).

Grâce à ce théorème, tout programme peut faire référence à sa propre description.

Théorème II.18 (de récursion)

Soit la MT T qui calcule la fonction $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.
$\exists$ MT R qui calcule la fonction $r : \Sigma^* \to \Sigma^*$ telle que $\forall w \in \Sigma^*$, $r(w) = t(\langle R\rangle, w)$.

Autrement dit, on voudrait construire une MT R qui utilise sa propre description lors de son exécution $R(w)$ sur une entrée w.

Pour cela, on peut construire une MT T qui effectuerai le même travail mais en recevant la description de R en paramètre (*i.e.* avec $T(\langle R\rangle, w)$).

Dire qu'il existe R telle que $r(w) = t(\langle R\rangle, w)$ signifie qu'il est possible :

- de créer une machine qui fait référence à sa description,
- d'intégrer sa propre description au code d'une MT.

Voyons maintenant comment.
On reprend l'idée de SELF, afin qu'un programme puisse obtenir sa propre description et l'utiliser pour des traitements.

Démonstration:

On construit une MT R en trois partie A, B et T (*i.e.* $\langle R\rangle = \langle ABT\rangle$).

$A = P_{\langle BT\rangle}$ est décrite par $q(\langle BT\rangle)$.
Mais, pour conserver l'entrée w, on modifie q telle que $P_{\langle BT\rangle}$ écrive sa sortie après l'entrée w déjà présente sur la bande (*i.e.* après l'exécution de A, la bande contient $w\langle BT\rangle$).

Ensuite, B lit la bande et applique q à son contenu donnant A.
Puis, B combine A, B et T en une MT R dont la description est $\langle ABT\rangle$.
Cette description est de nouveau combinée avec w, afin d'écrire $\langle R, w\rangle$ sur la bande.

Enfin, le contrôle est passé à T, ce qui permet bien d'obtenir la fonction recherchée. □

Pour résumer, à la fin de l'exécution T a bien pour entrée $\langle R, w\rangle$:

exécution		A	B	T
contenu de la bande	w	$w\langle BT\rangle$	$\langle\langle ABT\rangle, w\rangle = \langle R, w\rangle$	

Quel est l'intérêt du théorème de récursion ?

- permet d'inclure au code d'une MT M l'instruction «obtenir la propre description de M» (avec le théorème de récursion).

  `SELF` = | obtenir la propre description de `SELF`
  écrire $\langle$`SELF`$\rangle$
- une fois cette description obtenue, M peut calculer son nombre d'états, simuler $\langle M\rangle$, ...
- certains virus informatique utilisent ce principe pour se dupliquer.
- certaine démonstration d'indécidabilité ou de non récursive-énumérabilité peuvent aussi être effectués avec le théorème de récursion.

4.2 Problème de l'arrêt

Le théorème de récursion permet de fournir une nouvelle preuve du problème de l'acceptation.

Théorème II.19 (par le théorème de récursion)

$A_{\mathtt{MT}}$ est indécidable.

Démonstration:

Par l'absurde : supposons qu'il existe une MT T qui décide $A_{\mathtt{MT}}$ (*i.e.* $T(\langle M, w\rangle)$ accepte si $M(w)$ accepte et rejette sinon).

Soit la MT M suivante :

$M(\langle w\rangle) =$
```
OBTENIR ⟨M⟩                // th. de récursion
EXÉCUTER T(⟨M, w⟩)
DÉCIDER l'opposé de T
```

Comme M décide l'inverse de ce que T indique décider, T ne peut pas exister et $A_{\mathtt{MT}}$ est indécidable. □

4.3 Problème de la machine de Turing minimale

Définition II.12 (longueur de la description d'une MT)

La longueur de la description d'une MT M est le nombre de symbole de la chaîne de caractère décrivant $\langle M\rangle$.

Définition II.13 (MT minimale)

Une MT M est dite minimale s'il n'existe aucune autre MT équivalente (*i.e.* reconnaissant le même langage) et dont la longueur de la description est plus petite.

On définit le langage MIN_{MT} le langage constitué de l'ensemble des MTs minimales.

$$\textit{i.e. } \text{MIN}_{\text{MT}} = \{\langle M\rangle \mid M \text{ est une MT minimale}\}$$

Théorème II.20

MIN_{MT} n'est pas récursivement énumérable.

Démonstration:

Supposons qu'il existe un énumérateur E qui énumère MIN_{MT}.

Soit la MT C suivante :

$C(\langle w\rangle) =$
```
OBTENIR ⟨C⟩        // th. de récursion
EXÉCUTER l'enumérateur E jusqu'à trouver < D >
            telle que |⟨D⟩| > |⟨C⟩|
SIMULER D(w)
DÉCIDER comme D
```

Comme MIN_{MT} est infini, il existe nécessairement une machine D avec une description plus longue que C. Donc, l'énumération s'arrête toujours.

Or C simule D (et produit donc le résultat de D). Donc C et D sont équivalents. Mais C a une description plus petite que D (D choisi avec une description plus grande), donc D n'est pas minimale.

Or, D fait partie des machines énumérées par E. Contradiction : l'énumérateur E n'existe pas, et MIN_{MT} n'est pas récursivement énumérable. □

Exercice 20. *Montrer que tout sous-ensemble infini de MIN_{TM} n'est pas récursivement énumérable.*

5 Décidabilité des théories logiques

5.1 Idée

On aimerait pouvoir écrire un programme qui nous permettrait de tester des expressions mathématiques telles que :

1. $\forall q, \exists p, \forall x, y\,[p > q \wedge (x, y > 1 \Rightarrow xy \neq p)]$
   existence d'un nombre infini de nombres premiers (preuve Euclide, -300 ans)
2. $\forall a, b, c, n\,[(a, b, c > 0 \wedge n > 2) \Rightarrow a^n + b^n \neq c^n]$
   dernier théorème de Fermat (conjecture Pierre de Fermat 1937, preuve Andrew Wiles, 1993)
3. $\forall q \exists p \forall x, y\,[p > q \wedge (x, y > 1 \Rightarrow (xy \neq p \wedge xy \neq p + 2))]$
   conjecture des nombres premiers jumeaux (non prouvé)

En terme d'informatique théorique, cela revient à :

- définir un alphabet contenant les symboles utilisés pour définir ces expressions,
- définir un langage pour lequel l'ensemble des propositions vraies font parties du langage,
- déterminer si ce langage est décidable.

5.2 Enoncé

On définit l'alphabet du langage $\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1, \ldots, R_k\}$.
composé des opérateurs booleens ($\wedge, \vee, \neg$), des parenthèses, des quantificateurs, des variables ($x_1 = x$, $x_2 = xx$, $x_3 = xxx$, ...), des relations ($R_1, \ldots, R_k$, par exemple $R_1 \equiv <$).
Définitions :

- Une formule atomique est une chaine de la forme : $R_i(x_1, \ldots, x_j)$.
- Une chaine ϕ est une formule si ϕ est :
  - — une formule atomique,
  - — de la forme $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg\phi_1$ où ϕ_1 et ϕ_2 sont des formules,
  - — de la forme $\exists x_i\,[\phi_1]$ ou $\forall x_i\,[\phi_1]$ où ϕ_1 est une formule.
- Une variable libre est une variable sans quantificateur.
- Une formule sans variable libre est un énoncé (ou formule close).

Exemple 22: Enoncé $\forall x_1, \exists x_2, \exists x_3\,[R_1(x_1) \wedge R_2(x_1, x_2, x_3)]$ est un énoncé.

Ce type d'énoncé logique est du premier ordre (= quantification des variables), par opposition aux logiques d'ordre supérieur où les relations peuvent être quantifiées.

5.3 Univers et modèle

Pour définir complètement un langage, il est encore nécessaire d'ajouter :

- un univers sur lequel les variables peuvent prendre leurs valeurs (exemple : $\mathbb{N}$),
- un modèle $\mathcal{M}$ est un $k+1$-uple $(U, P_1, \ldots, P_k)$ où U est l'univers, et P_i l'assignation de R_i.

Exemple 23: Soit l'énoncé $\phi = \forall x_1 \forall x_2\,[R_1(x_1, x_2) \vee R_1(x_2, x_1)]$.
Soit le modèle $\mathcal{M}_1 = (\mathbb{N}, \leq)$ (*i.e.* $x_i \in \mathbb{N}$ et $R_1 = \leq$). ϕ est vrai dans $\mathcal{M}_1$.
Par contre, ϕ est faux dans $(\mathbb{N}, <)$ dans le cas où $x_1 = x_2$.

Exemple 24: On définit la relation $\texttt{PLUS}(a, b, c) = \texttt{vrai}$ si $a + b = c$.
Soit $\psi = \forall x_1, \exists x_2\,[R_1(x_1, x_1, x_2)] = \forall x_1, \exists x_2\,[x_1 + x_1 = x_2]$.
Soit le modèle $\mathcal{M}_2 = (\mathbb{R}, \texttt{PLUS})$.
ψ est vrai (signifie que x peut être divisé par 2) dans $\mathcal{M}_2$.
ψ est faux dans $(\mathbb{N}, \texttt{PLUS})$.

5.4 Théorie

Si $\mathcal{M}$ est un modèle, on appelle la théorie de $\mathcal{M}$ (notée $\mathrm{Th}(\mathcal{M})$) l'ensemble des énoncés vrais dans le langage du modèle.

En 1928, Hilbert posa le «Entscheidungsproblem», à savoir s'il existait un algorithme capable de déterminer si tout énoncé issu d'une logique du premier ordre était décidable.

Les résultats suivants ont été obtenus (non démontrés ici, voir Sipser) :

Théorème II.21
Th($\mathbb{N}, +$) est décidable.

Théorème II.22
Th($\mathbb{N}, +, \times$) est récursivement énumérable.

Théorème II.23 (Church, Turing, 1936)
Th($\mathbb{N}, +, \times$) est indécidable.

Autrement dit, il est impossible de démontrer si certains énoncés de Th($\mathbb{N}, +, \times$) sont vrais ou faux.

5.5 Théorème de Gödel

Une preuve formelle π d'un énoncé ϕ est une suite d'énoncé $S_1, S_2, \ldots, S_n$ où $S_n = \phi$, et où chaque S_i est obtenu à partir de S_{i-1}, d'axiomes et de règles d'implication.

On attend d'une preuve les deux propriétés suivantes :

- la validité de la preuve d'un énoncé peut être vérifiée par une machine (*i.e.* $\{\langle \phi, \pi \rangle \mid \pi$ est une preuve de $\phi\}$ est décidable),
- le système de preuve est consistant : si un énoncé est démontrable (*i.e.* a une preuve formelle), alors il est vrai.

Pour un système de preuve ayant ces deux propriétés, on a le théorème suivant :

Théorème II.24 (Gödel)
Certains énoncés vrais de Th($\mathbb{N}, +, \times$) ne sont pas démontrables.

On dit qu'une théorie est complète si l'ensemble des énoncés vrais sont démontrables.

Théorème II.25 (d'incomplétude, Gödel, 1931)
Pour tout système raisonnable formalisant la notion de preuve en théorie des nombres, certains énoncés vrais ne sont pas démontrables.

Le théorème de Gödel affirme donc qu'aucune théorie peut être à la fois consistante et complète.

Ce théorème a mis fin à 50 ans d'effort et de tentatives pour trouver un ensemble d'axiomes permettant de construire un système logique complet utilisable de manière générale par les mathématiques.

Néanmoins, certaines logiques restreintes du premier ordre sont décidables.

Résumé

- un langage est décidable si il existe une MT qui le reconnaisse et s'arrête toujours.
- si un langage n'est pas décidable, alors lui ou son complémentaire n'est pas énumérable.
- Le problème d'acceptation, le problème de l'arrêt est indécidable, le problème de déterminer si un langage est vide … sont indécidables.
- (Rice) toute propriété non triviale sur un langage est indécidable.

- une exécution contrôlée consiste à simuler une MT sur une MT universelle afin de modifier la simulation si certaines conditions sont remplies.
- une fonction calculable est une MT qui s'arrête avec le résultat de la fonction écrit sur la bande.
- une réduction consiste à transformer un problème sur un langage A en un problème sur un langage B (plus complexe et connu),
- le théorème de récursion rend possible à une MT de faire référence à son propre code lors de son exécution.
- Les théories logiques sont en général récursivement énumérables mais pas décidables.
- Pour tout système raisonnable formalisant la notion de preuve en théorie des nombres, certains énoncés vrais ne sont pas démontrables (Gödel).