Chapitre II

Réductibilité

1 Décidabilité

1.1 Lien entre $\mathcal{R}$ et $\mathcal{RE}$

Rappel : 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.
  = s'il existe une MT qui le reconnaît.
  Note : on dit aussi «Turing-reconnaissable».
- **décidable** $\stackrel{\Delta}{=}$ s'il existe une MT M telle que M qui accepte tout $w \in L$ et s'arrête dans tous les autres cas. *i.e.* si $w \notin L$, alors M rejette et ne boucle jamais.
  Note : on dit aussi «Turing-décidable».

Donc :

- la décidabilité est une notion plus forte que l'énumérabilité.
  $\Rightarrow$ si un langage L est décidable, alors il est énumérable.
- si L est décidable, alors $\bar{L}$ aussi.
  $\Rightarrow$ il suffit de remplacer q_{accept} par q_{reject} et vice-versa.

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.

Avec ces notations, les constatations deviennent :

- si un langage L est décidable, alors il est énumérable : $\mathcal{R} \subseteq \mathcal{RE}$
- si L est décidable, alors $\bar{L}$ aussi : $\mathcal{R} \subseteq co\mathcal{RE}$
- donc $\mathcal{R} \subseteq \mathcal{RE} \cap co\mathcal{RE}$

Théorème 7 : 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_1 une MT qui accepte L.
  Soit M_2 une MT qui accepte $\bar{L}$.

Construisons une MT qui exécute M_1 et M_2 en parallèle :
— si M_1 accepte, alors M accepte.
— si M_2 accepte, alors M rejette.
Comment exécuter M_1 et M_2 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 M décide L.
Toute chaîne w est soit dans L, soit dans $\bar{L}$
$\Rightarrow$ soit M_1, soit M_2 accepte w.
$\Rightarrow$ M s'arrête chaque fois que M_1 ou M_2 accepte.
$\Rightarrow$ M s'arrête pour tout w
De plus par construction, M accepte les chaînes de L et rejette les chaînes de $\bar{L}$.
Donc M décide L, et L est décidable. □

Théorème 8 : (le même reformulé)
Un langage est décidable si et seulement si il est à la fois énumérable et co-énumérable.

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

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 possède une racine entière".

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.

Clairement, la formulation de Hilbert induit que :

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

Cas des polynômes à une variable (=univarié) :

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

On peut trouver une MT M qui reconnaît D :

Entrée : polynôme $p(x)$
Algorithme :
pour $x = 0, 1, -1, 2, -2, \ldots$
évaluer $p(x)$
si $p(x) = 0$ alors **accepter** sinon **continuer**.

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 : donc aussi énumérable.

Ces langages sont énumérables, mais sont-il décidables ?

Cas des polynômes à une variable

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

Soit la MT M modifiée D :

Entrée : polynôme $p(x)$

Algorithme :

soit $A = [-|kc_{\max}/c_1|, +|kc_{\max}/c_1|]$
pour tous les x entiers dans A
évaluer $p(x)$
si $p(x) = 0$ alors **accepter** sinon **continuer**.
rejeter

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

En conséquence :

— 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.

Le problème de décidabilité est l'un des problèmes les plus importants en informatique théorique.

Les ordinateurs :

— les ordinateurs sont **fondamentalement limités**.
beaucoup de problèmes ne sont pas décidables.
— ne sont **pas omnipotents**
encore plus ne sont pas récursivement énumérables.

2 Problème de décidabilité

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.

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 des savoir si un triangle ressemble à un autre.

Ce n'est PAS DU TOUT le cas.

Qu'en est-il ?

Soient :

$$\begin{aligned} A_{\text{ADF}} &= \{< M, w > \mid M \text{ est un ADF qui accepte } w\} \\ A_{\text{ANF}} &= \{< M, w > \mid M \text{ est un ANF qui accepte } w\} \\ A_{\text{GLC}} &= \{< M, w > \mid M \text{ est une GLC qui accepte } w\} \\ A_{\text{MT}} &= \{< M, w > \mid M \text{ est une MT qui accepte } w\} \end{aligned}$$

Théorème 9 : décidabilité des modèles

1. A_{ADF}, A_{ANF}, A_{GLC} sont décidables.
2. A_{MT} est récursivement énumérable.
3. A_{MT} est indécidable.

Démonstration:

1. construire une MT qui simule le modèle, et qui s'arrête (cf TD).
2. déjà démontré : car une MT peut être simulée sur une MT universelle.
3. avant d'en apporter la preuve, besoin d'un complément théorique.

□

2.1 Notions d'infini

2.1.1 Comment comparer la taille de deux ensembles A et B ?

Cas des ensembles finis

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.

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 $\forall b \in \mathbb{N}$ 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.

Cas des ensembles infinis

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

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 !

Cas des ensembles dénombrables

Définition d'un 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 :

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.

□

2.1.2 Tous les ensembles sont-ils dénombrables ?

Ensemble des rationnels positifs $\mathbb{Q}^+$:

Rappel : $\mathbb{Q}^+ = \left\{ m/n \mid (m,n) \in \mathbb{N}^{*2} \right\}$

Troisième affirmation :

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

Plus difficile à démontrer.

Première approche :

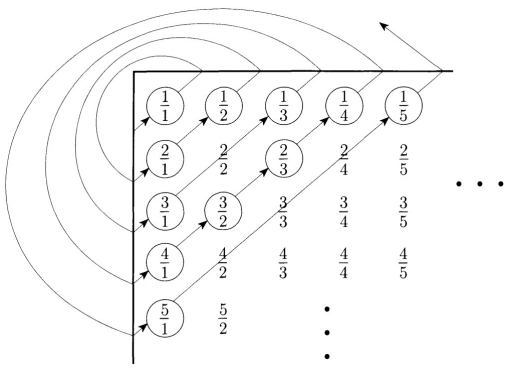
On considère $\mathbb{Q}$ comme un tableau à 2 dimensions.

Commençons par compter la première ligne ...

Zut, elle est infinie !

Seconde approche :

Si on compte le long des diagonales et en sautant les doubles comme ceci :



On va bien tout compter.

Tous les ensembles infinis sont-ils dénombrables ?

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

Théorème 10 : $\mathbb{R}$ n'est pas dénombrable.
Notation : $\aleph_1 = \#\mathbb{R}$

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
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 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 démontrable. □

2.1.3 Applications aux machines de Turing

Lemme 11 : si Σ un ensemble fini, alors Σ^* est dénombrable.

Démonstration:
Notons Σ_n les chaînes de longueurs n de Σ^*. Il est clair que $\Sigma^* = \bigcup_i \Sigma_i$.
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 12 : L'ensemble des MTs est dénombrable.

Démonstration:

Chaque MT M est codée comme une chaîne $< M >$.

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 13 : $\mathbb{B}$ 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 $\mathbb{B}$. □

Lemme 14 : L'ensemble des langages $\mathcal{L}$ sur un alphabet Σ fini n'est pas dénombrable.

Démonstration:

Soit $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$ 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 15:

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 générés par une MT. □

Preuve d'existence de ces langages (sans en exhiber un seul).

2.2 Problème de l'acceptation

Montrons maintenant que A_{MT} est indécidable.

Démonstration: (par l'absurde)

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 :

— appeler H pour déterminer ce qu'une MT M fait lorsque son entrée est sa propre description $< M >$ (*i.e.* évaluer $H(\langle M, \langle M\rangle\rangle)$).

— renvoyer alors l'opposé de ce que décide H,
i.e. rejeter si H accepte $\langle M, \langle M\rangle\rangle$ et accepter si H rejette $\langle M, \langle M\rangle\rangle$.

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

Donc, D peut s'écrire :

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

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

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

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

Donc, ni D, ni H n'existent.

Si H n'existe pas, A_{MT} n'est pas décidable. □

Conclusion :

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

Cette preuve est en fait une forme de 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	
⋮			⋮		⋱		
D	reject	reject	accept	accept	…	???	
⋮			⋮				⋱

2.3 Langage non-énumérable

Donnons maintenant un exemple de langage non-énumérable.

RAPPELS:
- **Définition :** un langage co-énumérable si son complément est un langage énumérable.
- **Théorème :** un langage est décidable si et seulement si il est à la fois énumérable et co-énumérable.

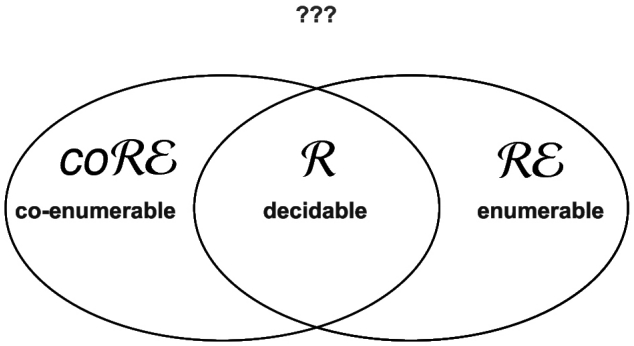
COROLLAIRE 16 : si un langage n'est pas décidable, soit le langage lui-même et son complément n'est pas énumérable.

DÉMONSTRATION:
proposition logique inverse du théorème. □

THÉORÈME 17 : $\overline{A_{\text{MT}}}$ n'est pas énumérable.

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

Les MTs s'organisent donc comme suit :



Y-a-t-il d'autres ensembles ?
— Tous les ensembles $\mathcal{RE}$, $co\mathcal{RE}$ et $\mathcal{R}$ sont dénombrables (car ils ne contiennent que des MTs).
— Il manque tous les ensembles non énumérables.

2.4 Problème de l'arrêt

$A_{\text{MT}} = \{< M, w > \mid M \text{ est une MT qui accepte } w\}$ est indécidable.
⇒ Il n'existe pas de programme générique qui s'arrête toujours permettant de tester si une machine fait ce qu'on lui demande.

Soit maintenant :

$$H_{\text{MT}} = \{< M, w > \mid M \text{ est une MT qui s'arrête sur } w\}$$

Même question : ce langage est-il décidable ?

A savoir : Existe-t-il un programme générique qui s'arrête toujours, permettant de tester si une machine s'arrête pour une entrée donnée ?

Ces deux problèmes se ressemblent.

La preuve est similaire.

THÉORÈME 18 : 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. Cette machine modifiée accepte H_{MT}. □

Théorème 19 : H_{MT} est indécidable.

Démonstration:
Même type de preuve.

On suppose qu'il existe une machine Halt($< M, w >$) qui reconnaît H_{MT}. Considerons le programme :

$D(M)$ = si Halt($< M, M >$) alors boucler() sinon accepter

où *boucler*() est défini par *boucler*() $\stackrel{\Delta}{=}$ *boucler*() (appel récursif).

Alors, $D(D)$ donne le résultat suivant :
— si Halt(D, D) = rejette, alors $D(D)$ accepte $\Rightarrow$ Halt(D, D) devrait accepter.
— si Halt(D, D) = accepte, alors $D(D)$ boucle $\Rightarrow$ Halt(D, D) devrait rejeter.
— si Halt(D, D) = 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} :

$S(\langle M, w\rangle)$ = exécuter $R(\langle M, w\rangle)$.
si R rejette alors rejeter.
si R accepte alors simuler $M(\langle w\rangle)$.
si M accepte alors accepter.
si M rejette alors rejeter.

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.

2.5 Problème du vide

Soit $E_{\text{MT}} = \{< M > \mid M \text{ est une MT et } L(M) = \emptyset\}$

Théorème 20 : 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 :

$M_1(\langle x\rangle)$ = si $x \neq w$ alors rejeter.
si $x = w$ alors exécuter $M(\langle w\rangle)$.
si M accepte alors accepter.

Cette MT M_1 n'accepte que w, et seulement si $w \in \mathcal{L}(M)$.

Le langage reconnu par M_1 est par conséquent :

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

Cette machine produit l'ensemble vide si $w \notin \mathcal{L}(M)$ (cas où M rejette ou boucle) et $\{w\} \in \mathcal{L}(M)$ (cas où M accepte).

Supposons qu'il existe une MT R qui décide E_{MT}.

Construisons une MT S qui décide A_{MT} :

$S(\langle M, w\rangle)$ = utiliser M et w pour construire M_1.
Exécuter $R(\langle M_1\rangle)$
si R accepte alors rejeter.
si R rejette alors renvoyer $M(\langle w\rangle)$.

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. □

2.6 Problème de l'égalité

Soit

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

Théorème 21 : 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.

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

$S(\langle M\rangle)$ = exécuter $R(\langle M, M_\emptyset\rangle)$.
si R accepte alors accepter.
si R rejette alors rejeter.

Si R décide EQ_{MT}, alors S décide E_{MT}.

Comme E_{MT} est indécidable, EQ_{MT} n'est pas décidable. □

Note : Plus on résout de problèmes (par réduction), plus on a de choix de réduction.

3 Réductibilité

Implique l'utilisation de deux problèmes A et B

Propriété recherchée :

si A peut se réduire à B

alors toute solution pour B peut être utilisée pour trouver une solution pour A.

A savoir :

si A est réductible à B alors

— on a les relations suivantes :

1. A n'est pas plus compliqué que B

2. B n'est pas plus simple que A

— si B est décidable alors A est décidable (par ①)
— si A est indécidable alors B est indécidable (par ②))

Nous formalisons cette notion, et démontrons ces propriétés.

3.1 Définition de la réductibilité

Définition 9 : application réductive

Soit A et B deux langages.

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

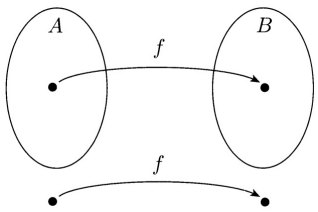
$$w \in A \Leftrightarrow f(w) \in B$$

La fonction f est appelée une réduction de A vers B.

Notation : $A \leq_m B$

Remarque 2 :

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



3.2 Lien avec la décidabilité

Théorème 22 : décidabilité par réduction
si $A \leq_m B$ et B est décidable alors A est décidable.

Démonstration:
Soit M une MT qui décide pour B, et f une réduction de A vers B. Alors, la MT N suivante décide pour A :

$N(\langle w \rangle)$ = calculer $f(w)$
exécuter $M(\langle f(w) \rangle)$
retourner la sortie de M

Si $w \in A$, alors $f(w) \in B$ car f est une réduction de A à B. Par hypothèse, $M(\langle u \rangle)$ accepte et s'arrête pour tout $u \in B$. Or, pour tout $w \in A$, $f(w) \in B$, donc $M(\langle f(w) \rangle)$ accepte et s'arrête aussi dans tous les cas (M décide B et f calculable). Donc, $N(\langle w \rangle)$ est une MT qui décide pour A. Donc, A est décidable. □

Corollaire 23 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 24 : $A \leq_m B$ implique $\bar{A} \leq_m \bar{B}$

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

Note : utile pour démontrer les propriétés sur les ensembles complémentaires.

3.3 Lien avec l'énumérabilité

Théorème 25 : énumérabilité par réduction
Si $A \leq_m B$ et B est récursivement énumérable alors A est récursivement énumérable.

Démonstration:
Comme $B \in \mathcal{RE}$, il existe une MT M qui accepte B. Comme f calculable et $A \leq_m B$, alors $M(\langle f(.) \rangle)$ reconnaît A. □

Corollaire 26:
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 $\bar{A} \leq_m \bar{B}$. □

4 Décidabilité d'une propriété

Regardons maintenant s'il existe des sous-ensembles de $\mathcal{RE}$ qui sont décidables.

Définition 10 :
- Une **propriété** P est un ensemble particulier de langages inclus dans $\mathcal{RE}$.
  Exemple : $P = \{L \mid L \text{ est un langage décidable}\}$.
- Un langage L **possède la propriété** P ssi $L \in P$.
- Une propriété P est dite **triviale** ssi $P = \emptyset$ ou $P = \mathcal{RE}$.
- Une propriété P est **dite non-triviale** ssi :
  - il existe une MT M qui reconnaît P.
  - il existe une MT $\overline{M}$ qui reconnaît $\overline{P}$.

Théorème 27 : décidabilité des propriétés triviales
1. La propriété $P_\emptyset$ qui ne reconnaît aucun langage est décidable.
   $P_\emptyset = \emptyset$
2. La propriété P_{all} qui reconnaît tous les langages est décidable.
   $P_{\text{all}} = \{L \mid L \text{ est un langage reconnaissable}\}$

Démonstration:
1. Construisons une MT $M_\emptyset$ qui reconnaît $P_\emptyset$.
   Soit $M_\emptyset(\langle M, w\rangle)$ = rejeter
   Il est clair que :
   - $M_\emptyset$ reconnaît $P_\emptyset$ (ou $\mathcal{L}(M_\emptyset) = P_\emptyset$).
   - $M_\emptyset$ s'arrête toujours (ne dépend pas de l'entrée).

   Donc, $M_\emptyset$ décide $P_\emptyset$. Donc, $P_\emptyset$ est décidable.

2. Construisons une MT M_{all} qui reconnaît P_{all}.
   Soit $M_{\text{all}}(\langle M, w\rangle)$ = accepter
   Il est clair que :
   - M_{all} reconnaît P_{all} (ou $\mathcal{L}(M_{\text{all}}) = P_{\text{all}}$).
   - M_{all} s'arrête toujours (dépend pas de l'entrée).

   Donc, M_{all} décide P_{all}. Donc, P_{all} est décidable.

□

4.1 Théorème de Rice

Nous abordons maintenant un théorème important qui montre que toute propriété non triviale est indécidable.

Théorème 28 : de Rice
Soit P une propriété non-triviale des langages $\mathcal{RE}$. Alors P est indécidable.

Démonstration:
par réduction

Sans perte de généralité, supposons que $\emptyset \notin P$ (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 $L \in P$.

Soit f la fonction de réduction $\mathcal{RE} \times \Sigma^* \to P$ qui construit une MT M_w telle que $\mathcal{L}(M_w) \in P$ à partir de :
- une MT M quelconque de $\mathcal{RE}$
- par hypothèse, une MT M_L accepte le langage L

de la manière suivante :

Donc, soit f la fonction qui construit la machine de Turing suivante à partir de M et de M_L :

$M_w(\langle x \rangle)$ **=** simuler $M(\langle w \rangle)$
si M accepte
alors simuler $M_L(\langle x \rangle)$
si M_L accepte **alors** accepter **sinon** rejeter

Notons que f est une fonction calculable : c'est la concaténation d'une MT universelle et de M_L.

Examinons le comportement de M_w :
si $< M, w > \in A_{\text{MT}}$
alors M_w exécute $M_L(\langle y \rangle)$; par conséquent, $\mathcal{L}(M_w) = L$ et $\mathcal{L}(M_w) \in P$
sinon ($< M, w > \notin A_{\text{MT}}$) et $\mathcal{L}(M_w) = \emptyset$; par conséquent, $\mathcal{L}(M_w) \notin P$.

En conséquence, f est bien une réduction de A_{MT} à P ($A_{\text{MT}} \leq_m P$).

Or, s'il existe un décideur pour P, l'existence de cette réduction implique que A_{MT} serait également décidable. Comme A_{MT} est indécidable, un tel décideur n'existe pas, et P est indécidable. □

4.2 Conséquences

Interprétation en termes de programme :

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

Ceci est vrai pour **TOUTE** propriété non triviale.

A savoir :

— savoir si un programme fait ce qu'on lui demande.
— savoir si un programme s'arrête.
— savoir si deux programmes font la même chose.
— ...

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 :

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

Comment $\mathcal{L}(M) = \emptyset$ pas décidable ?
Mais $P_\emptyset$ était décidable ?

$\mathcal{L}(M) = \emptyset$ est reconnu par la propriété P_{vide} qui reconnaît le langage vide.

Et non par $P_\emptyset$ la propriété qui ne reconnaît rien.

$P_{\text{vide}} = \{L \mid L = \emptyset\}$ n'est pas décidable.

Ne sont pas concernés par le théorème de Rice :

— Les propriétés concernant l'encodage d'une MT
Exemple : "$< M >$ 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 non utilisable.

4.3 Exécution contrôlée

Réduction par exécution controlée :

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

Prenons par exemple, $L_\infty = \{\langle M\rangle \mid L(M) \text{ est infini}\}$.
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 29 : $\overline{H_{\text{MT}}} \notin \mathcal{RE}$

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

Démonstration:
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.

soit la fonction de réduction $\langle M_0\rangle = f(\langle M, w\rangle)$ définie par :
$M_0(\langle y\rangle)$ **=** exécuter $|y|$ transitions de la MT universelle $U(\langle M, w\rangle)$
au bout de ces $|y|$ transitions,
si U **ne s'est pas arrêtée** alors **accepter**.
si U **s'est arrêtée** alors **rejeter**.

Notons que M_0 s'arrête sur toutes les entrées (seules les $|y|$ premières transitions sont exécutées).

Examinons le comportement de $\mathcal{L}(M_0)$:
— si M **ne s'arrête pas** sur w, alors M_0 accepte y 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 seulement si $|y| < k$ donc $\mathcal{L}(M_0)$ est fini, et $\langle M_0\rangle \notin L_\infty$.

Pour résumer, le comportement de $L(M_0)$:
— $\langle M, w\rangle \in \overline{H_{\text{MT}}} \Rightarrow \langle M_0\rangle \in L_\infty$.
— $\langle M, w\rangle \notin \overline{H_{\text{MT}}} \Rightarrow \langle M_0\rangle \notin L_\infty$.
Il est clair que f est une fonction calculable. Donc, f est une réduction de $\overline{H_{\text{MT}}}$ dans L_∞ et $L_\infty \leq_m \overline{H_{\text{MT}}}$. Or $\overline{H_{\text{MT}}} \notin \mathcal{RE}$, ce qui implique $L_\infty \notin \mathcal{RE}$ (cf corollaire réductibilité). □

REMARQUES: 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^* ?

5 Fonction calculable

5.1 Définition

Une MT peut être utilisée pour calculer des fonctions.

DÉFINITION 11 : 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.

NOTES:

- marche aussi pour les fonctions avec plus d'une variable.
  Encoder les paramètres sur la chaîne d'entrée intercalée de séparateurs.
  Chaque paramètre peut représenter n'importe quel objet.
- même remarque pour la sortie.
- une bande particulière peut être utilisée pour la sortie.

Notation : on note $\perp$ la "sortie" de la MT si elle ne s'arrête pas.

DÉFINITION 12 : 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.

NOTE : Les fonctions calculables sont également appelées fonctions (totalement ou partiellement) récursives.

5.2 Exemple

Affirmations :

— Toutes les fonctions arithmétiques classiques sur les entiers sont totalement calculables : addition, soustraction, multiplication, division (quotient, reste), ...

— Toutes les fonctions non-arithmétiques sont également totalement calculables : trigonométrique, logarithmique, ...
en utilisant les séries de Taylor (ou d'autres techniques d'approximation) à une précision spécifiée.

5.3 Addition binaire

implémentation de l'addition binaire sur une MT à 2 bandes, infinie des deux côtés, initialisées avec des blancs, et avec un pointeur LSR (S =stay (ne pas bouger)).

— q_{copy} : pour copier la première chaîne sur la deuxième bande (état de départ).
— q_{find} : pour se déplacer sur le LSB de la 1ère bande.
— q_{add} : pour faire la somme de 2 chiffres binaires sans retenue (0+0, 0+1, 1+0).
— q_{carry} : pour faire la somme de 2 chiffres binaires avec retenue tant qu'il y en a.
— q_{end} : état final.

q_{copy}	0	␣	q_{copy}	␣	0	R	R
	1	␣	q_{copy}	␣	1	R	R
	#	␣	q_{find}	␣	␣	R	S
q_{find}	0	␣	q_{find}	0	␣	R	S
	1	␣	q_{find}	1	␣	R	S
	␣	␣	q_{add}	␣	␣	L	L
q_{add}	0	0	q_{add}	0	␣	L	L
	0	␣	q_{add}	0	␣	L	L
	␣	0	q_{add}	0	␣	L	L
	1	0	q_{add}	1	␣	L	L
	1	␣	q_{add}	1	␣	L	L
	0	1	q_{add}	1	␣	L	L
	␣	1	q_{add}	1	␣	L	L
	1	1	q_{carry}	0	␣	L	L
	␣	␣	q_{end}	␣	␣	R	R
q_{carry}	0	0	q_{add}	1	␣	L	L
	␣	0	q_{add}	1	␣	L	L
	0	␣	q_{add}	1	␣	L	L
	0	1	q_{carry}	0	␣	L	L
	␣	1	q_{carry}	0	␣	L	L
	1	0	q_{carry}	0	␣	L	L
	1	␣	q_{carry}	0	␣	L	L
	1	1	q_{carry}	1	␣	L	L
	␣	␣	q_{end}	1	␣	S	R

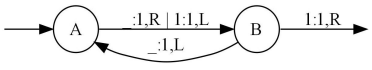
5.4 Le castor affairé

— Soit l'ensemble de toutes les MTs à une bande avec $\Sigma = \{␣, 1\}$, la bande infinie des deux côtées, initialisée avec ␣.
— 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.

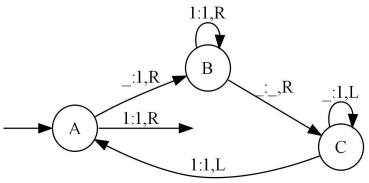
— Cette fonction est celle qui croît le plus rapidement
$\mathcal{S}(1) = 1$, $\mathcal{S}(2) = 6$, $\mathcal{S}(3) = 14$, $\mathcal{S}(4) = 107$, $\mathcal{S}(5) = 47.176.870$, $\mathcal{S}(6) = 7.412.10^{36534}$, $\mathcal{S}(12) > 4096$ mis 166 fois à la puissance 4096.
— les premières MTs associées à $\mathcal{S}(n)$ peuvent être trouvées :

1RB = écrire 1, aller à droite, passer dans l'état B.

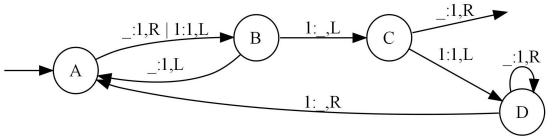
	A	B
␣	1RB	1LA
1	1LB	1RH



	A	B	C
␣	1RB	␣RC	1LC
1	1RH	1RB	1LA



	A	B	C	D
␣	1RB	1LA	1RH	1RD
1	1LB	␣LC	1LD	␣RA



Théorème 31 : 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 transitions 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 :

$S(\langle M\rangle)$ = 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.

5.5 Fonction à exécution contrôlée

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

Exemple 6: fonction à exécution contrôlée

Machine F qui termine l'exécution de la machine M qu'elle exécute 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 l'entrée w de la façon suivante :

1. à la position du pointeur, identifier la transition à prendre par M.
2. vérifier que la transition n'essaye pas d'aller à gauche du début de la bande.
3. si c'est la cas, alors écrire ϵ (donc au début de la bande) et s'arrêter.
4. sinon exécuter la transition.
5. continuer au 1.

6 Théorème de récursion

6.1 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 32 :
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 :

Preuve constructive : on considère la MT Q suivante qui calcule $q(w)$:

$Q(\langle w\rangle)$ = Construction de la machine P_w suivante :
$P_w(\langle x\rangle)$ = Effacer l'entrée x sur la bande.
Ecrire w sur la bande.
Accepter.
Ecrire $\langle P_w\rangle$.

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

La MT P_w s'arrête donc bien dans tous les cas avec $\langle P_w\rangle$ écrit sur sa bande.

Donc, P_w 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 $\langle$`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 :

$A =$ $P_{\langle B\rangle}$ = écrit B sur la bande

$B =$ | // M = *contenu de la bande* = B
| calculer $q(< M >)$ $= q(\langle B\rangle)$
| concaténer le résultat avec $\langle M\rangle$ $= q(\langle B\rangle)\langle B\rangle = \langle A\rangle\langle B\rangle$
| écrire $q(\langle M\rangle)\langle M\rangle$ sur la bande

SELF = AB

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

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.
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 33 : 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).

$\texttt{SELF} =$ | obtenir la propre description de SELF
| écrire $\langle \texttt{SELF} \rangle$

— 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.

6.2 Problème de l'arrêt

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

Théorème 34 : par le théorème de récursion

$A_{\texttt{TM}}$ est indécidable.

Démonstration :

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

Soit la MT M suivante :

$M(\langle w \rangle) =$ | obtenir la propre description de M.
| exécuter $T(< M >, w)$.
| retourner l'opposé de la décision de T (= accepte si T rejette, rejette si T accepte).

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

6.3 Problème de la machine de Turing minimale

Définition 13 : 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 14 : 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 $\text{MIN}_{\texttt{TM}}$ le langage constitué de l'ensemble des MTs minimales.

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

Théorème 35 :

$\text{MIN}_{\texttt{TM}}$ n'est pas récursivement énumérable.

Démonstration :

Supposons qu'il existe une MT E qui énumère $\text{MIN}_{\texttt{TM}}$.

Soit la MT C suivante :

$C(\langle w \rangle) =$ | obtenir la propre description de C.
exécuter l'enumérateur E jusqu'à trouver une machine D avec une description plus grande que C.
Simuler $D(w)$.

Comme MIN_{TM} 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_{TM} n'est pas récursivement énumérable. □

7 Décidabilité des théories logiques

7.1 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.

7.2 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 7: 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.

7.3 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 8:
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 9:
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 $(\mathbb{N}, \texttt{PLUS})$.

7.4 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 36 :
$\mathrm{Th}(\mathbb{N}, +)$ est décidable.

Théorème 37 :
$\mathrm{Th}(\mathbb{N}, +, \times)$ est récursivement énumérable.

Théorème 38 : Church, Turing, 1936
$\mathrm{Th}(\mathbb{N}, +, \times)$ est indécidable.

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

7.5 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 39 : Gödel

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

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

Théorème 40 : d'incomplétude, Gödel, 1931

Pour tout système raisonable 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 raisonable formalisant la notion de preuve en théorie des nombres, certains énoncés vrais ne sont pas démontrables (Gödel).