Bibliographie

1. *Introduction to Automata Theory, Languages, and Computation*, J. Hopcroft, D. Ullman, Addison Wesley, 1979.
2. *Calculabilité et Décidabilité*, J.-M. Autebert, Masson, 1992.
3. *Codes and Cryptography*, D. Welsh, Clarendon Press, Oxford,1988.
4. *Randomized Algorithms*, R. Motwani and P. Raghavan, Cambridge University Press, 1995.

Calculabilité

1. Machines de Turing
2. Machines RAM
3. Indécidabilité

Introduction

Formalisation de différentes notions.

- algorithme
- machine
- décidabilité
- calculabilité

Existence de fonctions non calculables par algorithme.

Thèse de Church

Tous les modèles inventés pour définir la calculabilité sont équivalents.

MACHINES DE TURING

Machines de Turing

a_1	a_2	$\ldots$	a_i	$\ldots$	a_n	#	#	#	$\ldots$

contrôle fini

$M = (Q,\Sigma,\Gamma,\delta,q_0,\#,F)$

- Q états en nombre fini
- $q_0 \in Q$ état initial
- $F \subset Q$ états terminaux
- Γ symboles de la bande
- $\Sigma \subset \Gamma$ symboles d'entrée
- $\# \in \Gamma$, n'appartient pas à Σ. (blanc)
- δ fonction de transition, non nécessairement définie partout, de
  $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L,R\}$

Exemple 1 $(p,a) \longrightarrow (q,b,R)$

Machine de Turing

Description instantanée

a_1	a_2	$\ldots$	a_i	$\ldots$	a_n	$\#$	$\#$	$\#$	$\ldots$

état q

$\alpha_1 = a_1a_2\ldots a_{i-1}$ et $\alpha_2 = a_i\ldots a_n$. On peut avoir α_1 et/ou α_2 égaux à ϵ. Ils peuvent contenir le symbole $\#$.
On dit que $a_1\ldots a_{i-1}qa_i\ldots a_n$ est une *description instantanée.*

- Si $(q,a_i) \longrightarrow (p,Y,L)$ et $i = 1$, il n'y a pas de mouvement. On reste en $q\alpha_2$.
- Idem avec $i \neq 1$. On a le mouvement :

  $$a_1\ldots a_{i-1}qa_i\ldots a_n \vdash_M a_1\ldots a_{i-2}pa_{i-1}Ya_{i+1}\ldots a_n$$
- Si $(q,a_i) \longrightarrow (p,Y,R)$ on a le mouvement :

  $$a_1\ldots a_{i-1}qa_i\ldots a_n \vdash_M a_1\ldots a_{i-1}Ypa_{i+1}\ldots a_n$$

Acceptation

Langage accepté par M:

$$\{w \in \Sigma^* \mid q_0 w \vdash_M^* \alpha_1 p \alpha_2, p \in F, \alpha_i \in \Gamma^*\}$$

On peut supposer, sans perte de généralité, que la MT s'arrête lorsqu'un mot est accepté. Pour les mots non acceptés, on peut avoir arrêt ou pas.

Exemple 2 Soit $L = \{0^n 1^n \mid n \geq 1\}$.

Soit

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
- $\Sigma = \{0,1\}$. $\Gamma = \{0,1,X,Y,\#\}$.
- $F = \{q_4\}$

Les actions possibles sont :

$$
\begin{array}{rcl}
(q_0,0) & \longrightarrow & (q_1,X,R) \\
(q_1,0) & \longrightarrow & (q_1,0,R) \\
(q_1,1) & \longrightarrow & (q_2,Y,L) \\
(q_2,0) & \longrightarrow & (q_2,0,L) \\
(q_2,X) & \longrightarrow & (q_0,X,R) \\
(q_1,Y) & \longrightarrow & (q_1,Y,R) \\
(q_2,Y) & \longrightarrow & (q_2,Y,L) \\
(q_0,Y) & \longrightarrow & (q_3,Y,R) \\
(q_3,Y) & \longrightarrow & (q_3,Y,R) \\
(q_3,\#) & \longrightarrow & (q_4,\#,R)
\end{array}
$$

Automate "représentant" une machine de Turing :

Langages et fonctions calculables

Un langage est dit *récursivement énumérable* s'il est accepté par une machine de Turing. Remarque : Si un mot w n'est pas dans le langage il est possible que M ne s'arrête pas.

L est récursivement énumérable :

$$w \longrightarrow \boxed{\text{M}} \begin{cases} \longrightarrow \text{oui} & w \in L \\ \longrightarrow \text{non} & w \notin L \\ \longrightarrow \text{ne s'arrête pas} & w \notin L \end{cases}$$

Un langage est dit *récursif* s'il est accepté par une machine de Turing qui s'arrête sur toute entrée.

L est récursif :

$$w \longrightarrow \boxed{\text{M}} \begin{cases} \longrightarrow \text{oui} & w \in L \\ \longrightarrow \text{non} & w \notin L \end{cases}$$

Exemple 3 Tout langage context-free est récursif.

Machines de Turing et fonctions

Machine à calculer sur les entiers : On peut représenter les entiers positifs ou nuls de plusieurs façons :

$$
\begin{array}{l}
0 : | \\
1 : |\,| \\
2 : |\,|\,| \\
3 : |\,|\,|\,| \\
\ldots
\end{array}
$$

ou bien :

$$
\begin{array}{l}
0 : \\
1 : | \\
2 : |\,| \\
3 : |\,|\,| \\
\ldots
\end{array}
$$

ou bien on code les entiers en base k avec $k \geq 2$.

On s'intéresse au contenu de la bande lorsque la machine de Turing s'arrête.

Au départ : $|^{i_1}0|^{i_2}0\dots0|^{i_k}$

A l'arrêt : $|^m$

La machine M calcule f où :

$$f(i_1,i_2,\dots,i_k) = m.$$

f est récursive partielle :

$$i_1,i_2,\dots,i_k \longrightarrow \boxed{\mathrm{M}} \begin{cases} \longrightarrow \text{arrêt avec } f(i_1,i_2,\dots,i_k) \\ \longrightarrow \text{ne s'arrête pas } (f \text{ indéfini}) \end{cases}$$

f est récursive totale :

$$i_1,i_2,\dots,i_k \longrightarrow \boxed{\mathrm{M}} \left\{ \longrightarrow \text{arrêt avec } f(i_1,i_2,\dots,i_k) \right.$$

Il n'y a pas d'états acceptants.

Exemple 4 La soustraction propre :

$$m \ominus n = \begin{cases} m - n & \text{si } m \geq n \\ 0 & \text{sinon} \end{cases}$$

Techniques de construction de machines de Turing

Stockage d'une information dans les états
Cette technique s'applique lorsque l'information est finie.

Exemple 5 *Machine qui teste si le premier symbole d'un mot n'apparaît pas à un autre endroit.*

Exemple 6 *Machine décalant le contenu de la bande d'une case à droite.*

Stockage de l'information dans l'alphabet de bande
Ceci revient à considérer les caractères sur la bande comme des k-uplets.

Exemple 7 *Machine testant la primalité d'un entier.*

Exemple 8 *Machine testant la primalité d'un entier.*

On divise la bande en 3 pistes. Sur le premier, on mets l'entier n en base 2 entouré de marqueurs % et \$. Les symboles d'entrée sont donc [%,#,#], [0,#,#], [1,#,#], et [\$,#,#].

- On mets l'entier 2 en base 2 sur la deuxième piste.
- On recopie la première piste sur la troisième.
- Par soustraction successive de la seconde piste à la troisième, on calcule n mod 2. Si on trouve 0, n n'est pas premier. Sinon, on incrémente l'entier de la seconde piste de 1 et on recommence. Si la seconde piste est égale à la première, l'entier n est premier.

Marquage de symboles

On ajoute une deuxième piste qui contient les symboles $\#$ ou $\$$.

Exemple 9 *Machine qui reconnaît le langage* $L = \{wcw \mid w \in \{a,b\}^+\}$.

$$Q = \{[q,d] \mid q = q_1,q_2, \ldots ,q_9,d = a,b,\#\}$$
$$\Sigma = \{[\#,d] \mid d = a,b,c\}$$

Le symbole d'entrée $[\#,d]$ s'identifie à d

$$q_0 = [q_1,\#] \; F = \{[q_9,\#]\}$$
$$\Gamma = \{[X,d] \mid X = \#,\$, \text{ et } d = a,b,c,\#\}$$

La fonction de transition δ est, où, dans ce qui suit, d ou e valent a ou b :

1. $(q_0,[\#,d]) \longrightarrow ([q_2,d],[\$,d],R)$
   M teste le symbole d'entrée, le stocke dans son contrôle fini et va à droite.
2. $([q_2,d],[\#,e]) \longrightarrow ([q_2,d],[\#,e],R)$
   M va à droite chercher c.
3. $([q_2,d],[\#,c]) \longrightarrow ([q_3,d],[\#,c],R)$
   M a trouvé c. On rentre dans q_3.
4. $([q_3,d],[\$,e]) \longrightarrow ([q_3,d],[\$,e],R)$
   On saute les symboles marqués.

5. $([q_3,d],[\#,d]) \longrightarrow ([q_4,\#],[\$,d],L)$
   M recontre un symbole non marqué. Si c'est celui qui est dans sa mémoire, il le marque et va à gauche. Sinon arrêt. Arrêt aussi si on ne trouve pas de symbole non marqué.
6. $([q_4,\#],[\$,e]) \longrightarrow ([q_4,\#],[\$,e],L)$
   On saute vers la gauche les symboles marqués.
7. $([q_4,\#],[\#,c]) \longrightarrow ([q_5,\#],[\#,c],L)$
   On saute le c.
8. $([q_5,\#],[\#,d]) \longrightarrow ([q_6,\#],[\#,d],L)$
   Si le symbole à gauche du c est non marqué, on va à gauche chercher le premier symbole marqué.
9. $([q_6,\#],[\#,d]) \longrightarrow ([q_6,\#],[\#,d],L)$
10. $([q_6,\#],[\$,d]) \longrightarrow ([q_1,\#],[\$,d],R)$
    Retour à q_1.
11. $([q_5,\#],[\$,d]) \longrightarrow ([q_7,\#],[\$,d],R)$
    Si juste avant le c, le symbole est marqué, M vérifie que tous les symboles de la droite on été marqués. Dans ce cas on accepte le

mot.

12. $([q_7,\#],[\#,c]) \longrightarrow ([q_8,\#],[\#,c],R)$

13. $([q_8,\#],[\$,d]) \longrightarrow ([q_8,\#],[\$,d],R)$

14. $([q_8,\#],[\#,\#]) \longrightarrow ([q_f,\#],[\$,\#],L)$

Sous-programmes

On peut chaîner les machines de Turing en renommant les états.

Exemple 10 *Machine calculant la fonction récursive totale multiplication entière.*

$m,n \geq 0$.

On choisit la représentation suivante :

$$|^m \; 0 \; |^n \longrightarrow |^{mn}$$

Idée générale : On place un 0 à droite de $|^m \; 0 \; |^n$. On copie m fois $|^n$ à droite de ce 0. A chaque fois on détruit un bâton de $|^m$. A la fin, on enlève $0 \; |^n \; 0$.

Le sous-programme COPIE

Il amène de la configuration:

$$|^m \; 0 \; q_1 \; |^n \; 0 \; |^i \; \#$$

à la configuration :

$$|^m \; 0 \; q_5 \; |^n \; 0 \; |^{i+n} \; \#$$

Le sous-programme DEPART
Il amène de la configuration:

$$q_0 \mid^m 0 \mid^n \ldots$$

à la configuration :

$$\# \mid^{m-1} 0 \; q_1 \mid^n \ldots$$

Le sous-programme DECREMENTE
Il amène de la configuration:

$$\#^i \mid^{m-i} 0q_5 \mid^n 0 \mid^{ni} \ldots$$

à la configuration :

$$\#^{i+1} \mid^{m-i-1} 0q_1 \mid^n 0 \mid^{ni} \ldots$$

qui relance COPIE et teste si $i = m$. Dans ce cas, le $0 \mid^n 0$ est enlevé et le calcul s'arrête.

Variantes sur les machines de Turing

Machine à une bande infinie dans les deux sens

$\ldots$	$\#$	$\#$	$\#$	a	b	c	d	$\#$	$\#$	$\#$	$\ldots$

$\boxed{\text{contrôle fini}}$

$M = (Q,\Sigma,\Gamma,\delta,\#,F)$

Mouvements comme sur l'autre modèle avec en plus :

Si $(q,X) \longrightarrow (p,Y,L)$ alors $qX\alpha \vdash_M p\#Y\alpha$

Si $(q,X) \longrightarrow (p,\#,R)$ alors $qX\alpha \vdash_M p\alpha$

Théorème 1 *L est accepté par une machine de Turing à bande bi-infinie ssi L est accepté par une machine de Turing ordinaire.*

Preuve

$\Longleftarrow$ facile, on commence par marquer la cellule à gauche de la position initiale. On simule la machine infinie à droite. Si on atteint l'état marqué, on arrête sans acceptation

$\Longrightarrow$ Soit M_1 une machine à bande bi-infinie. On construit M_2 une machine à bande infinie simulant M_1, à deux pistes :

bande de M_1 :

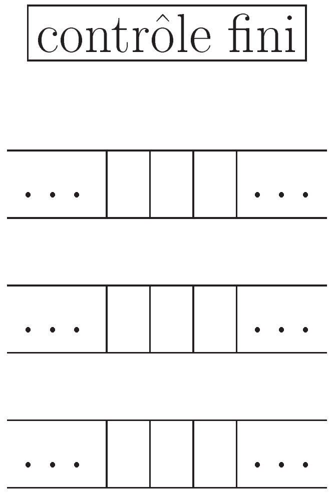
$\ldots$	$\#$	$\#$	$\#$	A_{-1}	A_0	A_1	A_2	A_3	$\#$	$\ldots$

bande de M_2 :

A_0	A_1	A_2	A_3	$\#$	$\#$	$\#$	$\ldots$
$\$$	A_{-1}	A_{-2}	$\#$	$\#$	$\#$	$\#$	$\ldots$

Le contrôle de M_2 indique quelle piste on regarde (U ou D). Les symboles de bande de M_2 sont des couples $[X,Y]$, où X et Y sont des symboles de bande de M_1. De plus Y peut être égal au $.

Machines multibandes



On a k bandes et k têtes fonctionnant indépendamment et pouvant rester immobiles. L'action dépend de l'état et des symboles lus sur chacune des têtes.

Théorème 2 *L est accepté par une machine multibande ssi L est accepté par une machine de Turing classique.*

Preuve

Soit L accepté par une machine M_1 à k bandes. On construit une machine M_2 à une bande et $2k$ pistes, 2 pour chaque bande. L'un contient le contenu de la bande et l'autre des # excepté à la position de la tête de lecture de cette bande où se trouve un marqueur X.
Le contrôle de M_2 stocke l'état de M_1 plus un compteur indiquant le nombre de marqueurs de tête se trouvant à droite de la tête de M_2.

tête 1	...	X	...
bande 1	A_1	A_2	...
tête 2	X	...	...
bande 2	B_1	B_2	...

Notion de complexité :

$L = \{w\tilde{w} \mid w \in \{0,1\}^*\}$.

Avec une bande, $O(n^2)$ mouvements.
Avec deux bandes, $O(n)$ mouvements.

Machines de Turing non déterministes

Un mot est accepté s'il $\boxed{\textbf{existe}}$ une suite de mouvements conduisant à un état acceptant.

Théorème 3 *L est accepté par une machine non déterministe ssi il est accepté par une machine déterministe.*

Preuve
$\Longleftarrow$ évident
$\Longrightarrow$ on numérote tous les choix de mouvements de la machine non déterministe M_1 : $1,2,\ldots,r$. Toute suite finie de mouvements est représentée par un mot de $\{1,2,\ldots,r\}^*$. On construit M_2 déterministe à 3 bandes. On copie l'entrée sur la première. On génère toutes les suites de $\{1,2,\ldots,r\}^*$ sur la deuxième. Pour chaque séquence générée, M_2 copie la première bande sur la troisième et simule M_1.

Machines off-line

On dispose de plusieurs bandes dont une est en lecture seulement. L'entrée est entourée de marqueurs : % . . . entrée . . . $.
Sur la bande d'entrée, la tête de lecture ne peut quitter la région délimitée par les marqueurs.
C'est un cas particulier de machine multibande.
Ce modèle est équivalent aux MT.

Modèles équivalents aux machines de Turing

On va considérer des modèles qui sont à priori moins puissants, mais seulement à priori.

Automates à 2 piles déterministes

Un automate à 2 piles déterministe est une MT avec une entrée en lecture seulement et 2 bandes de stockage. Si une tête va à gauche sur une bande, un # est marqué (dépilement).

Proposition 4 *Une MT peut être simulée par un automate à 2 piles et réciproquement.*

Preuve

Soit M_1 une machine de Turing à simuler. Les symboles à gauche de la tête de M_1 sont placés sur une bande, les symboles de droite sur une autre bande (les symboles près de la tête en haut de la pile).

bande de M_1 :

$\ldots$	#	#	#	A_{-1}	A_0	A_1	A_2	A_3	#	$\ldots$

bandes de l'automate à 2 bandes :

A_3	A_2	A_1	A_0	#	#	#	$\ldots$

A_{-2}	A_{-1}	#	#	#	#	#	$\ldots$

Machines à compteurs

Machine de Turing à $k+1$ bandes dont une entrée "off-line". Les k autres bandes sont des *compteurs*. Les symboles de bande sont Z et $\#$. (Z marque le fond de la pile). Un entier i est stocké sur chaque compteur. On l'incrémente et le décrémente en allant à gauche ou à droite.

%	entrée	$

contrôle fini

Z	$\#$	$\#$	$\#$	$\ldots \quad \ldots \quad \ldots \quad \ldots$

Z	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\ldots$

Théorème 5 *On peut simuler une machine de Turing par une machine à 4 compteurs.*

Preuve
On simule un automate à 1 pile par une machine à 2 compteurs.
Si on a $k-1$ symboles de pile : $Z_1,Z_2,\ldots,Z_{k-1}$, on représente la pile $Z_{i_1},Z_{i_2},\ldots,Z_{i_m}$ par l'entier $j = i_m + ki_{m-1} + \cdots + k^{m-1}i_1$. On mets j dans un compteur et 0 dans l'autre.
Pour empiler Z_r, on fait $j \leftarrow jk + r$.(Ceci est réalisé en utilisant l'autre compteur).
Pour dépiler, on fait $j \leftarrow \lfloor j/k \rfloor$.
Pour dire quel est le symbole au sommet de la pile, si j est dans un compteur, on peut le recopier dans un autre en stockant $j \bmod k$ dans le contrôle fini.

Théorème 6 *On peut simuler* 4 *compteurs avec* 2 *compteurs.*

Preuve
Soient 4 compteurs i,j,k,l. Un seul compteur peut représenter ces 4 entiers. Il contient $n = 2^i 3^j 5^k 7^l$. On mets 0 dans un autre compteur. Pour incrémenter i,j,k ou l, on multiplie n par 2,3,5 ou 7. Idem pour décrémenter. (on divise). On doit aussi montrer comment la machine détermine le mouvement. La machine à 2 compteurs regarde la même cellule que la machine à 4. L'état de la machine à 4 compteurs est stocké dans le contrôle fini. Le mouvement dépend aussi du fait que i,j,k ou l sont ou non nuls. (On teste la divisibilité par 2,3,5 ou 7).

D'où : MT $\Longleftrightarrow$ Machine à 2 compteurs $\Longleftrightarrow$ Automate à 2 piles $\Longleftrightarrow$ Machine à $k \geq 2$ compteurs $\Longleftrightarrow$ Machine à $k \geq 2$ piles.

Enumération par machines de Turing

Un *énumérateur* est une machine de Turing avec une bande de sortie où l'écriture est séquentielle (on n'efface pas).
Sur la sortie M écrit des suites sur l'alphabet Σ séparées par un délimiteur \$.

$G(M)$ est le *langage engendré* par M

$$= \{w \in \Sigma^* \mid w \text{ sera écrit sur la bande}\}$$

$$\text{bandes de travail} \leftarrow \boxed{M} \rightarrow \boxed{\$} \boxed{w_1} \boxed{\$} \boxed{w_2} \boxed{\$} \boxed{w_3} \boxed{\$} \boxed{\ldots}$$

Remarques : si M s'arrête, $G(M)$ est fini. Les mots peuvent être répétés et apparaître dans n'importe quel ordre.

Théorème 7 *Si $L = G(M)$ pour une machine de Turing M, L est récursivement énumérable et réciproquement.*

Preuve

$\Longrightarrow$ Soit M_1 tq $G(M_1) = L$. Soit M_2 avec une bande de plus que M_1. M_2 simule M_1 en utilisant les bandes de M_1. Quand M_1 écrit un \$, M_2 compare le mot qui vient d'être écrit avec son entrée. Si c'est le même, M_2 accepte et sinon continue à simuler M_1.

$\Longleftarrow$ On commence par deux lemmes :

lemme 1 : Il existe une machine de Turing qui engendre $\{a_1,a_2,\ldots,a_k\}^*$ dans l'ordre hiéarchique (Exercice).

lemme 2: Il existe une machine de Turing qui engendre tous les couples d'entiers positifs (i,j) (Exercice).

$(1,1),(1,2),(2,1),(1,3),(2,2),(3,1),\ldots.$

Soit $L = L(M_1)$. M_2 simule la machine qui engendre tous les couples (i,j). Quand (i,j) est généré, M_2 produit le $i^{ème}$ mot w_i pour l'ordre hiéarchique, et simule j mouvements de M_1 sur

w_i. Si M_1 accepte w_i à l'étape j, M_2 génère w_i.
On a $G(M_2) = L$.

Corollaire 8 *Si L est récursivement énumérable, il existe une machine de Turing génératrice de L qui énumère chaque mot de L une fois et une seule.*

Les langages récursifs sont les langages acceptés par des machines de Turing qui s'arrête sur chaque donnée.

Théorème 9 *Un langage est récursif ssi il existe une machine de Turing énumérative M telle que $G(M) = L$ et qui écrit chaque mot de L dans l'ordre hiéarchique.*

Preuve

$\Longrightarrow$ Soit $L = L(M_1)$ avec M_1 s'arrête sur chaque donnée. On construit M_2 qui engendre tous les mots de Σ^* dans l'ordre hiéarchique. Après chaque génération d'un mot w, M_2 simule M_1 sur w.
$\Longleftarrow$ Si M_1 engendre L dans l'ordre hiéarchique. On construit une MT M_2 qui, sur la donnée w, simule M_1 jusqu'à ce que M_1 engendre w ou un mot plus grand. Ceci convient si L est infini

car M_2 s'arrête alors toujours. Si L est fini, M_2 peut boucler sur la donnée w. Or si L est fini, il est récursif. Ceci achève la preuve.

Remarque

Cette preuve n'est pas constructive car il n'y a pas d'algorithme de construction de M_2 à partir de M_1.

MACHINES RAM

Le modèle des machines RAM

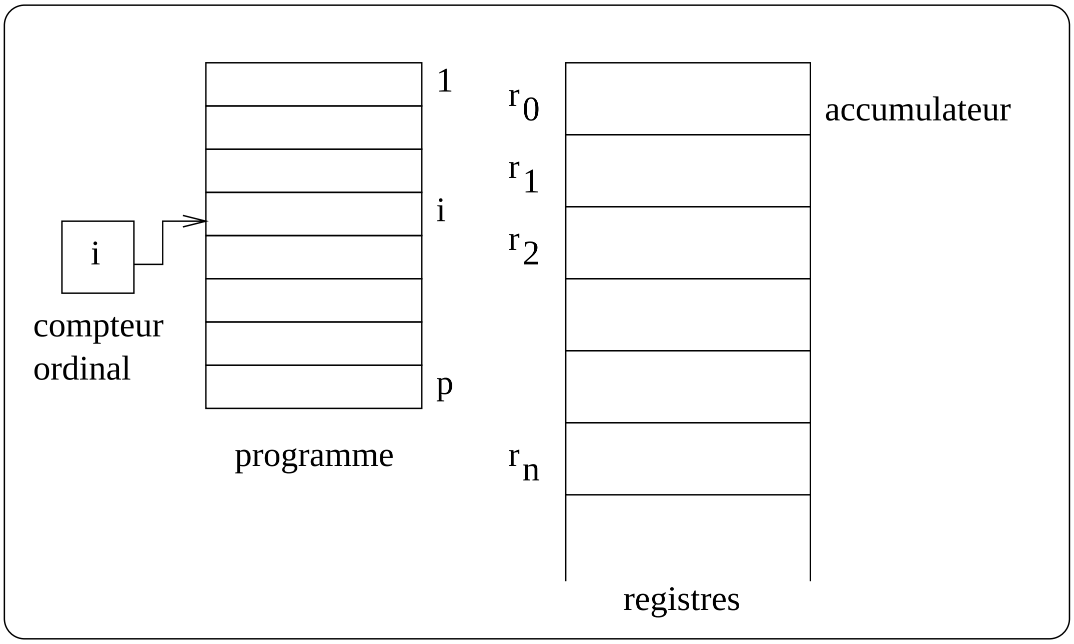
C'est un modèle proche des ordinateurs et de l'assembleur.

Les machines RAM sont constituées de :

- une bande d'entrée (infinie à droite)
- une bande de sortie (infinie à droite)
- des mémoires numérotées appelées *registres*, r_0,r_1,r_2, ... ,r_n, ..., en nombre arbitrairement grand. Chacun d'eux contient un entier. L'indice d'un registre est l'adresse de la mémoire. Le registre 0 et spécial. Il sert à l'arithmétique. On l'appelle l'accumulateur.
- un deuxième ensemble de mémoires numérotées, qui contient de manière inaltérable le programme. C'est une suite d'instructions élémentaires. Chaque instruction a un numéro appelé l'étiquette de l'instruction.
- une mémoire qui contient un numéro d'instruction. C'est le compteur ordinal initialisé à 1.

Dans ce type de machine, on accède à chaque registre par son adresse. D'où le nom RAM.

x_1	x_2	$\ldots$	x_i	$\ldots$	x_n	$\ldots$



y_1	y_2	$\ldots$	y_j	$\ldots$	y_n	$\ldots$

Le programme est une suite d'instuctions élémentaires. Chaque instruction élémentaire a une partie code opération et une partie adresse.
Les opérandes peuvent être de 3 types :

- Si c'est un entier n, l'opérande désigne le contenu du registre d'adresse n.
- Si c'est $A : n$, où est un entier, l'opérande désigne l'entier n lui-même (Absolu).
- Si c'est $I : n$, où est un entier, l'opérande désigne le contenu du registre ayant pour adresse le contenu du registre d'adresse n.

Pour expliquer la signification des instructions, le contenu du registre i est noté $< i >$. Le contenu du registre ayant pour adresse le contenu du registre i, est noté $<< i >>$. ACC désigne l'accumulateur et CO le compteur ordinal.

Instruction	signification
CHARGER n	$\text{ACC}\leftarrow < n >$
CHARGER $A : n$	$\text{ACC}\leftarrow n$
CHARGER $I : n$	$\text{ACC}\leftarrow << n >>$
RANGER n	$n \leftarrow <\text{ACC}>$
RANGER $I : n$	$< n >\leftarrow <\text{ACC}>$
INCREMENTER n	$n \leftarrow < n > +1$
DECREMENTER n	$n \leftarrow < n > -1$
AJOUTER n	$\text{ACC}\leftarrow <\text{ACC}> + < n >$
SOUSTRAIRE n	$\text{ACC}\leftarrow <\text{ACC}> - < n >$
MULTIPLIER n	$\text{ACC}\leftarrow <\text{ACC}> * < n >$
DIVISER n	$\text{ACC}\leftarrow <\text{ACC}> / < n >$
SAUT A n	$\text{CO}\leftarrow n$
SAUT SI ZERO n	$\text{CO}\leftarrow n$ si $<\text{ACC}>= 0$
	$\text{CO}\leftarrow <\text{CO}> +1$ sinon
ARRET	provoque l'arrêt
LIRE n	$n \leftarrow$ l'entier en entrée
ECRIRE n	écriture en sortie de $< n >$.

La machine possède un dispositif qui effectue la conversion d'une chaîne de caractères représentant un entier en représentation binaire placée dans un registre (ce que fait LIRE) et l'inverse (ce

que fait ECRIRE). Les entiers manipulés sont supposés pouvoir être stockés chacun dans un seul registre.
Après chaque instruction, le compteur ordinal est augmenté de 1 sauf si c'est une instruction de rupture de séquence.
Par rapport à un langage assembleur, il manque les instructions de manipulation des caractères et les opérations logiques.

Exemple 11 *Calcul de $n!$.*

1 : LIRE	0	
2 : RANGER	1	{compteur}
3 : RANGER	2	{résultat}
4 : SAUT SI ZERO	14	
5 : DECREMENTER	1	
6 : CHARGER	1	
7 : SAUT SI ZERO	12	
8 : CHARGER	2	
9 : MULTIPLIER	1	
10 : RANGER	2	
11 : SAUT A	5	
12 : ECRIRE	2	
13 : ARRET		
14 : CHARGER	$A:1$	
15 : ECRIRE	0	
16 : ARRET		

Différences entre ce modèle et les machines existantes

- nombre illimité de registres
- on range un nombre arbitrairement grand dans un registre.
- le programme est dans des mémoires inaltérables distinctes des registres.
- instructions élémentaires plus limitées en général que dans un assembleur.

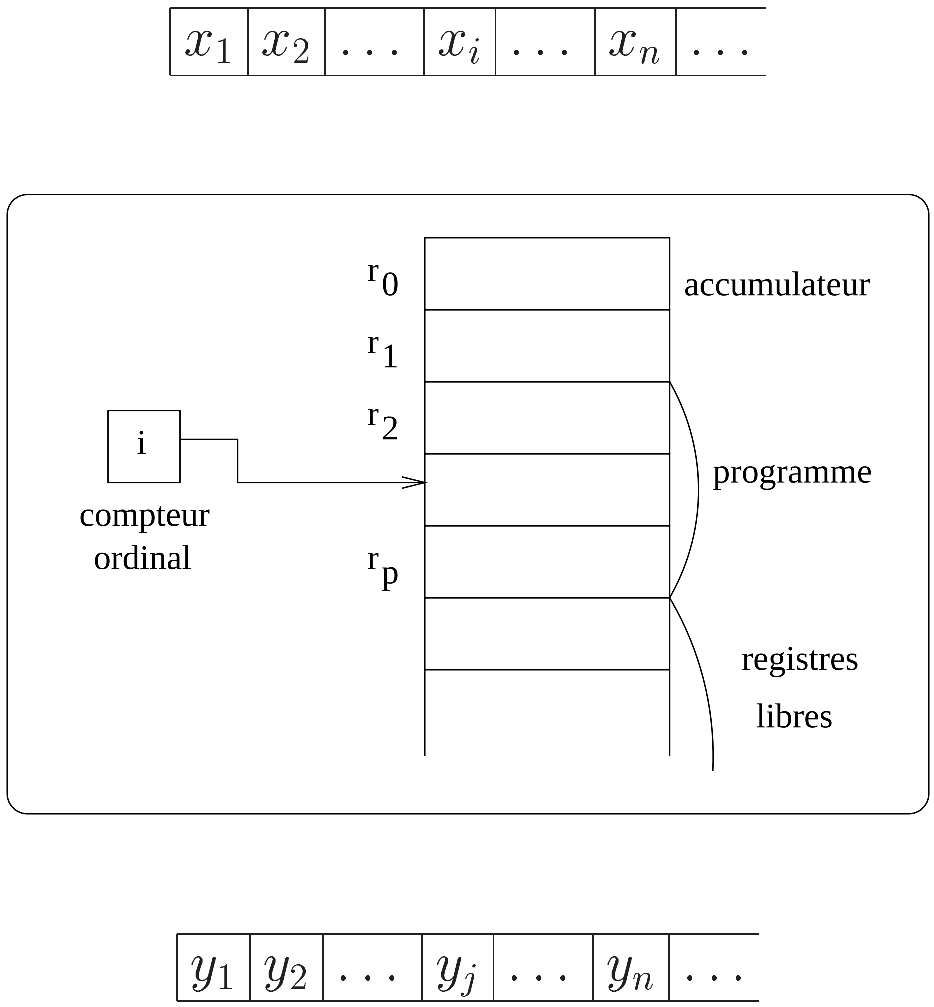
L'avant dernier point conduit aux machines RASP (Random Access with Stored Program).

Le modèle des machines RASP

Comme les RAM mais les mémoires dans lesquelles sont rangées le programme ne se distinguent pas des autres. On va supposer que chaque instruction occupe 2 registres consécutifs, le premier pour l'opération, le second pour l'adresse.

Le registre 0 est toujours l'accumulateur, le registre 1 sert à faire des sauvegardes temporaires de l'accumulateur. Les registres 2 à p contiennent le programme.

La différence essentielle avec le modèle de la machine RAM est donc que l'on peut par programme modifier un registre qui contient ce programme, et donc modifier le programme lui-même.


x_1 x_2 ... x_i ... x_n ...
r_0
r_1
r_2
r_p
accumulateur
programme
registres libres
i
compteur ordinal
y_1 y_2 ... y_j ... y_n ...

Simulation d'une machine par une autre

Lorsqu'on a une RAM M de programme P qui utilise lors de l'exécution le registre 0 et les registres 1 à p, on dit qu'une RAM M' de programme P' simule M s'il existe une injection I de $\mathbb{N}$ dans $\mathbb{N}$ qui envoie 0 sur 0 et si chaque instruction de P est remplacée par une suite d'instructions de P' de telle sorte que, pour toute exécution de P, le contenu de chaque registre $I(r)$ de M' est le même après l'exécution de la suite d'instructions de P' que le contenu du registre r de M après l'exécution des instructions correspondantes de P. (idem pour les RASP)

Exemple : la translation sur les numéros des registres.

A partir d'une RAM de programme P qui utilise le registre 0 et les registres 1 à p, on construit une RAM de programme P_k qui la simule avec $I(0) = 0$ et $I(r) = r + k$ pour $r > 0$. S'il n'y a pas d'indirection, on change les numéros des registres utilisés. Pour l'indirection, on ajoute la valeur k (par AJOUTER $A : k$) au contenu du registre utilisé pour faire l'indirection.

Théorème 10 *On peut simuler une machine RAM par une machine RASP*

Preuve

On écrit le programme de la RAM dans les registres 2 à p de la RASP. On effectue une translation de p sur tous les numéros des registres de la RAM et on les recopie comme registres de la RASP.

La réciproque est vraie. C'est ce qu'on va voir plus tard.

Proposition 11 *Dans le modèle RASP, on peut se passer de l'indirection.*

Preuve Soit un programme RASP écrit dans les registres 2 à p et utilisant les registres $p+1$ à q. On construit un nouveau programme qui s'écrit dans les registres 2 à p' en utilisant les registres $p'+1$ à $q+p'-p$ en remplaçant chaque instruction d'indirection par 6 instructions n'en comportant pas.

Exemple : simulation de MULTIPLIER $I : n$:

10 : RANGER 1 (sauvegarde de l'accumulateur)

12 : CHARGER $n + p' - p$

14 : AJOUTER $p' - p$ (ACC contient alors l'adresse du registre dans lequel se trouve la valeur à multiplier)

16 : RANGER 21 (l'exécution modifie le programme lui-même)

18 : CHARGER 1 (on restitue l'accumulateur)

20 : MULTIPLIER x (on effectue l'opération)

Un modèle de RAM rudimentaire

Simulation des opérations sur les chaînes de caractères par les opérations sur les entiers

On code chaque caractère par une suite de lettres (8 par exemple) sur l'alphabet $\{0,1\}$. Une telle suite est ensuite manipulée en temps que représentation d'un entier.

Simplification des instructions de saut

On peut se restreindre à SAUT SI ZERO et ARRET.

Exemple : Montrons que l'on peut se passer de "SAUT A n".

On utilise le registre 1 comme registre auxiliaire pour sauvegarder l'accumulateur. Devant chaque instruction, on insère la séquence :

RANGER 1
CHARGER 1

Puis, on remplace le saut par :
RANGER 1
CHARGER $A : 0$
SAUT SI ZERO n' (où n' est l'étiquette de l'instruction CHARGER 1 introduite devant l'instruction d'étiquette n.

Simplification des expressions arithmétiques
On peut se restreindre à INCREMENTER et DECREMENTER.
Exemple : Montrons comment on peut se passer de MULTIPLIER n. Ceci revient à ajouter $< n >$ fois le contenu de l'accumulateur à 0. On utilise les registres 1 à 3 comme auxiliaire.

```
         RANGER 1
         CHARGER n
         RANGER 2
         CHARGER A : 0
         RANGER 3
retour:  CHARGER 2
         SAUT SI ZERO fin
         DECREMENTER 2
         CHARGER 3
         AJOUTER 1
         RANGER 3
         SAUT A retour
fin:     CHARGER 3
```

On peut se passer de "AJOUTER n" en incrémentant $< n >$ fois le contenu de l'accumulateur. Etc ...

Une RAM rudimentaire

Instruction	Signification
$I < n >$	Incrémenter le contenu du registre n
$D < n >$	Décrémenter le contenu du registre n
$Z < n >$	Mise à zéro du contenu du registre n
$E < n,m >$	Echanger les contenus des registres n et m
$T < n > (i,j)$	Si le contenu du registre n est nul aller à l'étiquette i sinon aller à l'étiquette j
Arrêt	Arrêt de l'exécution

Exemple : Si les registres 1 et 2 contiennent n et p, ils contiennent $n + p$ et 0 après :

1 : $T < 2 > (5,2)$
2 : $I < 1 >$
3 : $D < 2 >$
4 : $T < 3 > (1,1)$
5 : Arrêt

Une RAM universelle

On construit une RAM qui simule n'importe quelle RASP sans indirection.

On lit le programme RASP que l'on met dans les registres 4 à p.
registre 0 : accumulateur de la RAM
registre 1 : sert pour faire de l'adressage indirect
registre 2 : simule le compteur ordinal de la RASP
registre 3 : simule l'accumulateur de la RASP

On effectue une boucle sur les opérations :
CHARGER I : 2
analyser le code de l'opération et se brancher au début d'une suite d'instructions adaptée.
Exemple : Si x (resp. z) est le code de DIVISER (resp. SAUT SI ZERO) et si y (resp. t) est l'étiquette des opérations pour simuler DIVISER (resp. SAUT SI ZERO),

SOUSTRAIRE $A : x$
SAUT SI ZERO y
AJOUTER $A : x$
SOUSTRAIRE $A : z$
SAUT SI ZERO t
AJOUTER $A : z$

A l'étiquette y, on effectue la simulation de DIVISER:
INCREMENTER 2 (le registre 2 contient l'adresse du code de l'opérande)
CHARGER 2
RANGER 1 (on met cette adresse dans le registre 1)
CHARGER 3
DIIVISER I : 1
RANGER 3 (on simule l'opération)
INCREMENTER 2 (2 contient l'adresse de l'instruction suivante à simuler)
SAUT A boucle

A l'étiquette t, on simule SAUT SI ZERO:

CHARGER 3	(on teste si l'accumulateur de la RASP est nul)
SAUT SI ZERO etiq	
INCREMENTER 2	
INCREMENTER 2	
SAUT A boucle	
etiq: INCREMENTER 2	(test positif, le registre 2 contient l'adresse du registre contenant l'opérande)
CHARGER 2	
RANGER 1	(range dans 1)
CHARGER I : 1	(on charge l'adresse du saut conditionnel)
RANGER 2	
SAUT A boucle	

La machine RAM a pour donnée le programme P d'une RASP, suivie d'une donnée D pour le programme P. Elle a pour résultat le résultat de P sur D.

Corollaire 12 *Les modèles RAM et RASP ont même puissance.*

De même, on peut construire une RAM qui simule le fonctionnement de toutes les RAMs. Une telle machine d'un modèle de calcul qui simule le fonctionnement de tous les élements du modèle s'appelle un élément **universel** pour le modèle.

Machines de Turing et machines RAM

Théorème 13 *Les modèles RAM et MT ont même puissance.*

$\Longleftarrow$ Une MT peut simuler une RAM :
On construit une machine de Turing multibande.

- une bande pour l'entrée
- une bande pour la sortie
- une bande pour l'accumulateur
- une bande pour le compteur ordinal
- une bande pour chaque registre
- une bande où est stocké le programme sous la forme :

  $$\$0 * v_0\$1 * v_1\$10 * v_2\$ \ldots \$i * v_i\$ \ldots ,$$

  où v_i est le code de l'instruction d'étiquette i et i est stocké en binaire.
- une bande de travail

Supposons que le compteur ordinal vaut i, la MT recherche sur la bande du programme $\$i*$. Supposons que la MT M lise après "AJOUTER 2". M additionne le contenu du registre 2 dans l'accumulateur, et incrémente la bande compteur ordinal. On répète le cycle avec l'instruction suivante.
$\Longrightarrow$ On écrit un programme en C simulant une machine de Turing. On compile ce programme en assembleur de machine à accès direct.

INDECIDABILITE I

Indécidabilité

En codant les paramètres d'un problème au moyen de symboles, on peut formaliser un problème dont la réponse est oui ou non en terme de langages.

Exemple 12

Problème : Déterminer si une machine de Turing arbitraire s'arrête sur le mot vide.
Donnée : Une machine de Turing.
Question : Cette machine de Turing accepte-t-elle le mot vide ?

On code une MT par une suite de 0 et de 1. L'ensemble des suites codant une MT qui accepte le mot vide est un langage récursivement énumérable mais pas récursif.
Il n'existe pas d'algorithme pour décider si une MT accepte ou non le mot vide.

Exemple 13

Problème : AMB(G)
Donnée : Une grammaire G.
Question : Cette grammaire est-elle ambiguë?

0n code la grammaire G. Equivalence des codages de G. AMB est décidable ssi le langage constitué des grammaires ambiguës est récursif.

$$G \longrightarrow \boxed{M} \begin{cases} \longrightarrow \text{oui} & G \text{ ambiguë} \\ \longrightarrow \text{non} & G \text{ non ambiguë} \end{cases}$$

ou bien, fonction logique =prédicat

$$\text{AMB}(G) = \begin{cases} \text{oui, vrai, 1} & \text{si } G \text{ ambiguë} \\ \text{non, faux, 0} & \text{si } G \text{ non ambiguë} \end{cases}$$

Problème décidable $\Longleftrightarrow$ prédicat récursif total.

Un problème dont le langage est récursif est dit *décidable*. Un problème est indécidable ssi il n'existe pas d'algorithme qui prend en entrée une instance du problème et détermine si la réponse pour cette donnée est oui ou non.

On peut imaginer d'autres questions :

$$\text{TROUVER}(G) = \begin{cases} w \text{ avec 2 arbres} & \text{si } G \text{ ambiguë} \\ \text{syntaxiques} & \\ \text{non} & \text{sinon} \end{cases}$$

En fait, AMB et TROUVER sont équivalents.

On peut *réduire* AMB à TROUVER et réduire TROUVER à AMB.

Réduire AMB à TROUVER : Un algorithme pour TROUVER fournit un algorithme pour AMB.
Réduire TROUVER à AMB : Un algorithme pour AMB fournit un algorithme pour TROUVER.

Réduction

Réduire un problème A à un problème B :

Pour toute instance (ou donnée) i du problème A, on peut construire par un algorithme une instance i' du problème B telle que la réponse à la question de B sur i' est oui ssi la réponse à la question de A sur i est oui.

un algorithme pour B fournit un algorithme pour A.

Propriétés des langages récursifs et récursivement énumérables

Théorème 14 *Les langages récursifs sont fermés par complémentation, union, intersection.*

Preuve

$$w \longrightarrow \boxed{M} \begin{cases} \longrightarrow \text{oui} & \text{oui} \\ \longrightarrow \text{non} & \text{non} \end{cases}$$

$$w \longrightarrow \begin{array}{c} \boxed{M_1} \\[2em] \boxed{M_2} \end{array} \left. \begin{array}{l} \begin{cases} \longrightarrow \text{oui} \\ \longrightarrow \text{non} \end{cases} \\ \begin{cases} \longrightarrow \text{oui} \\ \longrightarrow \text{non} \end{cases} \end{array} \right\} \text{ou logique} \begin{array}{l} \longrightarrow \text{oui} \\ \longrightarrow \text{non} \end{array}$$

Théorème 15 *Les langages récursivement énumérables sont fermés par union.*

$$w \longrightarrow \begin{array}{c} \boxed{M_1} \\ \boxed{M_2} \end{array} \begin{array}{l} \{ \longrightarrow \text{oui} \\ \{ \longrightarrow \text{oui} \end{array} \longrightarrow \text{oui}$$

Théorème 16 *Si L et $\overline{L}$ sont tous les deux récursivement énumérables, alors L est récursif.*

Preuve

$$w \longrightarrow \begin{array}{l} \boxed{M_1} \left\{ \longrightarrow \text{oui} \longrightarrow \text{oui} \right. \\ \boxed{M_2} \left\{ \longrightarrow \text{oui} \longrightarrow \text{non} \right. \end{array}$$

En conclusion, on a ou bien

- L et $\overline{L}$ sont récursifs
- Ni L ni $\overline{L}$ ne sont récursivement énumérables
- De L et $\overline{L}$, l'un est récursivement énumérable, l'autre pas

Technique pour montrer que L est non récursif : montrer que $\overline{L}$ n'est pas récursivement énumérable.

Machines universelles et diagonalisation

Codage des machines de Turing

$$M = (Q,\{0,1\},\{0,1,\#\},\delta,q_1,\#,\{q_2\})$$

Ces restrictions ne compromettent pas la généralité de l'argumentation.
On code l'alphabet en unaire :

$$\begin{array}{ll} 0 & \longrightarrow 0 \\ 1 & \longrightarrow 00 \\ \# & \longrightarrow 000 \\ L & \longrightarrow 0 \\ R & \longrightarrow 00 \end{array}$$

Et $\delta(q_i,X) = (q_j,Y,R)$ est codé par :

$$0^i 1 \underbrace{0 \ldots 0}_{X} 1 0^j 1 \underbrace{0 \ldots 0}_{Y} 1 \underbrace{0 \ldots 0}_{R}$$

La fonction δ est alors codée par :

$$111 code_1 11 code_2 11 \ldots 11 code_r 111$$

N'importe quel code d'une machine de Turing M est noté $< M >$. Un code de M suivi d'une donnée w est noté $< M,w >$.

Numérotage des machines de Turing

Bijection entre
entiers $\longleftrightarrow$ codes de machines (peut être réalisé par machine) $\longleftrightarrow$ machines

$\boxed{\text{Gén}}$ engendre tous les codes en ordre hiéarchique et vérifie s'ils correspondent au code d'une machine de Turing.

(On vérifie que deux codes de mouvements ne commencent pas par 0^i10^j1 pour les mêmes indices i et j. On teste aussi que si $0^i10^j10^k10^l10^m$ est le code d'un mouvement, alors $1 \leq j \leq 3$, $1 \leq l \leq 3$, et $1 \leq m \leq 2$.)
L'ensemble des codes des machines de Turing est un langage récursif.

$$\begin{array}{rl} \boxed{\text{Gén}} \longrightarrow & \boxed{\begin{array}{l}\text{Teste l'égalité}\\ + \text{ compteur}\end{array}} \\ \text{machine } M \longrightarrow < M > \longrightarrow & \\ & \quad \downarrow \text{numéro } n \end{array}$$

$$\begin{array}{r} \boxed{\text{Gén}} \longrightarrow \\ \text{numéro } n \longrightarrow \end{array} \boxed{\begin{array}{l}\text{Compteur}\\ \text{jusqu'à } n\end{array}} \longrightarrow < M > \longrightarrow M$$

On peut alors parler de **la** machine de Turing M_n. Du point de vue de la calculabilité, opérer sur n ou sur le code de M_n est équivalent.

Exemple 14 *Premières machines de Turing*

111111	aucune transition
111010101010111	$\delta(q_1,0) = (q_1,0,L)$
1110010101010111	$\delta(q_2,0) = (q_1,0,L)$
1110100101010111	$\delta(q_1,1) = (q_1,0,L)$
...	...

Les machines n'acceptent rien au début. La $3^{ème}$ ci-dessus est incorrecte car q_2 est état final.

L'argument diagonal

Table universelle :

$$U[i,j] = \begin{cases} 1 & \text{si } w_i \in L(M_j) \\ 0 & \text{sinon} \end{cases}$$

où w_i est le $i^{ème}$ mot pour l'ordre hiéarchique dans $\{0,1\}^*$. Cette table est à usage théorique uniquement. Soit

$$L_d = \{w_i \mid w_i \notin L(M_i)\}$$

ou bien

$$L_d = \{w_i \mid U[i,i] = 0\}$$

Proposition 17 *L_d n'est pas récursivement énumérable.*

Preuve Si $L_d = L(M_k)$ pour un certain indice k, on considère w_k. On a

$$w_k \in L_d = L(M_k) \iff U[k,k] = 1 \iff w_k \notin L_d$$

Ceci contredit l'hypothèse $L_d = L(M_k)$.

Le langage universel

$$L_u = \{(n,k) \mid w_n \in L(M_k)\}$$

ou bien

$$L_u = \{(n,k) \mid U[n,k] = 1\}$$

Théorème 18 *L_u est récursivement énumérable.*

Preuve On construit une machine de Turing M_u qui accepte L_u.

- Cette machine construit $< M_k,w_n >$ à partir de (n,k) en simulant $\boxed{\text{Gén}}$, et met le résultat dans sa première bande.
- La seconde bande de M_u simule la bande de M_k.
- La troisième bande de M_u contient l'état courant de M_k, q_i représenté par 0^i.
- Bandes de calcul annexes eventuelles.

Le comportement de M_u est :

- Initialiser la bande 2 à w_n repéré après le deuxième bloc de 111. Initialiser la bande 3 à 0.
- Si la bande 3 vaut 00, arrêter et accepter.
- Si X_j et le symbole courant sur la tête de la bande 2 et si la bande 3 vaut 0^i, chercher sur la bande 1 s'il existe le code d'un mouvement commençant par 110^i10^j1. Si on n'en trouve pas, arrêter en rejetant. Si on trouve $110^i10^j10^k10^l10^m$, mettre 0^k sur la bande 3, écrire X_l sur la bande 2 et décaler la tête en direction D_m. Itérer la simulation des mouvements de M_k.

Il est clair que M_u accepte (n,k) ssi M_k accepte w_n.

C'est la situation habituelle en informatique : aucune différence entre programme M_k et données w_n.

$(n,k) \longrightarrow \boxed{M_u} \longrightarrow$ accepte ssi M_k accepte w_n

M_u est appelée *machine de Turing universelle.*

Théorème 19 *L_u n'est pas récursif.*

Preuve L_d est non récursivement énumérable, donc $\overline{L_d}$ n'est pas récursif. Supposons L_u récursif :

$$(n,k) \longrightarrow \boxed{A} \begin{cases} \longrightarrow & \text{oui} \\ \longrightarrow & \text{non} \end{cases}$$

$$i \longrightarrow (i,i) \longrightarrow \boxed{A} \begin{cases} \longrightarrow & \text{oui, } w_i \in L(M_i) \\ \longrightarrow & \text{non, } w_i \notin L(M_i) \end{cases}$$

ce qui établirait la récursivité de $\overline{L_d} = \{i \mid w_i \in L(M_i)\}$. On a ainsi réduit $\overline{L_d}$ à L_u.

Corollaire 20 *Le problème $w \in L(M)$ est indécidable.*

Preuve Via la numérotation, ce problème se ramène à $w_n \in L(M_k)$, donc à $(n,k) \in L_u$, qui n'est pas récursif.

D'autres problèmes indécidables

Le problème de l'arrêt

Théorème 21 *Le problème "M avec w en entrée s'arrête-t-elle ou boucle-t-elle ? " est indécidable.*

Preuve Supposons qu'il existe un algorithme $\boxed{A}$ pour ce problème.

$$(n,k) \longrightarrow \boxed{A} \begin{cases} \longrightarrow \text{oui, si } M_k \text{ s'arrête sur } w_n \\ \longrightarrow \text{non, si } M_k \text{ boucle sur } w_n \end{cases}$$

On en déduit un algorithme pour le problème "$w \in L(M)$? ".

$$(n,k) \longrightarrow \boxed{A} \begin{cases} \longrightarrow \text{oui} \longrightarrow \boxed{M_u} \begin{cases} \to \text{oui} \to \text{oui} \\ \end{cases} \\ \longrightarrow \text{non} \longrightarrow \text{non} \end{cases}$$

On a réduit le problème "$w \in L(M)$? " au problème de l'arrêt.

Autre preuve (directe)

$$n \longrightarrow (n,n) \longrightarrow \boxed{A} \begin{cases} \longrightarrow \text{oui} \longrightarrow \boxed{\text{Boucle}} \\ \longrightarrow \text{non} \longrightarrow \text{oui} \end{cases}$$

machine B

La machine B s'arrête sur la donnée n si et seulement si M_n ne s'arrête pas sur la donnée n. La machine B a un numéro : k. Donc $B = M_k$. Donc M_k s'arrête sur la donnée k ssi M_k ne s'arrête pas sur la donnée k. Contradiction.

Théorème 22 *Le problème "$L(M)$ est-il récursif ? " est indécidable.*

Preuve Soient :

$$L_r = \{k \mid L(M_k) \text{ est récursif}\}$$

$$L_{nr} = \{k \mid L(M_k) \text{ n'est pas récursif}\}$$

Aucun des deux langages n'est récursivement énumérable, d'où le théorème.

Montrons que L_r n'est pas récursivement énumérable.

$$\begin{array}{l}(n,k) \longrightarrow \boxed{M_u} \longrightarrow \text{oui} \longrightarrow \boxed{M_u} \longrightarrow \text{oui} \\ (m,l) \end{array}$$

Machine A

La machine A a comme entrée (n,k) et (m,l).
Soit $A(n,k)$ la machine avec comme entrée (m,l) qui recopie (n,k), puis déroule A avec arguments (n,k) et (m,l).
D'où l'algorithme $\boxed{\text{M}}$:

$$(n,k) \longrightarrow \boxed{M} \longrightarrow a(n,k),$$

où $a(n,k)$ est le numéro de $A(n,k)$.
On a :

$$L(A(n,k)) = \begin{cases} \emptyset & \text{si } M_k \text{ n'accepte pas } w_n \\ L_u & \text{si } M_k \text{ accepte } w_n \end{cases}$$

Si $w_n \in L(M_k)$, (m,l) est accepté par $A(n,k)$ ssi $(m,l) \in L_u$.
Si $w_n \notin L(M_k)$, $A(n,k)$ accepte $\emptyset$.

Supposons L_r récursivement énumérable, soit M_r qui accepte L_r.

$$k \longrightarrow \boxed{M_r} \big\{ \longrightarrow \text{oui} \iff L(M_k) \text{ récursif}$$

Soit M' la machine :

$$(n,k) \longrightarrow \boxed{M} \longrightarrow a(n,k) \longrightarrow \boxed{M_r} \big\{ \longrightarrow \text{oui} \longrightarrow \text{oui}$$

M' s'arrête avec oui ssi :

$$\begin{aligned} & L(A(n,k))\text{récursif} \\ \iff & L(A(n,k)) = \emptyset \\ & (\text{car } L_u \text{ n'est pas récursif}) \\ \iff & w_n \notin L(M_k) \\ \iff & (n,k) \in \overline{L_u} \end{aligned}$$

La machine M' accepte $\overline{L_u}$. Contradiction car L_u n'est pas récursif.

Montrons que L_{nr} n'est pas récursivement énumérable.
C'est la même technique avec une machine B :

$$\left.\begin{array}{l}(n,k) \longrightarrow \boxed{M_u} \longrightarrow \text{oui} \\ (m,l) \longrightarrow \boxed{M_u} \longrightarrow \text{oui}\end{array}\right\} \text{ou} \longrightarrow \text{oui}$$

Puis $B(n,k)$ et $\boxed{M}$:

$$(n,k) \longrightarrow \boxed{M} \longrightarrow b(n,k),$$

où $b(n,k)$ est le numéro de $B(n,k)$.
Si $w_n \in L(M_k)$ alors (m,l) est accepté par $B(n,k) \Longleftrightarrow (m,l)$ quelconque.
Si $w_n \notin L(M_k)$ alors (m,l) est accepté par $B(n,k) \Longleftrightarrow (m,l) \in L_u$.

$$L(B(n,k)) = \begin{cases} L_u & \text{si } M_k \text{ n'accepte pas } w_n \\ \{0,1\}^* & \text{si } M_k \text{ accepte } w_n \end{cases}$$

Supposons L_{nr} récursivement énumérable, soit M_{nr} qui accepte L_{nr}.
Soit M' la machine :

$$(n,k) \longrightarrow \boxed{M} \longrightarrow b(n,k) \longrightarrow \boxed{M_{nr}} \Big\{ \longrightarrow \text{oui} \longrightarrow \text{oui}$$

M' accepte (n,k) ssi :

$$\begin{aligned} & L(B(n,k))\text{non récursif} \\ \iff & L(B(n,k)) = L_u \\ \iff & w_n \notin L(M_k) \\ \iff & (n,k) \in \overline{L_u} \end{aligned}$$

La machine M' accepte $\overline{L_u}$. Contradiction car L_u n'est pas récursif.

Théorème de Rice

Quels sont les problèmes décidables sur les langages spécifiés par machine de Turing?

Soit $\mathcal{L}$ l'ensemble des langages récursivement énumérables.
Soit $\mathcal{S} \subset \mathcal{L}$ un ensemble de langages récursivement énumérables.
On appelle $\mathcal{S}$ une *propriété* des langages. Le langage L a la propriété $\mathcal{S}$ ssi $L \in \mathcal{S}$.

Exemples

$$\begin{aligned}
\mathcal{S} &= \{\text{langages } L \mid \epsilon \in L\} \\
\mathcal{S} &= \{\emptyset\} \text{ langage vide} \\
\mathcal{S} &= \mathcal{L} \text{ ou } \mathcal{S} = \emptyset \text{ propriété triviale}
\end{aligned}$$

Théorème 23 *Si $\mathcal{S}$ n'est pas triviale, le problème "$L(M)$ a-t-il la propriété $\mathcal{S}$?" ou "$L(M) \in \mathcal{S}$?" est indécidable.*

Preuve Même technique.
Supposons qu'il existe un algorithme pour décider "$L(M) \in \mathcal{S}$", soit $M_{\mathcal{S}}$. Il décide aussi pour "$L(M) \notin \mathcal{S}$" $\iff L(M) \in \overline{\mathcal{S}}$. On a soit $\mathcal{S}$ soit $\overline{\mathcal{S}}$ ne contient pas $\emptyset$. Supposons que $\emptyset \notin \mathcal{S}$. Puisque $\mathcal{S}$ non trivial, on trouve $L \in \mathcal{S}$ et M_L qui accepte L en utilisant $M_{\mathcal{S}}$ sur toutes les machines de Turing jusqu'à trouver M_L.

$$\begin{array}{l}
(n,k) \longrightarrow \boxed{M_u} \longrightarrow \text{oui} \longrightarrow \boxed{M_L} \Big\{ \longrightarrow \text{oui} \longrightarrow \text{oui} \\
m \text{———————————————}
\end{array}$$

Machine A

D'où

$$(n,k) \longrightarrow \boxed{M} \longrightarrow a(n,k),$$

où $a(n,k)$ est le numéro de $A(n,k)$ telle que :
Si $w_n \in L(M_k)$ alors $L(A(n,k)) = L$.
Si $w_n \notin L(M_k)$ alors $L(A(n,k)) = \emptyset$.
Puis :

$$(n,k) \longrightarrow \boxed{M} \longrightarrow a(n,k) \longrightarrow \boxed{M_{\mathcal{S}}} \begin{cases} \longrightarrow \text{oui} \longrightarrow \text{oui} \\ \longrightarrow \text{non} \longrightarrow \text{non} \end{cases}$$

Cette machine s'arrête avec oui ssi :

$$\begin{aligned} & L(A(n,k)) \in \mathcal{S} \\ \Longleftrightarrow\ & L(A(n,k)) = L \\ \Longleftrightarrow\ & w_n \in L(M_k) \\ \Longleftrightarrow\ & (n,k) \in L_u \end{aligned}$$

Elle s'arrête avec non ssi $(n,k) \in \overline{L_u}$. Contradiction avec L_u non récursif.

Corollaire 24 *Nombreuses applications.*
$L(M) = \emptyset$.
$L(M)$ *fini,* $L(M)$ *rationnel,* $L(M)$ *context-free.*
$\epsilon \in L(M)$, $L(M) = L$ *donné,* $L \in \mathcal{L}$.
$L(M_1) = L(M_2)$ *(réduction du précédent).*
$L(M_1) \cap L(M_2) = \emptyset$.

Raffinement du théorème de Rice récursif

Soit $L_{\mathcal{S}} = \{n \mid L(M_n) \in \mathcal{S}\}$.
On a vu que $L_{\mathcal{S}}$ n'est pas récursif. Est-il au moins récursivement énumérable?

Théorème 25 *$L_{\mathcal{S}}$ est récursivement énumérable ssi $\mathcal{S}$ satisfait 3 conditions :*

1. *Si $L \in \mathcal{S}$ et $L \subset L'$, avec L' r.e, alors $L' \in \mathcal{S}$. (Propriété de contenance)*
2. *Si $L \in \mathcal{S}$ est infini, alors il existe un sous-ensemble fini de L dans $\mathcal{S}$.*
3. *L'ensemble des langages finis de $\mathcal{S}$ est énumérable.*

Corollaire 26 *Ne sont pas récursivement énumérables :*

$$\begin{array}{ll} L(M) = \emptyset & (1) \\ L(M) \textit{ non récursif} & (1) \\ L(M) \textit{ récursif} & (1) \\ L(M) \textit{ a un élément} & (1) \\ L(M) \textit{ rationnel} & (1) \\ L(M) = \{0{,}1\}^* & (2) \\ L(M) - L_u \neq \emptyset & (3) \end{array}$$

Sont récursivement énumérables :

$$\begin{array}{l} L(M) \neq \emptyset \\ L(M) \textit{ a au moins } 10 \textit{ éléments} \\ w \in L(M) \textit{ où } w \textit{ fixé} \\ L(M) \cap L_u \neq \emptyset \end{array}$$

INDECIDABILITE II

Autres problèmes indécidables

Le Problème de Correspondance de Post (PCP)

Etant données deux listes finies de mots sur Σ :

$$A = w_1, \ldots, w_k \text{ et } B = x_1, \ldots, x_k$$

Le problème a une solution s'il existe $i_1, i_2, \ldots, i_m$ avec $m \geq 1$ et :

$$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$$

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

	A	B
1	1	111
2	10111	10
3	10	0

a une solution :

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110$$

Le PCP :

	A	B
1	10	101
2	011	11
3	101	011

n'a pas de solution.

Problème modifié (PCPM)
On exige que la solution débute par :

$$w_1 \ldots = x_1 \ldots$$

Théorème 27 *Si PCP est décidable, alors PCPM est décidable. Ou encore PCPM se réduit à PCP.*

Preuve Soit $(A = w_1 \ldots, w_k, B = x_1, \ldots, x_k)$ une instance de PCPM. On convertit cette instance de PCPM en une instance de PCP telle que l'instance de PCP a une solution ssi l'instance de PCPM a une solution.

Soit Σ l'alphabet des listes A et B. Soit $\Sigma' = \Sigma \cup \{\#, \$\}$. Soient C et D deux listes de mots y_i, et z_i, où y_i est obtenu en insérant $\#$ après chaque caractère de w_i et z_i est obtenu en insérant # avant chaque caractère de x_i. On ajoute aussi les nouveaux mots :

$$\begin{array}{rclr} y_0 & = & \#y_1 & y_{k+1} = \$ \\ z_0 & = & z_1 & z_{k+1} = \#\$ \end{array}$$

Une solution de PCPM $(A,B) \iff$ Une solution de PCP (C,D).

Théorème 28 *PCP est indécidable.*

Preuve On montre que PCPM est indécidable. Pour cela on réduit "$w \in L(M)$" ou L_u à PCPM. A partir de (n,k), on construit une instance de PCPM qui a une solution ssi M_k accepte w_n. Cette instance est telle que, si elle a une solution, elle en a une qui commence par :

$$\#q_0w_n\#\alpha_1q_1\beta_1\#\ldots\#\alpha_rq_r\beta_r\#$$

où les $\alpha_iq_i\beta_i$ sont des configurations successives de M_k sur w_n, et q_r est état final. Soit $M_k = (Q,\Sigma,\Gamma,\delta,q_0,\$,F)$. Les listes A et B sont :

Première paire :

A	B
$\#$	$\#q_0w_n\#$

Groupe 1 : pour chaque X dans Γ.

A	B
X	X
$\#$	$\#$

Groupe 2 : pour chaque $q \in Q - F$, $p \in Q$, $X,Y,Z \in \Gamma$.

A	B	
qX	Yp	si $(q,X) \longrightarrow (p,Y,R)$
ZqX	pZY	si $(q,X) \longrightarrow (p,Y,L)$
$q\#$	$Yp\#$	si $(q,\$) \longrightarrow (p,Y,R)$
$Zq\#$	$pZY\#$	si $(q,\$) \longrightarrow (p,Y,L)$

Groupe 3 : pour chaque $q \in F$, et $X,Y \in \Gamma$.

A	B
XqY	q
Xq	q
qY	q

Groupe 4 : pour chaque $q \in F$.

A	B
$q\#\#$	$\#$

On dit que (x,y) est une solution partielle à PCPM si x et y sont concaténation de mots en correspondance des listes A et B resp. et si x est préfixe de y. On note z le mot tel que $xz = y$. Supposons qu'il existe une suite valide de r descriptions instantanées à partir de q_0w_n. On a alors la solution partielle :

$$\begin{aligned} x &= \#q_0w_n\#\alpha_1q_1\beta_1\#\ldots\#\alpha_{r-1}q_{r-1}\beta_{r-1}\# \\ y &= \#q_0w_n\#\alpha_1q_1\beta_1\#\ldots\#\alpha_rq_r\beta_r\# \\ z &= \alpha_rq_r\beta_r\#. \end{aligned}$$

Se montre par récurrence sur r.
Pour $r = 0$, on a la solution partielle $(\#,\#q_0w_n\#)$.
Si le résultat est vrai pour r avec $q_r \in Q-F$, on utilise les paires des groupes 1 et 2 pour obtenir le résultat pour $r + 1$. Il y a au plus une paire du groupe 2 qui permet de continuer la solution partielle après q_r.
La nouvelle solution partielle obtenue est

$(y,y\alpha_{r+1}q_{r+1}\beta_{r+1}\#)$ et $\alpha_{r+1}q_{r+1}\beta_{r+1}$ est une configuration accessible en un coup à partir de la configuration $\alpha_r q_r \beta_r$.
De plus il n'y a pas d'autres solutions partielles dont la longueur du second mot est $|y\alpha_{r+1}q_{r+1}\beta_{r+1}\#|$.
Si $q_r \in F$, on utilise les paires des groupes 1 et 3 et on termine par la paire du groupe 4 pour prolonger la solution partielle (x,y) en une solution pour l'instance (A,B) de PCPM.

Donc, si M_k accepte w_n, (A,B) a une solution. Sinon, les paires des groupes 3 et 4 ne peuvent être utilisées. Il n'y a que des solutions partielles (x,y) avec $|y| > |x|$, donc pas de solution. On a donc réduit L_u à PCPM.

Théorème 29 *Le problème "La grammaire context-free G est-elle ambiguë?" est indécidable.*

Preuve On montre que PCP se réduit à ce problème. Soient

$$A = w_1, \ldots ,w_n \text{ et } B = x_1, \ldots ,x_n.$$

Soient $a_1,a_2, \ldots ,a_n$ de nouveaux symboles

$$L_A = \{w_{i_1}w_{i_2}\ldots w_{i_m}a_{i_m}\ldots a_{i_2}a_{i_1} \mid m \geq 1\}$$

$$L_B = \{x_{i_1}x_{i_2}\dots x_{i_m}a_{i_m}\dots a_{i_2}a_{i_1} \mid m \geq 1\}$$

Soit G :

$$\begin{aligned} S &\longrightarrow S_A \mid S_B \\ S_A &\longrightarrow w_iS_Aa_i \mid w_ia_i \ (1 \leq i \leq n) \\ S_B &\longrightarrow x_iS_Ba_i \mid x_ia_i \ (1 \leq i \leq n) \end{aligned}$$

On a $L(G) = L_A \cup L_B$.
G est ambiguë $\Longleftrightarrow$ PCP (A,B) a une solution.

Calculs de machines de Turing

Soit $M = (Q,\Sigma,\Gamma,\delta,q_0,\$,F)$.
Un *calcul valide* de M est un mot :

$$w_1\#\tilde{w}_2\#w_3\#\tilde{w}_4\#\ldots w_n(\text{ou } \tilde{w}_n)\#$$

où les w_i représentent les descriptions instantanées du fonctionnement correct de M :
w_1 description initiale q_0x,
w_n description finale, dans $\Gamma^*F\Gamma^*$,
et $w_i \vdash_M w_{i+1}$.

Théorème 30 *L'ensemble des calculs valides de M est l'intersection de deux langages context-free L_1 et L_2. Des grammaires G_1 et G_2 engendrant ces langages peuvent effectivement être construites.*

Preuve Soit :

$$L_3 = \{y\#\tilde{z} \mid y \vdash_M z\}.$$

On construit un automate à pile qui accepte L_3 par pile vide. Soit ensuite :

$$L_1 = (L_3\#)^*(\{\epsilon\} \cup \Gamma^* F \Gamma^* \#)$$

L_1 est context-free. De même

$$L_2 = q_0 \Sigma^* \#(L_4\#)^*(\{\epsilon\} \cup \Gamma^* F \Gamma^* \#),$$

où $L_4 = \{\tilde{y}\#z \mid y \vdash_M z\}$ est context-free. L'ensemble des calculs valides de M est égal à $L_1 \cap L_2$.

Théorème 31 *Si G_1 et G_2 sont deux grammaires context-free, "$L(G_1) \cap L(G_2) = \emptyset$" est indécidable.*

Preuve Par le théorème précédent, on peut construire, à partir d'une MT M, deux grammaires G_1 et G_2 telles que $L(G_1) \cap L(G_2)$ soit l'ensemble des calculs valides de M.
Alors "$L(G_1) \cap L(G_2) = \emptyset$" $\iff$ "$L(M) = \emptyset$". On a réduit "$L(M) = \emptyset$" à "$L(G_1) \cap L(G_2) = \emptyset$".

Théorème 32 *L'ensemble des calculs non valides (le complémentaire de l'ensemble des calculs valides) d'une MT est context-free.*

Preuve Si w est un calcul non valide, on a l'une des possibilités suivantes :

1. w n'est pas de la forme $x_1\#\ldots\#x_m\#$, où x_i est une configuration de M.
2. x_1 n'est pas initial ($x_1 \notin q_0\Sigma^*$).
3. x_m n'est pas final ($x_m \notin \Gamma^* F \Gamma^*$).
4. $x_i \vdash_M \tilde{x}_{i+1}$ est faux pour un i impair.
5. $\tilde{x}_i \vdash_M x_{i+1}$ est faux pour un i pair.

les mots satisfaisant 1,2,3 sont reconnaisables. Les mots satisfaisant 4,5 sont context-free. L'union de deux context-free et trois rationnels donnent un context-free.

Corollaire 33 *Si G est une grammaire context-free, "$L(G) = \Sigma^*$" est indécidable.*

Preuve On réduit "$L(M) = \emptyset$" à "$L(G) = \Sigma^*$" en associant à une MT M une grammaire qui engendre les calculs non valides de M. Alors "$L(M) = \emptyset$" $\iff$ "$L(G) = \Sigma^*$".

Théorème de Greibach

Analogue du théorème de Rice pour des classes de langages telles que les langages context-free.

Théorème 34 *Soit $\mathcal{C}$ une classe de langages qui est effectivement close par concaténation avec les langages rationnels et union, et pour laquelle "$= \Sigma^*$" est indécidable pour tout Σ assez grand fixé. Soit P une propriété non triviale vraie pour tous les rationnels et préservée par $L \longrightarrow La^{-1}$. Alors P est indécidable pour $\mathcal{C}$.*

Application : $\mathcal{C}$ est la classe des langages context-free.

Preuve Soit $L_0 \subset \Sigma^*$, $\in \mathcal{C}$ qui n'a pas la propriété P. On associe :

$$L \in \mathcal{C} \longrightarrow L_1 = L_0\#\Sigma^* \cup \Sigma^*\#L$$

On a $L_1 \in \mathcal{C}$.
Si $L = \Sigma^*$, $L_1 = \Sigma^*\#\Sigma^*$ est rationnel et a donc la propriété P.
Si $L \neq \Sigma^*$, soit w qui n'est pas dans L. On a $L_1(\#w)^{-1} = L_0$. Donc L_1 n'a pas la propriété P.
Ainsi L_1 a la propriété P ssi $L = \Sigma^*$.
Ainsi "$= \Sigma^*$" pour $\mathcal{C}$ se réduit à la propriété P pour $\mathcal{C}$.

Corollaire 35 *Si G est une grammaire context-free, "$L(G)$ est régulier" est indécidable.*

Fonctions récursives

Ce sont les fonctions calculables par machine de Turing. Si M_k prend en entrée $1^{i_1}01^{i_2}0\ldots01^{i_m}$ et s'arrête avec 1^j, on dit que $f_k(i_1,\ldots,i_m)=j$. Sinon f_k est non définie.

Une fonction *récursive totale* est une fonction calculable par une machine de Turing qui s'arrête toujours avec une suite de 1 entre les #.
Machine de Turing = Programme PASCAL.
On prend un sous ensemble de PASCAL : on ne garde que les structures itératives bornées (boucles FOR). Les fonctions calculables par de tels programmes sont les fonctions *primitives récursives*.

Théorème 36 *Il existe des fonctions récursives totales qui ne sont pas primitives récursives.*

Preuve On liste les programmes du PASCAL réduit qui calculent les fonctions entières d'un

entier (un seul READ, un seul WRITE juste avant END) : P_1, P_2, ...
La fonction $i \longrightarrow P_i(i)+1$ est totale récursive.
Si elle est calculée par P_j, on a $P_j(j) = P_j(j)+1$, contradiction.

Exemple

La fonction d'Ackermann :

$$
\begin{aligned}
&A(0,n) = n+1\\
&A(m,0) = A(m-1,1) \text{ pour } m \geq 1\\
&A(m,n) = A(m-1,A(m,n-1)) \text{ pour } m,n \geq 1
\end{aligned}
$$

On a :

$$
\begin{aligned}
A(0,n) &= n+1\\
A(1,n) &= n+2\\
A(2,n) &\simeq 2*n\\
A(3,n) &\simeq 2^n\\
A(4,n) &\simeq 2 \uparrow\uparrow n \stackrel{\text{def}}{=} \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\}n
\end{aligned}
$$

Une fonction est primitive récursive si elle est obtenue par un nombre fini de compositions

et d'applications du schéma de récursion primitive à partir la fonction constante égale à 0, de la fonction successeur et des projections $P_i(x_1, \ldots ,x_n) = x_i$.
Une récursion primitive est une définition de $f(x_1, \ldots ,x_n)$ par :

$f(x_1, \ldots ,x_n) =$
si $x_n = 0$ **alors** $g(x_1, \ldots ,x_{n-1})$
sinon $h(x_1, \ldots ,x_n,f(x_1, \ldots ,x_{n-1},x_n - 1))$

Proposition 37 *Il existe des fonctions partout définies non récursives.*

Preuve Soit

$$s_i = \max\{0,M(i) \mid M \text{ a } i \text{ états et s'arrête sur } i\}$$

Soit $f(i) = s_i+1$, une fonction totale. Si $f(i) = M(i)$ avec k états pour M, alors $f(k) = M(k) = s_k+1 > M(k)$, contradiction. En fait, $f(i)$ croît plus vite que toute fonction calculable.

Le théorème S_{mn}

Soit f_i la fonction récursive calculée par la machine de Turing de numéro i.

Théorème 38 *Soit* $(x,y) \longrightarrow g(x,y)$ *une fonction récursive. Alors il existe une fonction récursive* totale σ *telle que* $f_{\sigma(x)}(y) = g(x,y)$ *pour tous* x *et* y*.*

Preuve

$$\left.\begin{array}{l} x \longrightarrow \\ y \longrightarrow \end{array}\right\} \boxed{M} \longrightarrow g(x,y)$$

$$M(x)$$

A x on associe le numéro de $M(x) = \sigma(x)$.

Le théorème de la récursion (point fixe)

Théorème 39 *Si* σ *est récursive totale, il existe* x_0 *tel que* $f_{x_0}(x) = f_{\sigma(x_0)}(x)$ *pour tout* x*. (Deux machines de Turing équivalentes).*

Si on modifie toutes les machines de Turing d'une certaine façon, il existe toujours une machine de Turing modifiée qui est équivalente à la machine non modifiée.

Preuve

$$\left.\begin{array}{l} i \longrightarrow \boxed{M} \longrightarrow f_i(i) \longrightarrow \\ x \xrightarrow{\hspace{14em}} \end{array}\right\} \boxed{M_u} \longrightarrow$$

machine de numéro $g(i)$

Pour tous i,x, $f_{g(i)}(x) = f_{f_i(i)}(x)$.
On a g récursive totale. Donc $\sigma \circ g$ est récursive totale calculable par M_j. D'où $\sigma \circ g = f_j$.
Soit $x_0 = g(j)$. Pour tout x on a :

$$f_{x_0}(x) = f_{g(j)}(x) = f_{f_j(j)}(x) = f_{\sigma \circ g(j)}(x) = f_{\sigma(x_0)}(x).$$

Attention : $f_{x_0}(x)$ peut n'être jamais défini.

Théorème 40 *Soit F un système formel de preuves. Il existe une machine de Turing M telle que pour toute entrée j :*

- *M ne s'arrête pas*
- *Il n'existe pas de démonstration que M ne s'arrête pas.*

Preuve

$g(i,j) = \begin{cases} 1 \text{ si preuve que } f_i(j) \text{ ne s'arrête pas} \\ \text{indéfinie sinon} \end{cases}$

Pour calculer g :
énumérer toutes les preuves de F par longueur croissante. Si on en trouve une du non arrêt de $f_i(j)$, on s'arrête en retournant la valeur 1. Sinon on boucle indéfiniment.
Donc g est récursive partielle. Il existe σ telle que $f_{\sigma(i)}(j) = g(i,j)$.
D'après le théorème de récursion, on calcule i_0 tel que

$$f_{i_0}(j) = f_{\sigma(i_0)}(j) = g(i_0,j).$$

Or $g(i_0,j)$ défini $\Longleftrightarrow$ preuve que $f_{i_0}(j)$ indéfini $\Longleftrightarrow$ preuve que $g(i_0,j)$ indéfini.

Conclusion: si F est non contradictoire (on ne peut prouver une assertion et son contraire), $g(i_0,j)$ est indéfini et il n'existe pas de preuve que $g(i_0,j)$ est indéfini. Donc f_{i_0} répond à la question.

COMPLEXITE

Problèmes NP-complets

Complexité polynomiale

On considère les problèmes ayant des algorithmes efficaces :

- machines déterministes
- temps polynomial en fonction de la taille des données

Certains problèmes ne semblent pas résolubles de façon efficace : algorithmes exponentiels, mais ils sont facilement vérifiables :

- "Existe-t-il u tel que $P(u)$?"
  $\longrightarrow$ temps exponentiel.
- "Un u donné est-il solution de P ?"
  $\longrightarrow$ temps polynomial.

Exemples

Problèmes sur les langages rationnels sont polynomiaux.

Problème du circuit hamiltonien est facilement vérifiable.

Problème de la connexité d'un graphe est polynomial.
Problème du circuit eulérien est polynomial.

Pourquoi polynomial ?
Fait d'expérience : en général, polynômes de petit degré.
Non déterminisme :
facilement vérifiable = algorithme non déterministe polynomial.

Définitions
L est **décidable en temps polynomial** ssi il existe une machine M déterministe et un polynôme P tels que :

- $u \in L \iff M$ accepte u.
- Si $|u| \leq n$, alors M avec entrée u effectue au plus $P(n)$ transitions.

L est **décidable en temps polynomial non déterministe** ssi il existe une machine M non déterministe et un polynôme P tels que :

- $u \in L \iff$ il **existe** une suite de choix amenant à un état terminal.
- Si $|u| \leq n$, alors aucune suite de choix n'a plus de $P(n)$ transitions.

1$^{\text{er}}$ cas : $L \subset \mathcal{P}$
2$^{\text{ème}}$ cas : $L \subset \mathcal{NP}$

Pourquoi un polynôme?
stabilité : indépendant du nombre de bandes :

- si déterministe en $P(n)$ avec k bandes, alors déterministe en $P^2(n)$ avec 1 bande.
- idem pour non-déterministe.
- si non-déterministe en $P(n)$, alors déterministe en $c^{P(n)}$ = exponentiel.

Question: $\mathcal{P} = \mathcal{NP}$?

La classe $\mathcal{NP}$

Formalisation de $\mathcal{NP}$ = polynomialement vérifiable.
Si A et B sont des alphabets et $L \subset A^* \times B^*$, on code les couples (u,v) par $\#u\#v\#$ pour faire un langage.
On dit que $L \subset A^* \times B^*$ est à *sections polynomiales* ssi il existe un polynôme P tel que :

$$(u,v) \in L \Longrightarrow |v| \leq P(|u|)$$

La *projection* de L est l'ensemble des u tels que $(u,v) \in L$.

Théorème 41 *Soit $L \subset A^*$. Les conditions suivantes sont équivalentes :*

1. *L est dans $\mathcal{NP}$.*
2. *il existe B et $L' \subset A^* \times B^*$ à sections polynomiales tels que L' est dans $\mathcal{P}$ et L est projection de L'.*

Preuve

1) $\Longrightarrow$ 2) L est dans $\mathcal{NP}$, reconnu par M non

déterministe de complexité P. A chaque étape, k choix possibles par M. Soit $B = \{1,2,\ldots,k\}$ et :
$L' = \{(u,(i_1,i_2,\ldots,i_m)) \in A^* \times B^* \mid m \leq P(|u|)$ et $M(u)$ amène à un état final avec la suite de choix $(i_1,\ldots,i_m)\}$.

- L' est à sections polynomiales
- L est projection de L'
- L' est dans $\mathcal{P}$: on déroule M de façon déterministe en étant guidé par les choix et on constate si on arrive à un état final à la fin.

2) $\Longrightarrow$ 1)
Soit $L' \subset A^* \times B^*$ à sections polynomiales dans $\mathcal{P}$ (avec Q). Soit L sa projection.

$$(u,v) \in L' \Longrightarrow |v| \leq P(|u|)$$

On construit une machine non déterministe qui reconnaît L :
au départ $\#u\#$.
Sur une bande annexe, compteur de taille $P(|u|)$ (fait en temps polynomial).
La machine peut écrire n'importe quelle suite

de symboles de B de longueur au plus $P(|u|)$ à droite du # (non déterminisme).
On a donc $\#u\#v\#$.
Puis on teste l'appartenance à L'.
Temps : $P(|u|) + Q(|u| + 2 + P(|u|)) =$ polynomial non déterministe.

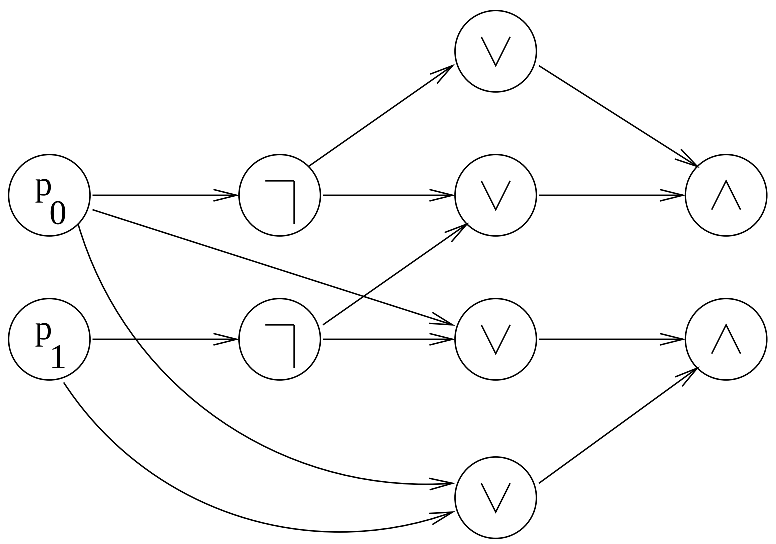
Un mot v tel que $(u,v) \in L'$ est appelé un *certificat* de x. Il certifie que x appartient bien à L.
Exemple : Circuits hamiltoniens (Exercice).

Circuits booléens

Un *circuit booléen* est un graphe sans cycle représentant une fonction booléenne de :

$$\{0,1\}^n \longrightarrow \{0,1\}^m.$$

Les sommets d'entrée sont des variables ou des constantes 0 ou 1. Les autres sommets sont des portes $\vee$, $\wedge$ (2 prédécesseurs), ou $\neg$ (1 prédécesseur). Les sommets de sortie sont sans successeur. La taille du circuit est le nombre de sommets.



Rapport avec les machines de Turing

Soit M une machine de Turing déterministe à une bande. Tout est codé en binaire :
alphabet $B \subset \{0,1\}^s$, $\#$ est 0^s, états $\subset \{0,1\}^s$.
Cases de la bande : $0,1,\ldots,m,\ldots$.
En m pas de calcul, on ne peut pas dépasser la case m.
On fabrique un circuit C_m de taille proportionnelle à m^2, dont les entrées codent une configuration d'entrée de M de taille $\leq m$ et dont les sorties codent la configuration obtenue après m étapes de calcul.

Variables :

- $p[i,l], 0 \leq i \leq m, 1 \leq l \leq s$
  codent le contenu de la $i^{\text{ème}}$ case pour i fixé.
- $q[l], 1 \leq l \leq s$
  codent l'état de la machine
- $h[i], 0 \leq i \leq m$
  vaut 1 si la tête est sur la case i, 0 sinon.
- e vaut 1 ssi le calcul est arrêté.

On a ainsi $(m+1)(s+1)+(s+1)$ variables.

Lemme 42 *Il existe K dépendant de M et C_m^1 de taille Km qui calcule la configuration suivante à partir d'une configuration.*

Preuve Soient $\hat{p}[i,l]$, $\hat{q}[l]$, $\hat{h}[i]$ et $\hat{e}$ les valeurs prises après la transition. Il faut les calculer à partir des $p[i,l]$, $q[l]$, $h[i]$ et e.

$$\hat{p}[i,l],1 \leq l \leq s \text{ et } \hat{h}[i]$$

ne dépendent que de :
$p[i-1,l]$, $p[i,l]$, $p[i+1,l],1 \leq l \leq s$
$h[i-1]$, $h[i]$, $h[i+1]$
$q[l],1 \leq l \leq s$
e
et de M, d'où taille K_1. Soient $K_1 m$ portes.

$$\hat{q}[l]$$

dépend des mêmes, d'où $K_2 m$ portes.

$$\hat{e}$$

dépend de e, $\hat{q}[l]$, des $p[i,l]$ et $h[i]$.
Ce circuit a pour taille Km. De plus il est calculable en temps polynomial à partir de M et m.
On empile les C^1_m pour rendre compte des m étapes de calcul, d'où :

– C_m de taille Km^2.
– calculé en temps polynomial

Exemple

$$(0,1) \longrightarrow (0,1,R)$$
$$(0,\#) \longrightarrow (1,\#,L)$$

M reconnaît 1^n.
Ici $s = 1$. Soit $m = 1$.
variables : p_0, p_1, q, h_0, h_1 et e.
$\hat{p}_0 = p_0$, $\hat{p}_1 = p_1$
$\hat{h}_0 = \neg p_1 \wedge h_1$, $\hat{h}_1 = p_0 \wedge h_0$
$\hat{q} = h_1 \wedge \neg p_1 \wedge \neg q$
$\hat{e} = e \vee \hat{q} \vee (h_0 \wedge \neg p_0 \wedge \neg q)$
pour $C_1^1 = C_1$. D'où le circuit booléen.

Problèmes NP-complets

Un circuit est *satisfiable* s'il existe une distribution de vérité donnant la valeur 1 à toutes ses sorties.

Proposition 43 *L'ensemble des codes des circuits satisfiables est dans* $\mathcal{NP}$.

Preuve Soit $L' = \{(c,\sigma)\}$ où

- c est le code d'un circuit C.
- σ est une distribution de vérité sur les variables de c.
- la valeur de $C(\sigma)$ vaut 1 pour toutes les sorties de C.

On a :

- L' est à sections polynomiales.
- L' est dans $\mathcal{P}$.
- L est projection de L'.

On dit que L' est *polynomialement réductible* à L ssi il existe une machine de Turing bornée en temps polynomialement telle que sur chaque entrée u elle produit une sortie v telle que $u \in L'$ ssi $v \in L$.

Proposition 44 *Soit L' un langage polynomialement réductible à L.*

- *L' est dans $\mathcal{NP}$ si L est dans $\mathcal{NP}$.*
- *L' est dans $\mathcal{P}$ si L est dans $\mathcal{P}$.*

Preuve On prouve le deuxième point. Supposons que la réduction est bornée en temps par $P_1(n)$ et que L est accepté en temps $P_2(n)$. Soit u une entrée de L' de longueur n. On construit v. On a $|v| \leq P_1(n)$, car on peut afficher au plus un caractère par mouvement. On teste si $v \in L$ en temps $P_2(P_1(n))$. On teste ainsi si $u \in L'$ en temps $P_1(n) + P_2(P_1(n))$.

Théorème 45 (Cook, 1971) *Un langage* $L \subset B^*$ *est dans* $\mathcal{NP}$ *ssi il existe une fonction calculable en temps polynomial* $u \longmapsto C(u)$, *où* $C(u)$ *est un circuit booléen, telle que* $u \in L \iff C(u)$ *est satisfiable.*

On réduit polynomialement tout problème dans $\mathcal{NP}$ au problème de la satisfiabilité des circuits booléens.

Preuve Soit L dans $\mathcal{NP}$. Il est projection de L' dans $\mathcal{P}$. Soit M qui décide en temps polynomial

$$\{\#u\#v \mid (u,v) \in L'\}$$

et (C_m) la suite des circuits associés.

On a $|v| \leq P(|u|)$ et le temps de M sur $\#u\#v$ est majoré par $Q(|u|)$.

Soit $C'(u)$ le circuit obtenu à partir de $C_{Q(|u|)}$ obtenu en fixant les variables p correspondant au début $\#u\#$. Le circuit $C'(u)$ calcule les valeurs des variables $q_1, \ldots ,q_s$ codant l'état obtenu connaissant v, $|v| \leq P(|u|)$ sur l'entrée $\#u\#v$ après $Q(|u|)$ pas de calcul. On prend

$C(u)$ modifié de $C'(u)$ avec une seule sortie qui vaut 1 si l'état est acceptant et 0 sinon (de toute façon M est arrêtée).
$C(u)$ satisfiable $\iff \exists v \mid \#u\#v$ accepté par $M \iff u \in L$.
Il reste à voir que $\{u \mid C(u) \text{ satisfiable}\}$ est $\mathcal{NP}$:

$$u \longrightarrow \text{calcul de } C(u) \text{ (pol. déterministe)}$$

$$C(u) \text{ satisfiable (pol. non déterministe)}$$

d'où le théorème.

Problème NP-complet
Définition : L est *NP-complet* si :
- Il est dans $\mathcal{NP}$.
- Tout problème dans $\mathcal{NP}$ est polynomialement réductible à L.

Théorème 46 *La satisfiabilité des circuits booléens est NP-complet.*

Autres problèmes NP-complets

Clauses sous forme normale conjonctive FNC : de la forme $E_1 \wedge E_2 \wedge \cdots \wedge E_k$. Chaque E_i est une clause de la forme $\alpha_{i_1} \vee \alpha_{i_2} \vee \cdots \vee \alpha_{i_{r_i}}$, où α_{ij} est un littéral de la forme x ou $\neg x$, noté $\bar{x}$.
Exemple $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4) \wedge \bar{x}_3$.

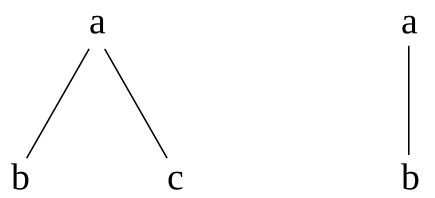
Problème :
Données : ensemble fini de clauses.
Question : existe-t-il une distribution de valeurs de vérité satisfaisant toutes les clauses?

Théorème 47 *Le problème de la satisfiabilité des clauses est NP-complet.*

Preuve Il est dans $\mathcal{NP}$ (évident).
La satisfiabilité des circuits est polynomialement réductible à la satisfiabilité d'un ensemble de clauses.
Au départ, on a un circuit de taille n avec k sorties et m variables.

On fabrique au plus $3n$ clauses avec n variables. On ajoute une variable par noeud interne du graphe du circuit et à chaque noeud, on transforme :

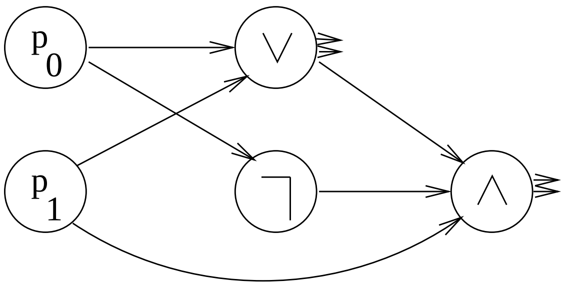
si $a = \wedge$	$p_a \longleftrightarrow p_b \wedge p_c$	$\neg p_a \vee p_b$ $\neg p_a \vee p_c$ $\neg p_b \vee \neg p_c \vee p_a$
si $a = \vee$	$p_a \longleftrightarrow p_b \vee p_c$	$\neg p_a \vee p_b \vee p_c$ $\neg p_b \vee p_a$ $\neg p_c \vee p_a$
si $a = \neg$	$p_a \longleftrightarrow \neg p_b$	$\neg p_a \vee \neg p_b$ $p_a \vee p_b$



circuit satisfiable $\Longrightarrow$ chaque clause satisfiable (feuilles $\longrightarrow$ racine)
chaque clause satisfiable $\Longrightarrow$ distribution satisfaisant le circuit (racine $\longrightarrow$ feuilles).

Calcul en temps linéaire.

Exemple



On obtient 8 clauses, 5 variables.

Remarque : chaque clause contient au plus 3 variables, d'où :

Clauses 3-FNC

Données : ensemble fini de clauses comportant au plus 3 variables.

Théorème 48 *Le problème de la satisfiabilité des clauses* 3*-FNC est NP-complet.*

Remarque : on peut exiger que chaque clause contient exactement 3 variables.

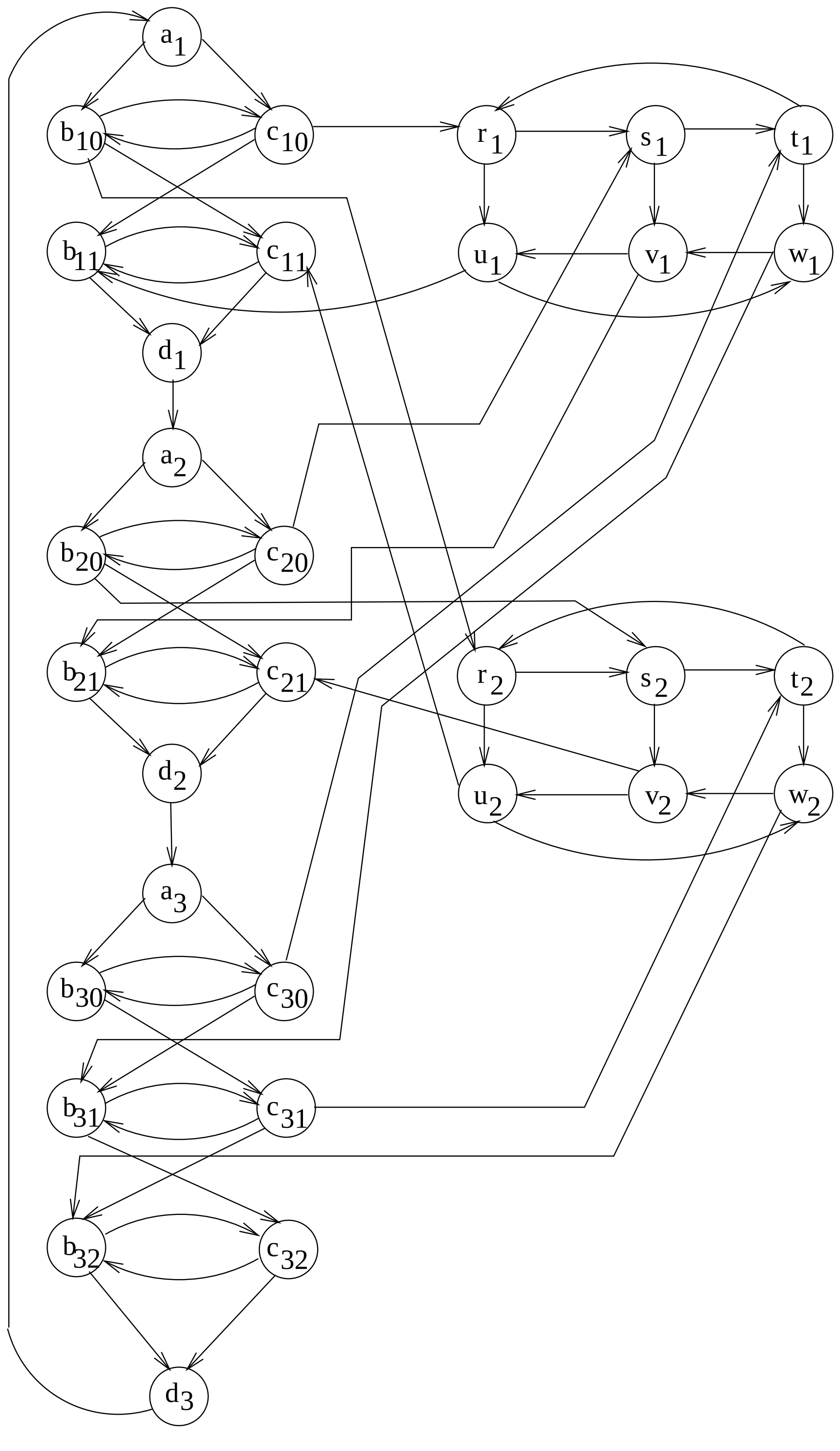
$$u \longrightarrow u \vee y \vee z, u \vee y \vee \neg z, u \vee \neg y \vee z, u \vee \neg y \vee \neg z$$
$$u \vee v \longrightarrow u \vee v \vee y, u \vee v \vee \neg y$$

avec de nouvelles variables y,z.

Circuits hamiltoniens
Données : graphe orienté G
Question : G admet-il un circuit hamiltonien ? (un circuit qui visite chaque sommet une fois et une seule).

Théorème 49 *Le problème du circuit hamiltonien est NP-complet.*

Preuve Le problème du circuit hamiltonien est dans $\mathcal{NP}$. On réduit 3-FNC aux circuits hamiltoniens en temps polynomial.


a1
b10
c10
r1
s1
t1
b11
c11
u1
v1
w1
d1
a2
b20
c20
b21
c21
r2
s2
t2
d2
u2
v2
w2
a3
b30
c30
b31
c31
b32
c32
d3

Remarque : On a le même résultat pour les graphes non-orientés :
On réduit le problème du circuit hamiltonien pour les graphes orientés au problème du cycle hamiltonien pour les graphes non orientés. Si G est un graphe orienté, on construit G' un graphe non orienté tel que si s est un sommet de G, G' admet les sommets s_0,s_1,s_2 avec un arc (s_0,s_1), un arc (s_1,s_2) et s'il existe une flèche $s \longrightarrow t$ dans G, un arc (s_2,t_0). Les sommets d'indice 1 ont seulement deux arcs adjacents. Les indices des sommets d'un cycle de G' doivent être 0,1,2,0,1,2 ... ou l'inverse.
G admet un circuit hamiltonien ssi G' admet un cycle hamiltonien.

Problème du voyageur de commerce
Données : Un ensemble fini S, une fonction distance $d : S \times S \longrightarrow \mathbb{N}$, un entier k.
Question : Existe-t-il un circuit hamiltonien $(x_1,x_2,\ldots,x_n)$ avec $\sum d(x_i,x_{i+1}) \leq k$?

Théorème 50 *Le problème du voyageur de commerce est NP-complet.*

Preuve Il est dans $\mathcal{NP}$. Le problème du cycle hamiltonien non orienté y est réductible :
Si G a n sommets, $d(x_i,x_j) = 1$ s'il y a une arête, $d(x_i,x_j) = 2$ sinon et $k = n$.
Problème du recouvrement par des sommets
Données : G un graphe non orienté, k un entier.
Question : Existe-t-il $A \subset S$ l'ensemble des sommets de G, de cardinal au plus k tel que chaque arête de G a au moins une extrémité dans A?

Théorème 51 *Le problème du recouvrement par des sommets est NP-complet.*

Preuve Il est dans $\mathcal{NP}$.
On réduit 3-FNC polynomialement à ce problème.
Si on a les clauses $C_1,C_2,\ldots,C_q$ et $C_i = \alpha_{i_1} \vee \alpha_{i_2} \vee \alpha_{i_3}$, on construit le graphe G de sommets :

$$S = \{(i,j) \mid 1 \leq i \leq q, j = 1,2,3\}$$

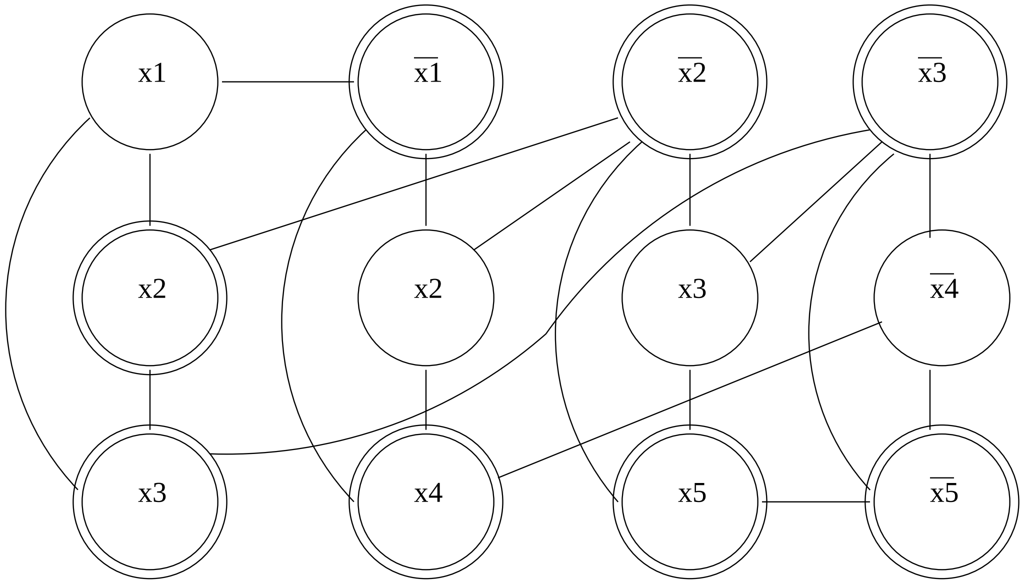
et d'arêtes :

$$\begin{array}{ll} (i,j)\text{—}(i,k) & \text{si } j \neq k \\ (i,j)\text{—}(k,l) & \text{si } \alpha_{i_j} = \neg\alpha_{k_l} \end{array}$$

Alors satisfiabilité $\Longleftrightarrow$ recouvrement par $2q$ sommets.
Si la formule est satisfiable, on fixe un assignement tel que la formule soit vraie. On selectionne un litéral par clause de valeur 1. Soit A l'ensemble des sommets qui ne correspondent pas aux litéraux selectionnés. Alors A est un recouvrement de taille $2q$.
Inversement si A est un recouvrement de taille $2q$, on assigne la valeur 1 aux litéraux correpondants à des sommets qui ne sont pas dans le recouvrement. La formule est alors satisfaite.

Exemple

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$
$$\wedge\ (\bar{x}_2 \vee x_3 \vee x_5) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5)$$



Le problème du sac à dos
Données : Un ensemble fini E d'entiers positifs, un entier s.
Question : Peut-on réaliser s comme somme d'éléments de E?

Théorème 52 *Le problème du sac à dos est NP-complet.*

Preuve Il est dans $\mathcal{NP}$. On réduit polynomialement 3-FNC à ce problème.
Si les clauses sont $C_1, C_2, \ldots, C_m$ et les variables $x_1, \ldots, x_n$.

$$\begin{aligned} k_i &= 8^{i-1}, 1 \leq i \leq n \\ l_j &= 8^{n+j-1}, 1 \leq j \leq m \end{aligned}$$

et n_i et m_i, $1 \leq i \leq n$ avec :

$$\begin{aligned} n_i &= k_i + \sum \{l_j \mid x_i \text{ apparaît dans } C_j\} \\ m_i &= k_i + \sum \{l_j \mid \bar{x}_i \text{ apparaît dans } C_j\} \end{aligned}$$

$$\begin{aligned} E &= \{m_i, n_i, l_j, 2l_j \mid 1 \leq i \leq n \text{ et } 1 \leq j \leq m\} \\ s &= \sum k_i + 4(\sum l_j) \end{aligned}$$

Autres problèmes

- Nombre chromatique
- Ordonnancement
- Programmation linéaire

Algorithmes randomisés

Test de Rabin en 1976 : identifier le nombre $2^{400} - 593$ comme le plus grand nombre premier inférieur ou égal à 2^{400} en quelques secondes et ce avec une probabilité d'erreurs inérieure à 2^{-100}.

Idée : Supposons que l'on veuille tester si un polynôme en n variables $p(x_1,x_2,\ldots,x_n)$ est identiquement nul.

- On tire un vecteur aléatoire $(r_1,r_2,\ldots,r_n)$;
- On calcule $p(r_1,r_2,\ldots,r_n)$;
- Si $p(r_1,r_2,\ldots,r_n) \neq 0$, on arrête avec $p \neq 0$.
- Sinon on retire un autre vecteur.

Au bout d'un certain nombre de tirages, on conclut que $p = 0$. La probabilité d'erreur est faible.

Pour un problème particulier, on peut choisir un entier k indépendant de la taille du problème, et aussi grand qu'on veut, tel que l'on peut apporter une réponse rapide et juste avec une probabilité supérieure à $1 - 2^{-k}$.

Tout algorithme randomisé est tel qu'on peut rendre la probabilité d'erreur aussi petite que l'on veut.

On peut rendre la probabilité d'erreur plus faible que la probabilité d'obtenir une réponse incorrecte suite à une erreur hardware.

Premier exemple

Donnée : Un polynôme $p(x_1,x_2,\ldots,x_n)$ en plusieurs variables.
Question : A-t-on $p = 0$?

Exemple :

$$p = \mathtt{det}\begin{pmatrix} 1 & x_1 & \ldots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \ldots & x_n^{n-1} \end{pmatrix} - \sum_{i<j}(x_i - x_j) = 0?$$

pour $n = 100$, degré 5000.

Lemme 53 *Soit p est un polynôme en n variables à coefficients dans A et non identiquement nul. Si I désigne une partie de A de cardinal $|I| \geq c\deg(Q)$, alors le nombre de zéros de p dans $I \times I \ldots \times I$ est au plus $\frac{|I|^n}{c}$.*

L'algorithme randomisé

1. Choisir I tel que $|I| \geq c \deg(Q)$ avec $c > 2$;
2. Choisir un entier k tel que c^{-k} soit assez petit, par exemple tel que $c^{-k} < 2^{-400}$;
3. Tirer aléatoirement k variables $(x_1, x_2, \ldots, x_n)$ dans $I \times I \ldots \times I$.
   Si l'une d'elle est un zéro de p alors $p \neq 0$, sinon arrêter en disant que $p = 0$.

D'après le lemme, la probalilité d'erreur est inférieure ou égale à $(\frac{1}{c})^k$.

Deuxième exemple nombres premiers

Rabin (1976)
Solovay et Strassen (1977)
Application à la cryptographie à clef publique (RSA).

Le principe : Pour tester si un nombre n est premier, on prend k entiers x distribués uniformément entre 2 et $n-1$. Pour chaque x on teste si une propriété $T(x,n)$ est vraie. Un couple (x,n) qui rend vraie $T(x,n)$ est appelé un *témoin*.

La propriété $T(x,n)$ doit vérifier :

- Si n est premier, $T(x,n)$ est faux pour tout x;
- Si n n'est pas premier, $T(x,n)$ est vraie pour certaine proportion, de taille fixée et strictement positive, des entiers x entre 2 et $n-1$;
- On peut tester $T(x,n)$ rapidement. (Ici en $O(\log(n))$).

Le test de primalité de Miller-Rabin

1. Calculer r,m tels que $n = 1 + 2^r m$ avec m impair;
2. Tirer aléatoirement k entiers x avec $1 < x < n-1$;
3. Pour i allant de 0 à r calculer $x_i = x^{m2^i}$;
4. Pour chaque x tester si $T(x,n)$ est vraie, où $T(x,n)$ est vraie si $x^{n-1} \neq 1$ (i) ou il existe un i avec $1 \leq i \leq r$ tel que $x_i = 1$ et $x_{i-1} \neq -1$ modulo n. (ii)
5. Si $T(x,n)$ est vraie pour un x, retourner Composé, sinon retourner Premier.

Proposition 54 *Si n est un entier impair non premier plus grand que 4, au moins $3/4$ des entiers x possibles vérifient soit (i) soit (ii). Si n est premier, aucun x entre 2 et $n-1$ ne vérifie (i) ou (ii).*

La probabilité de réponse fausse est donc inférieure ou égale à $(1/4)^k$.
Exercice : Montrer que $T(x,n)$ se calcule en un temps polynomial en $\log(n)$.

Le cryptosystème RSA

- On choisit deux grands nombres premiers p et q (avec 100 chiffres décimaux chacun);
- On calcule $n = pq$.
- On choisit un entier impair d premier avec $\varphi(n) = (p-1)(q-1)$;
- On calcule l'inverse e de d modulo $\varphi(n)$;
- On publie dans l'annaire la clé publique RSA pour Bob (e, n); Bob garde secrète sa clé secrète de décodage d;
- Alice veut faire connaître à Bob un message M;
- Elle envoie $C = M^e$ modulo n;
- Bob calcule C^d modulo $n = M$.

La classe $\mathcal{RP}$

Soit $L \subset A^*$ un langage.
On dit que L est dans $\mathcal{RP}$ ssi

1. Il existe B et $L' \subset A^* \times B^*$ à sections polynomiales (polynôme p) tels que L' est dans $\mathcal{P}$ (polynôme q) et L est projection de L'.
2. Si $u \in L$, le nombre de certificats v tels que (u,v) est dans L' est au moins la moitié du nombre de mots v possibles. (On peut remplacer moitié par une fraction fixée α avec $0 < \alpha < 1$).

Comparer avec la définition de L est dans $\mathcal{NP}$:

- Il existe B et $L' \subset A^* \times B^*$ à sections polynomiales tels que L' est dans $\mathcal{P}$ et L est projection de L'.

On a donc

$$\mathcal{P} \subset \mathcal{RP} \subset \mathcal{NP}$$

Algorithme randomisé pour la classe $\mathcal{RP}$

On veut tester si un mot u appartient à un langage L qui est dans $\mathcal{RP}$.

1. Générer aléatoirement v de longueur $p(|u|)$;
2. Tester si (u,v) est dans L' (en temps polynomial);
3. Répéter ces deux étapes k fois au total.
   Si (2) est vrai pour au moins un tirage, sortir avec $u \in L$ sûr.
   Sinon sortir avec $u \notin L$.

La probabilité d'erreur est inférieure ou égale à $(1-\alpha)^k$.

Quelques remarques sur les limites de RP

Il est difficile de trouver des algorithmes randomisés pour lesquels il n'existe pas d'algorithme polynomiaux.
On pense que $RP \neq NP$ et $RP = NP$ ssi il existe un problème NP-complet dans RP. Il est donc improbable de trouver un algorithme randomisé pour un problème NP-complet.
Il y a peu de problème "naturels" que l'on pense être dans $NP - P$ et qui n'ont pas été prouvés être NP-complets.

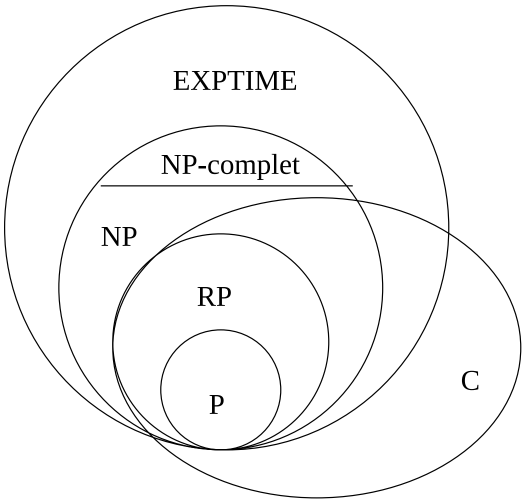
Proposition 55 *Si un langage est décidable en temps polynomial randomisé (dans $\mathcal{RP}$), alors il est décidable par des circuits de tailles polynomiales.*

C'est-à-dire, il existe une suite de circuits (C_n) et un polynôme p tels que C_n permet de tester si un mot u de taille n est dans le langage et la taille de C_n est inférieure ou égale à $p(n)$.

On a vu que si un langage est dans la classe $\mathcal{P}$, il est décidable par des circuits de tailles polynomiales. (La réciproque est fausse).

Preuve : Référence [3] p. 154.

Petit dessin pour les classes de complexité



On note $\mathtt{C}$ la classe des problèmes qui ont des petits circuits, c'est-à-dire qui sont décidables par des circuites de taille polynomiale.
On note $\mathtt{EXPTIME}$ la classe des problèmes qui sont décidables en temps $c^{p(n)}$ où p est un polynôme.

Ce qui est connu :
- $\mathtt{P}$ est inclus strictement dans $\mathtt{EXPTIME}$;
- $\mathtt{C}$ n'est pas inclus dans $\mathtt{EXPTIME}$.