COMPLEXITÉ

Frédéric Olive

CMI - bureau R328

frederic.olive@lif.univ-mrs.fr

http://www.lif.univ-mrs.fr/~olive

04 13 55 13 16

2011 / 12

Chapitre 1 : Calculabilité

Qu'est-ce qu'un algorithme ? Quels sont les problèmes susceptibles d'un traitement automatisé ? Quels sont les problèmes calculables par une machine ? Ce sont là différentes formulations de la question à laquelle la *théorie de la calculabilité* tente de répondre.

Il apparait incontournable d'évoquer cette problématique, d'autant que les concepts sur lesquels elle s'articule sont aussi au cœur de la *théorie de la complexité*, qui est le véritable objet de ce cours.

Pour donner un sens précis aux questions évoquées plus haut, il nous faudra formaliser les notions de *problème* et de *calcul*.

Nous évoquerons alors la *Thèse de Church*, qui affirme l'adéquation de cette formalisation avec la notion intuitive que l'on cherche à capturer : celle de *problèmes calculables*.

Un premier acquis de cette modélisation est la possibilité de prouver rigoureusement l'existence de problèmes non calculables. Nous conclurons le chapitre par la présentation de ce résultat.

Quelques problèmes

SAT

Entrée : Une formule Φ du calcul propositionnel ;

Question : Φ est-elle satisfaisable ?

CYCLE HAMILTONIEN

Entrée : Un graphe G ;

Question : G comporte-t-il un cycle hamiltonien ?

DÉCOMPOSITION EN FACTEURS PREMIERS

Entrée : Un entier n ;

Sortie : la décomposition de n en facteurs premiers.

EXPRESSION RÉGULIÈRE

Entrée : un langage régulier L ;

Sortie : une expression régulière qui dénote L.

Un “non-problème”

RÉGULARITÉ

Entrée : Un langage L ;

Question : L est-il régulier ?

Caractéristiques

- Les problèmes considérés sont des problèmes *génériques* (infinité d'instances).
- Les entrées et sorties de ces problèmes sont des objets :
  – *symboliques* (abstraits, mathématiques...) ;
  – *finis* (*i.e.* qui admettent une représentation finie).

Plusieurs grandes familles de problèmes

- Problèmes d'*évaluation* (on attend le calcul d'une solution).
- Problèmes de *décision* (on attend une réponse oui/non)
- Problèmes d'*optimisation*, d'*approximation*, d'*énumération*, ...

Représentation

Dans tous les cas, la question de la *représentation* des objets manipulés est cruciale.

Par exemple, un algorithme d'addition des entiers n'a de sens que relativement à une représentation donnée des entiers.

D'où l'idée que les entrées de nos problèmes sont des représentations des objets abstraits évoqués plus haut, plutôt que ces objets eux mêmes.

Comme la notion de représentation se ramène simplement à celle de mot sur un alphabet, on aboutit assez naturellement à la formalisation qui suit.

Formalisation des problèmes

(On se restreint aux cas de l'évaluation et de la décision.)

Le *codage* d'un problème d'évaluation π c'est la donnée :
– d'un codage des entrées de π en terme de mots sur un alphabet Σ ;
– d'une fonction $f_\pi : \Sigma^* \to \Sigma^*$.
Une instance de π est alors la donnée d'un mot $w \in \Sigma^*$ assorti de la question : *que vaut* $f_\pi(w)$ *?*

Le *codage* d'un problème de décision π c'est la donnée :
– d'un codage des entrées de π en terme de mots sur un alphabet Σ ;
– d'une partie $L_\pi \subseteq \Sigma^*$.
Une instance de π est alors la donnée d'un mot $w \in \Sigma^*$ assorti de la question : *a-t-on* $w \in L_\pi$ *?*

Exemple

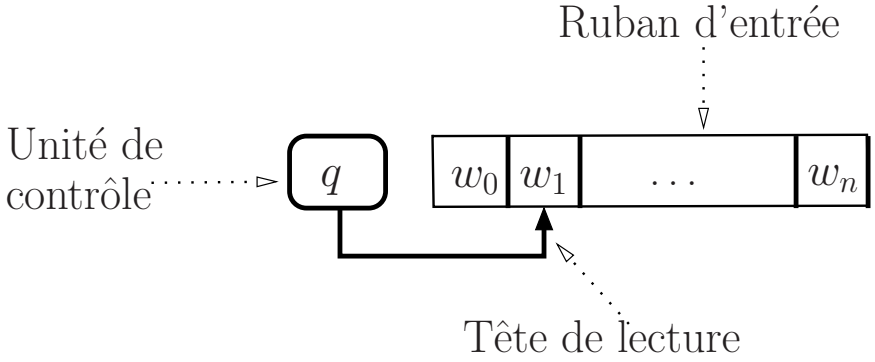
On considère le problème de décision suivant :

PARITÉ
Entrée : Un entier n ;
Question : n est-il pair

Codage sur $\Sigma = \{0,1\}$:
$L_{\text{PARITÉ}} = \{n_k \dots n_1 n_0 \in \Sigma^* \ : \ (k = 0 \text{ ou } n_k = 1) \text{ et } n_0 = 0\}$

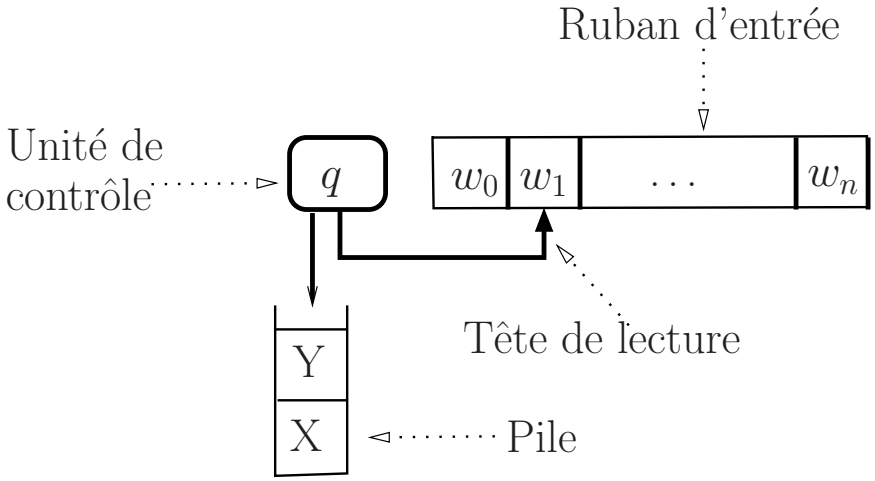
Modèles de calcul

Automates finis



$\{a^n b^n,\ n \in \mathbb{N}\}$ n'est reconnu par aucun automate fini.
Pourtant, il est calculable.

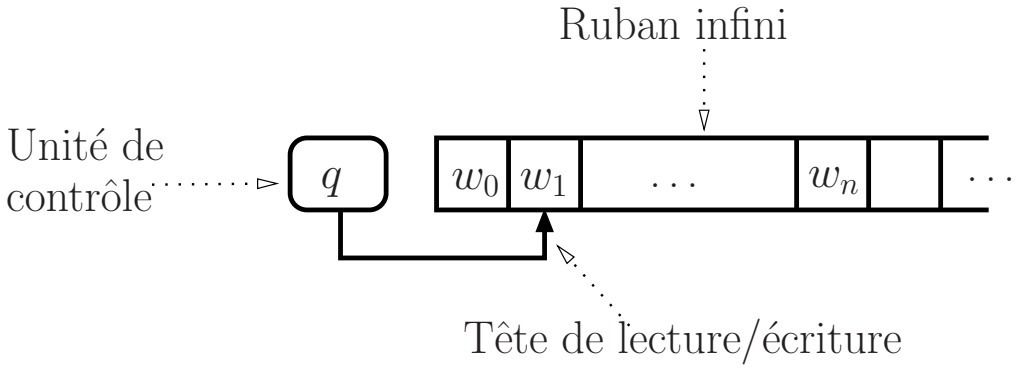
Automates à pile



$\{a^n b^n c^n,\ n \in \mathbb{N}\}$ n'est reconnu par aucun automate à pile.
Pourtant, il est calculable.

Ces modèles de calcul ne capturent donc pas la notion de calculabilité. On va chercher à les enrichir.

Machines de Turing



À chaque pas de calcul, la machine :
– lit le symbole courant ;
– le remplace (éventuellement) par un nouveau symbole ;
– se déplace d'une case à gauche, à droite ou reste immobile ;
– change d'état.

Formellement

Une MT est un 6-uplet $\mathscr{M} = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$, où :

– Q est ensemble fini d'états contenant au moins trois états spéciaux :
  ◦ q_0 : l'état initial ;
  ◦ q_a : l'état d'acceptation (c'est un état final) ;
  ◦ q_r : l'état de refus (final lui aussi) ;
– Σ est l'alphabet d'entrée, sur lequel agit la machine ;
– Γ est l'alphabet de travail. Il contient Σ et au moins un symbole spécial, B, appelé caractère « blanc » ;
– $\delta : \Gamma \times Q \to \Gamma \times Q \times \{\leftarrow, \downarrow, \rightarrow\}$ est la fonction de transition.

Configurations

Le calcul d'une MT se définit à l'aide des notions de *configuration* et de *dérivation* entre configurations.

Une configuration contient toute l'information nécessaire au calcul, à savoir : l'**état** de la machine, le **contenu** du ruban, la **position** de la tête de lecture. Cette information est représentée sous la forme

$$(q, \mathrm{B}ab\underline{b}aabba\mathrm{B}),$$

qui s'interprète comme suit :
– la MT est dans l'état q,
– son ruban contient le mot B*abbaabba*B suivi d'une infinité de B et
– sa tête de lecture est positionnée sur le deuxième « a ».

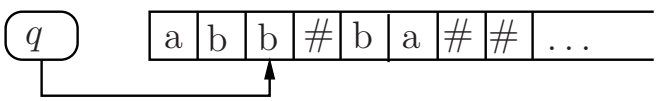
Ainsi, une configuration est la donnée d'un couple

$$(q, w), \text{ où } q \in Q \text{ et } w \text{ est un mot sur } \Gamma$$

dont une lettre est soulignée.

Exemple

La configuration :



est représentée par le couple :

$$(q, ab\underline{b}\mathrm{B}bba\mathrm{B}).$$

Dérivation en un pas

Soient C, C' deux configurations d'une machine de Turing $\mathscr{M}$. On dit que C' *dérive en un pas* de C si C' est la configuration obtenue à partir de C en un pas de calcul de $\mathscr{M}$. On note alors :

$$C \vdash_{\mathscr{M}} C'.$$

EXEMPLE. Si la machine $\mathscr{M}$ comporte les transitions

$$\delta(a,p) = (b,q,\rightarrow) \text{ et } \delta(b,q) = (b,q,\leftarrow),$$

alors

$$(p, \mathrm{B}abb\underline{a}ba) \vdash_{\mathscr{M}} (q, \mathrm{B}abbb\underline{b}a)$$

et

$$(q, \mathrm{B}abbb\underline{b}a) \vdash_{\mathscr{M}} (q, \mathrm{B}abb\underline{b}ba)$$

Plus généralement, si $\mathscr{M}$ comporte la transition

$$\delta(x,p) = (y,q,\mathsf{mvt}),$$

où $\mathsf{mvt} \in \{\leftarrow, \downarrow, \rightarrow\}$, et si C est la configuration

$$(p, \mathrm{B}u_1 \ldots u_{k-1}\underline{x}u_{k+1}\ldots u_n),$$

alors :

$$C \vdash_{\mathscr{M}} \begin{cases} (q, \mathrm{B}u_1 \ldots \underline{u}_{k-1} y u_{k+1} \ldots u_n) & \text{si } \mathsf{mvt} = \leftarrow \\ (q, \mathrm{B}u_1 \ldots u_{k-1} \underline{y} u_{k+1} \ldots u_n) & \text{si } \mathsf{mvt} = \downarrow \\ (q, \mathrm{B}u_1 \ldots u_{k-1} y \underline{u}_{k+1} \ldots u_n) & \text{si } \mathsf{mvt} = \rightarrow \end{cases}$$

Dérivation et calcul

Une *dérivation de $\mathscr{M}$ à partir de C* est une suite (finie ou infinie) $(C_i)_i$ de configurations telle que :

$$C_0 = C \text{ et } \forall i \ : \ C_i \vdash_{\mathscr{M}} C_{i+1}.$$

Un *calcul de $\mathscr{M}$ à partir de C* est une dérivation maximale à partir de C.

De plus, si l'une des configurations du calcul est dans un état final (q_a ou q_r), alors cette suite est finie et cette configuration est la dernière de la suite. On dit alors que le calcul *converge*, *termine*, ou *s'arrête*. Si la suite est infinie, on dit que le calcul *diverge*.

Si le calcul de $\mathscr{M}$ s'arrête dans l'état q_a, on dit qu'il est *acceptant*.

Pour tout $w = w_1 \dots w_n \in \Gamma^*$, la *configuration initiale sur w* est la configuration $(q_0, \underline{\mathrm{B}} w_1 \dots w_n)$. On la note $\mathrm{INIT}_{\mathscr{M}}(w)$.

Un *calcul de $\mathscr{M}$ à partir de w* est un calcul de $\mathscr{M}$ à partir de la configuration $\mathrm{INIT}_{\mathscr{M}}(w)$.

Si le calcul de $\mathscr{M}$ à partir de w est acceptant, on dit que $\mathscr{M}$ *accepte w* et on note $\mathscr{M} \models w$.

Rappel

Fonctions partielles

Soient X, Y deux ensembles. Une *fonction partielle* de X dans Y est une application de type $f : A \to Y$ où A est une partie de X.

L'ensemble A est appelé *domaine de* f et noté $\mathrm{dom}(f)$. L'ensemble $\{f(x) : \ x \in A\}$ est appelé *co-domaine de* f, ou *image de A par* f, et noté $f(A)$.

L'assertion « soit $f : X \to Y$ une fonction partielle » signifie que f est une fonction partielle de domaine inclus dans X et de co-domaine inclus dans Y.

Soient $f, g : X \to Y$ deux fonctions partielles. L'assertions « $\forall x \in X \ : \ f(x) = g(x)$ » signifie

$$\mathrm{dom}(f) = \mathrm{dom}(g) \text{ et } \forall x \in \mathrm{dom}(f) : f(x) = g(x).$$

Fonction caractéristique

Soient X un ensemble et A une partie de X. La *fonction caractéristique* de A est la fonction $\mathscr{X}_A : X \to \{0,1\}$ définie par :

$$\forall x \in X \ : \mathscr{X}_A(x) = \begin{cases} 1 \text{ si } x \in A \\ 0 \text{ sinon} \end{cases}$$

Langage et fonction calculés par une MT

Soit $\mathscr{M} = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$ une machine de Turing.

Le **langage accepté par** $\mathscr{M}$ est l'ensemble :

$$L_{\mathscr{M}} = \{w \in \Sigma^* : \mathscr{M} \models w\}.$$

La **fonction calculée par** $\mathscr{M}$ est la fonction partielle $f_{\mathscr{M}} : \Sigma^* \to \Sigma^*$ définie par :

1. $\mathrm{dom}(f_{\mathscr{M}}) = L_{\mathscr{M}}$;
2. pour $w \in L_{\mathscr{M}}$, $f_{\mathscr{M}}(w)$ est le premier mot de Σ^* inscrit sur le ruban à l'issue du calcul de $\mathscr{M}$ sur w.

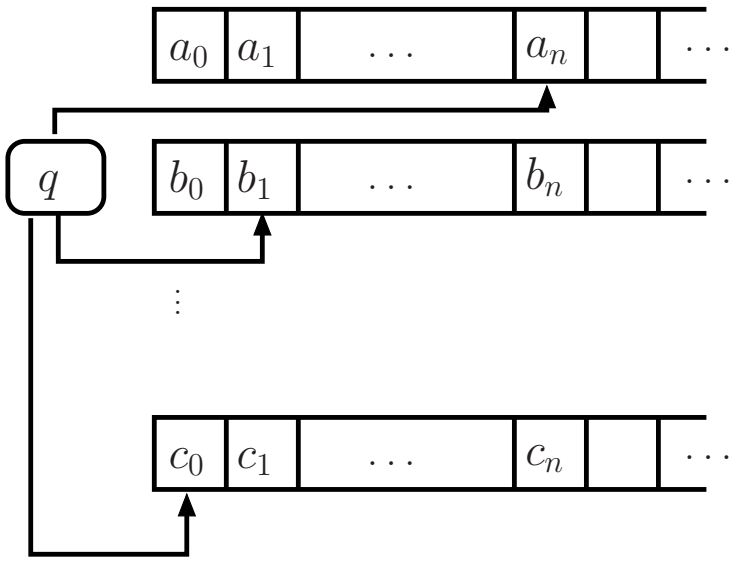
Un langage $L \subseteq \Sigma^*$ est dit **calculable**, ou **semi-décidable**, s'il existe une MT $\mathscr{M}$ telle que $L = L_{\mathscr{M}}$.

Une fonction $f : \Sigma^* \to \Sigma^*$ est dit **calculable**, s'il existe une MT $\mathscr{M}$ telle que $f = f_{\mathscr{M}}$.

Notons que le calcul de $\mathscr{M}$ sur un mot $w \notin L_{\mathscr{M}}$ peut s'arrêter dans l'état q_r, ou diverger.

Un langage $L \subseteq \Sigma^*$ est dit **décidable**, s'il existe une MT $\mathscr{M}$ *qui s'arrête sur toute entrée*, telle que $L = L_{\mathscr{M}}$. Autrement dit, $\mathscr{M}$ s'arrête dans l'état q_a sur tout $w \in L$ et dans l'état q_r sur tout $w \notin L$.

Machines de Turing à k rubans



À chaque pas de calcul, la machine lit les k symboles courants, les remplace par k nouveaux symboles, déplace chaque tête de lecture/écriture d'au plus une case, et enfin, change d'état.

Formellement, une MT à k rubans est un 6-uplet $\mathscr{M} = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$ dont toutes les composantes sont définies comme pour une MT classique, sauf la fonction de transition δ, qui est désormais une application de prototype :

$$\delta : \Gamma^k \times Q \to \Gamma^k \times Q \times \{\leftarrow, \downarrow, \rightarrow\}^k.$$

Les notions de *configuration* et de *calcul* d'un tel modèle sont définies de manière similaire aux notions correspondantes des machines à un ruban.

La *configuration initiale sur le mot* w, notée $\text{INIT}_{\mathscr{M}}(w)$, est celle obtenue en écrivant w au début du premier ruban et des symboles blanc partout ailleurs.

Le *langage accepté par* $\mathscr{M}$ est l'ensemble $L_{\mathscr{M}}$ des mots w tels que le calcul de $\mathscr{M}$ à partir de $\text{INIT}_{\mathscr{M}}(w)$ termine dans l'état q_a.

La *fonction calculée par* $\mathscr{M}$ est la fonction $f_{\mathscr{M}} : \Sigma^* \to \Sigma^*$ de domaine $L_{\mathscr{M}}$ et telle que $\forall w \in L_{\mathscr{M}}$, $f_{\mathscr{M}}(w)$ est le mot de Σ^* écrit le plus à gauche du deuxième ruban à la fin du calcul sur w.

Les notions de *fonction calculée*, *langage reconnu*, *langage décidé* par une MT à k rubans sont définies de manière similaire aux notions correspondantes pour les machines à un ruban.

Théorème 1

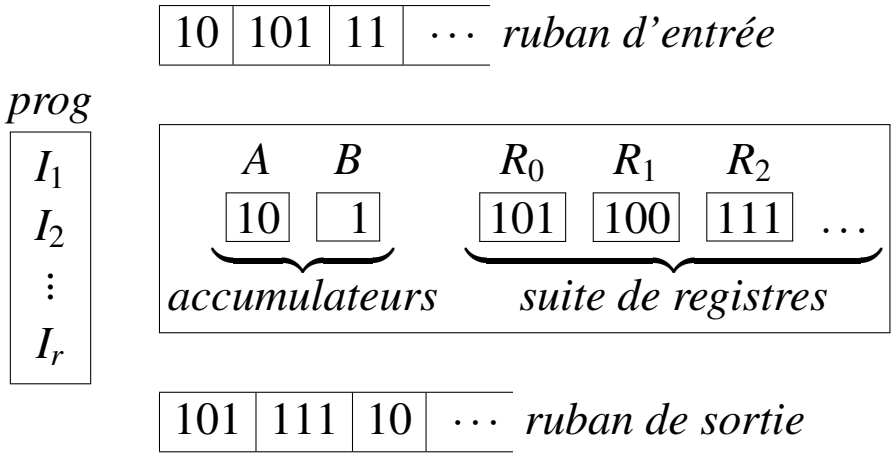
(a) *Toute fonction calculée par une MT à k rubans est calculée par une MT à un ruban.*

(b) *Tout langage reconnu par une MT à k rubans est reconnu par une MT à un ruban.*

(c) *Tout langage décidé par une MT à k rubans est décidé par une MT à un ruban.*

PREUVE. L'assertion (a) sera prouvée en TD. Les assertions (b) et (c) en découlent immédiatement. □

RAM



Le programme est une suite d'instructions étiquetées $I_1, \ldots, I_r$ de l'une des formes :
$A := 0$; $A := 1$; $B := A$; $A := A + B$;
$A := R_A$; $R_A := B$ (R_A = registre dont l'adresse est donnée par A. Ici par exemple : R_2) ;
si $A = B$ alors aller à I_k sinon aller à I_ℓ ; HALTE.

Ce modèle calcule la fonction $f : \{0,1\}^* \to \{0,1\}^*$ telle que : f converge sur w ssi le calcul de la RAM à partir de l'entrée w s'arrête et dans ce cas $f(w)$ est le mot binaire inscrit sur le ruban de sortie à la fin du calcul.

Théorème 2 *RAM-calculable = Turing-calculable.*

THÈSE DE CHURCH

« Calculable » = Turing-calculable

Commentaires

– Cette assertion n'est pas prouvable. Ce n'est pas un théorème, mais le constat expérimental d'une *adéquation entre un modèle et la réalité* que ce modèle prétend représenter.

– Le théorème 2 indique qu'on ne change pas la validité de cette thèse en remplaçant « Turing-calculable » par « RAM-calculable ».

– La thèse de Church a été formulée en 1936. Elle n'a jamais été démentie depuis, malgré de nombreuses tentatives. Dans les années 30-50, les logiciens ont imaginé toutes sortes de processus de calcul (*lambda calcul, système de Post, automates cellulaires,* ...) qui se sont tous avérés équivalents aux machines de Turing.

Deux démonstrations par diagonalisation

$\boxed{\mathbb{R} \not\sim \mathbb{N}}$

Rappelons que deux ensembles X, Y sont *équipotents* s'il existe une bijection de l'un sur l'autre. On note alors $X \sim Y$. Les ensembles *dénombrables* sont alors les ensembles équipotents à $\mathbb{N}$. En d'autres termes, un ensemble est dénombrable s'il existe une énumération $x_0, x_1, \ldots, x_k, \ldots$ de ses éléments.

Théorème 3 $\mathbb{R}$ *n'est pas dénombrable.*

PREUVE. Supposons le contraire. Soit $x_0, x_1, \ldots, x_k, \ldots$ une énumération des éléments de $\mathbb{R}$. Écrivons-la sous la forme suivante :

x_0	$0,\mathbf{1}201398\ldots$
x_1	$54,3\mathbf{7}44588\ldots$
x_2	$3,01\mathbf{3}0772\ldots$
x_3	$172,920\mathbf{5}436\ldots$
$\vdots$	

On considère alors le réel x de partie entière nulle et dont la $i^{\text{ème}}$ décimale est le successeur de la $i^{\text{ème}}$ décimale de x_i. Ici, par exemple : $x = 0,2846\ldots$ Alors x diffère par au moins une décimale de chacun des x_i et n'apparaît donc pas dans l'énumération. D'où une contradiction. □

$$\boxed{X \not\sim \mathscr{P}(X)}$$

Théorème 4 *Un ensemble n'est jamais équipotent à l'ensemble de ses parties.*

PREUVE. Soient X un ensemble, $\mathscr{P}(X)$ l'ensemble de ses parties et b une application de X dans $\mathscr{P}(X)$. Pour chaque $x \in X$, $b(x)$ est un sous-ensemble de X et on a soit $x \in b(x)$, soit $x \notin b(x)$.
Soit alors $X_0 = \{x \in X \ : \ x \notin b(x)\}$. Si b était surjective, il existerait $x_0 \in X$ tel que $X_0 = b(x_0)$. Mais on aurait alors $x_0 \in X_0 \Leftrightarrow x_0 \notin X_0$. Ainsi, b ne peut pas être surjective ni, *a fortiori*, bijective. □

$$\boxed{\mathscr{P}(\Sigma^*) \not\sim \mathbb{N}}$$

Corollaire 5 *Soit Σ un alphabet (fini). L'ensemble $\mathscr{P}(\Sigma^*)$ des langages sur Σ n'est ni fini, ni dénombrable.*

PREUVE. $\mathscr{P}(\Sigma^*)$ est infini puisqu'il contient, par exemple, la suite de langages $(\{a^n\})_{n\in\mathbb{N}}$.
De plus, l'ensemble Σ^* est dénombrable, puisqu'on peut l'énumérer en considérant un ordre total $\leq$ sur Σ puis en listant les éléments de Σ^* dans l'ordre lexicographique induit par $\leq$.
Par le théorème qui précède, $\mathscr{P}(\Sigma^*)$ n'est pas équipotent à Σ^*, donc n'est pas dénombrable. □

Codage des machines de Turing

Lemme 6 *Il existe un codage injectif des machines de Turing sur l'alphabet* $\{0,1\}$.

PREUVE. Notons $\mathscr{M} = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$. On peut supposer, sans perte de généralité, que $\Sigma = \{a_0, \ldots, a_{s-1}\}$, $\Gamma = \Sigma \cup \{a_s\}$ et $Q = \{q_0, \ldots, q_k\}$ $(k, s \in \mathbb{N})$.

Notant $\text{bin}(i)$ la représentation binaire d'un entier i, on peut coder la lettre a_i par $\text{bin}(i)$ et l'état q_i par $0\text{bin}(i)$ ou $\text{bin}(i)$ suivant que q_i est ou n'est pas terminal. On notera $\overline{a}_i$ (resp. $\overline{q}_i$) le code ainsi obtenu.

Une transition $\delta(q_i, a_j) = (q_{ij}, a_{ij}, m_{ij})$ est alors codée par le mot $\overline{q}_i : \overline{a}_j : \overline{q}_{ij} : \overline{a}_{ij} : m_{ij}$. Et la fonction de transition est représentée par le mot

$$\overline{q_0} : \overline{a}_0 : \overline{q}_{00} : \overline{a}_{00} : m_{00} \diamond \ \ldots \ \diamond \overline{q}_k : \overline{a}_s : \overline{q}_{ks} : \overline{a}_{ks} : m_{ks}$$

sur l'alphabet $\sigma = \{0, 1, :, \leftarrow, \downarrow, \rightarrow, \diamond\}$. Comme ce mot permet de plus de retrouver la description complète de $\mathscr{M}$, il constitue bien un codage de $\mathscr{M}$ sur σ.

On se ramène enfin à l'alphabet $\{0,1\}$ en codant σ sur $\{0,1\}^3$ *via*, par exemple, le codage : $0 \rightsquigarrow 000$, $1 \rightsquigarrow 001$, $: \rightsquigarrow 010$, $G \rightsquigarrow 011$, $I \rightsquigarrow 100$, $D \rightsquigarrow 101$, $\diamond \rightsquigarrow 110$. □

Remarque 7 *Ce codage est décidable (i.e., il existe une MT qui sur une entrée* $c \in \{0,1\}^*$*, accepte si c est le codage d'une MT, refuse sinon).*

Existence de problèmes indécidables

Théorème 8 *Il existe des problèmes indécidables.*

PREUVE. L'ensemble des problèmes décidables s'injecte dans l'ensemble des machines de Turing qui à son tour, s'injecte dans $\{0,1\}^*$ (voir Lemme 6). Or $\{0,1\}^*$ s'injecte dans $\mathbb{N}$ *via* l'application qui envoit $w \in \{0,1\}^*$ sur l'entier d'écriture binaire $1w$. Par conséquent, l'ensemble des problèmes décidables est dénombrable.

Or, l'ensemble des langages sur un alphabet fini est non dénombrable (voir Corollaire 5).

Il existe donc une infinité non dénombrable de langages indécidables. □

Machine de Turing universelle

Le Lemme 6 permet de représenter chaque machine de Turing $\mathscr{M}$ par un mot de $\{0,1\}^*$ que l'on notera $\langle \mathscr{M} \rangle$.

Si w est un mot sur $\Sigma_{\mathscr{M}}$, on note $\langle w \rangle$ (resp. $\langle \mathscr{M}, w \rangle$) le codage de w (resp. de $(\mathscr{M}, w)$) sur $\{0,1\}$ obtenu suivant des modalités similaires à celles du codage de $\mathscr{M}$.

Remarque 9 *Ces codages sont eux aussi décidables (voir Remarque 7).*

Le théorème suivant affirme l'existence d'une MT *universelle*, capable de simuler le fonctionnement de toute machine de Turing.

Théorème 10 *Il existe une machine $\mathscr{M}_{\text{UNIV}}$ qui calcule le langage L_{UNIV} et la fonction f_{UNIV} définis comme suit :*
– $L_{\text{UNIV}} = \{\langle \mathscr{M}, w \rangle,\ \mathscr{M} \models w\}$;
– $\forall \langle \mathscr{M}, w \rangle \in L_{\text{UNIV}}$: $f_{\text{UNIV}}(\langle \mathscr{M}, w \rangle) = \langle f_{\mathscr{M}}(w) \rangle$.

PREUVE. On construit $\mathscr{M}_{\text{UNIV}}$ selon les modalités suivantes : sur une entrée $u \in \{0,1\}^*$,

1. $\mathscr{M}_{\text{UNIV}}$ teste si u est un code de la forme $\langle \mathscr{M}, w \rangle$ (voir Remarque 9) ;
2. Dans la négative, $\mathscr{M}_{\text{UNIV}}$ rejette. ;
3. dans l'affirmative, $\mathscr{M}_{\text{UNIV}}$ simule le calcul de $\mathscr{M}$ sur w ;
4. si ce calcul s'arrête dans l'état acceptant de $\mathscr{M}$, alors $\mathscr{M}_{\text{UNIV}}$ recopie le résultat calculé par $\mathscr{M}$ sur w sur son ruban de sortie et s'arrête dans l'état q_a.

□

Le Théorème de l'arrêt

$\text{ACCEPT}_{\text{MT}}$
Entrée : Un couple $(\mathscr{M}, w)$, où $\mathscr{M}$ est une MT et w un mot sur l'alphabet de $\mathscr{M}$;
Question : $\mathscr{M}$ accepte-t-elle w ?

Théorème 11 *Le problème* $\text{ACCEPT}_{\text{MT}}$ *est semi-décidable mais pas décidable.*

PREUVE. Considérons les langages suivants :

$$\begin{aligned}
\text{ACCEPT}_{\text{MT}} &= \{\langle \mathscr{M}, w\rangle \ : \ \mathscr{M} \models w\} \\
\text{ACCEPT}_{\text{MT}}^{\text{diag}} &= \{\langle \mathscr{M}\rangle \ : \ \mathscr{M} \models \langle \mathscr{M}\rangle\} \\
\text{coACCEPT}_{\text{MT}}^{\text{diag}} &= \{\langle \mathscr{M}\rangle \ : \ \mathscr{M} \not\models \langle \mathscr{M}\rangle\}
\end{aligned}$$

$\text{ACCEPT}_{\text{MT}}$ est un codage sur $\{0,1\}$ du problème de l'arrêt. Or $\text{ACCEPT}_{\text{MT}}$ est le langage accepté par la machine $\mathscr{M}_{\text{UNIV}}$. Il est donc bien semi-décidable.

Supposons par ailleurs, en vue d'une contradiction, que $\text{ACCEPT}_{\text{MT}}$ est décidable.

Alors, d'une part, $\text{ACCEPT}_{\text{MT}}^{\text{diag}}$ est décidable : à partir d'une MT $\mathscr{M}_1$ qui décide $\text{ACCEPT}_{\text{MT}}$, on construit une MT $\mathscr{M}_2$ qui, sur l'entrée $\langle \mathscr{M}\rangle$, calcule le mot $\langle \mathscr{M}, \langle \mathscr{M}\rangle\rangle$ puis simule le calcul de $\mathscr{M}_1$ sur $\langle \mathscr{M}, \langle \mathscr{M}\rangle\rangle$. Cette machine décide bien $\text{ACCEPT}_{\text{MT}}^{\text{diag}}$.

D'autre part, la décidabilité de $\text{ACCEPT}_{\text{MT}}^{\text{diag}}$ entraine celle de $\text{coACCEPT}_{\text{MT}}^{\text{diag}}$ puisque l'ensemble des langages décidables est clos par complémentation (facile à voir).

Soit alors $\mathscr{M}_0$ une MT qui décide $\text{coACCEPT}_{\text{MT}}^{\text{diag}}$. On a :

$$\langle \mathscr{M}_0 \rangle \in \text{coACCEPT}_{\text{MT}}^{\text{diag}} \text{ ssi } \langle \mathscr{M}_0 \rangle \notin L(\mathscr{M}_0)$$
(par définition de $\text{coACCEPT}_{\text{MT}}^{diag}$*)*

et

$$\langle \mathscr{M}_0 \rangle \in \text{coACCEPT}_{\text{MT}}^{\text{diag}} \text{ ssi } \langle \mathscr{M}_0 \rangle \in L(\mathscr{M}_0)$$
(puisque $\mathscr{M}_0$ *décide* $\text{coACCEPT}_{\text{MT}}^{diag}$*).*

D'où une contradiction. Il s'ensuit que $\text{ACCEPT}_{\text{MT}}$ n'est pas décidable. □

Chapitre 2 : Complexité en temps

On s'intéresse désormais aux problèmes décidables et à ceux là uniquement. Peut-on définir une notion de *difficulté intrinsèque* de ces problèmes ? Tout problème décidable peut-il être *effectivement* résolu ?

Plan du chapitre

– Temps déterministe et classe P
– réductions polynômiales
– Thèse d'Edmonds
– Temps non-déterministe et classe NP
– NP et les certificats polynômiaux
– Problèmes NP-complets et Théorème de Cook-Levin
– D'autres problèmes NP-complets
– Théorème de Ladner
– La classe coNP

Complexité en temps

Idée informelle : Soit $t : \mathbb{N} \to \mathbb{N}$. Un problème π est dit de *complexité en temps* $t(n)$ s'il existe un **algorithme** qui résoud toute instance de **taille** n du problème en un **temps** au plus $t(n)$.

Taille d'une instance

La formalisation des problèmes vue précédemment (sous forme de langages) induit une notion naturelle de taille des entrées : la taille n d'une instance I de π sera simplement la longueur du mot qui code cette instance.

Nous savons cependant qu'un même problème peut être représenté et codé de plusieurs manières. Ainsi, un graphe $G = (S,A)$ à n sommets et p arêtes peut être représenté par sa matrice d'adjacence (taille : $\ell_1(G) = n^2$) ou par ses listes de successeurs (taille : $\ell_2(G) = n + 2p$).
Un entier N, quant à lui, peut être représenté en binaire (taille : $\ell_1(N) = log_2(N)$) ou en décimal (taille : $\ell_2(N) = \log_{10}(N)$).

L'essentiel pour notre propos, c'est que deux représentations raisonnables d'un même objet soient de tailles *polynomialement liés*. Ici, par exemple : $\ell_1(G) = \Theta(\ell_2^2(G))$ et $\ell_1(N) = \Theta(\ell_2(N))$.

Algorithme et temps de calcul

Par algorithme, nous entendrons, comme dans la première partie de ce cours, “programme de machine de Turing”. La durée du calcul d'une MT $\mathscr{M}$ sur une entrée w est alors simplement le nombre de transitions effectuées au cours du calcul de $\mathscr{M}$ sur w.

Là encore, on pourrait se demander si ces définitions sont à la fois assez générales et assez précises : n'aurions nous pas pu considérer la MT à k rubans comme modèle de référence ? Ou encore, la RAM, en définissant alors la durée d'un calcul comme le nombre d'instructions élémentaires exécutées au cours de ce calcul ?

Ces choix seraient effectivement licites. A nouveau, l'essentiel est que les notions de “temps de calcul” associés à deux modèles de calcul “raisonnables” sont polynomialement liées. On a par exemple :

Proposition 12

1. *Une MT à k rubans fonctionnant en temps $t(n)$ est simulée par par une MT à un seul ruban travaillant en temps $O(t^2(n))$.*
2. *Une MT à un ruban travaillant en temps $t(n)$ est simulée par une RAM en temps $O(t(n))$.*
3. *Une RAM travaillant en temps $t(n)$ est simulée par une MT à plusieurs rubans en temps $O(t^3(n)$.*

Les classes $\mathsf{DTIME}(t(n))$

Dans toute la suite, notre modèle de calcul de référence sera la *machine de Turing multi-rubans*. Nous continuerons à appeler "machine de Turing", ou "MT", ce modèle.

Soit $\mathscr{M}$ une machine de Turing. La *complexité en temps* de $\mathscr{M}$ est la fonction $t : \mathbb{N} \to \mathbb{N}$ qui associe à tout $n \in \mathbb{N}$ le temps de calcul maximal de $\mathscr{M}$ sur une entrée de taille n.

Si la complexité en temps de $\mathscr{M}$ est t, on dit que $\mathscr{M}$ *fonctionne* (ou *calcule*) en temps $t(n)$.

Un langage $L \subseteq \Sigma^*$ est dit *décidable en temps* $t(n)$ s'il est décidé par une machine fonctionnant en temps $t(n)$.

Une fonction $f : \Sigma^* \to \Sigma^*$ est dite *calculable en temps* $t(n)$ si elle est calculée par une machine fonctionnant en temps $t(n)$.

On appelle *classe de complexité en temps* $t(n)$, et on note $\mathsf{DTIME}(t(n))$, la classe des langages décidables en temps $t(n)$.

Si $f : \Sigma^* \to \Sigma^*$ est une fonction calculable en temps $t(n)$, on notera parfois, par abus de langage : $f \in \mathsf{DTIME}(t(n))$.

La classe P

On appelle *classe de complexité en temps déterministe polynômial* la classe :

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{DTIME}(n^k).$$

– Un langage L est donc dans P ssi il existe un entier k et une machine de Turing qui, pour chaque mot $w \in \Sigma^*$ de longueur n répond à la question : « a-t-on $w \in L$? » en au plus n^k étapes.

– Compte tenu des remarques faites précédemment, on peut indifféremment remplacer “machine de Turing” par “machine à trois rubans”, “RAM”, . . . ou n’importe quel modèle de calcul raisonnable.

– Soient π un problème et L_1, L_2 deux codages possibles de π sous forme de langage. Alors si ces deux codages sont raisonnables, on a $L_1 \in \mathsf{P}$ ssi $L_2 \in \mathsf{P}$. Ceci permet de parler des problèmes appartenant à P comme étant les problèmes dont l’un des (et donc tous les) codages est (sont) dans P.

– Pour prouver qu’un problème π est dans P, il suffira d’exhiber un algorithme qui résoud ce problème en temps polynômial, sans ce soucier de la représentation de π choisie ni du modèle de calcul sur lequel on envisage d’implémenter l’algorithme.

Réductions polynômiales

Pour tout problème de décision π, on note $\mathscr{I}(\pi)$ (resp. $\mathscr{I}^+(\pi)$) l'ensemble des instances (resp. des instances positives) de π.

Soient π_1 et π_2 deux problèmes de décision. Une *réduction polynômiale de π_1 à π_2* est une application

$$r : \mathscr{I}(\pi_1) \to \mathscr{I}(\pi_2)$$

telle que :

1. pour toute $I \in \mathscr{I}(\pi_1)$:

   $$I \in \mathscr{I}^+(\pi_1) \text{ ssi } r(I) \in \mathscr{I}^+(\pi_2).$$

   (On dit que *r préserve la positivité des instances*.)

2. r est calculable en temps polynômial.

Lorsqu'il existe une réduction polynômiale de π_1 à π_2, on dit que π_1 *se réduit polynômialement* à π_2 et on note

$$\pi_1 \leq_{\text{pol}} \pi_2.$$

Lemme 13 *Si* $\pi_1 \leq_{pol} \pi_2$ *alors :* $\pi_2 \in \mathsf{P} \Rightarrow \pi_1 \in \mathsf{P}$.

PREUVE. Soit donc r une réduction polynômiale de π_1 à π_2. Appelons

$\mathscr{M}_2$ une MT qui décide π_2 en temps n^k et

$\mathscr{M}_r$ une MT qui calcule r en temps n^ℓ.

On considère alors l'algorithme suivant : *Sur l'entrée* $w \in \mathscr{I}(\pi_1)$ *:*

1. *calculer* $r(w)$ *grâce à* $\mathscr{M}_r$ *;*
2. *décider si* $r(w) \in \mathscr{I}^+(\pi_2)$ *grâce à* $\mathscr{M}_2$.

Cet algorithme décide clairement π_1 puisque

$$r(w) \in \mathscr{I}^+(\pi_2) \text{ ssi } w \in \mathscr{I}^+(\pi_1).$$

De plus il fonctionne en temps polynômial, en vertu des remarques suivantes :

- la phase 1 de l'algorithme se fait en temps $|w|^k$;
- par conséquent, le résultat $r(w)$ de ce calcul est un mot de taille au plus $|w|^k$;
- la phase 2 de l'algorithme nécessite $|r(w)|^\ell$ étapes, c'est-à-dire au plus $(|w|^k)^\ell = |w|^{k\ell}$ étapes.

Ainsi π_1 est-il décidé en temps $n^k + n^{k\ell} \leq n^{k\ell+1}$ et appartient donc bien à P. □

Lemme 14 *La relation $\leq_{pol}$ est transitive.*

PREUVE. Soient

π_1, π_2, π_3 trois problèmes de décision,

r_1 une réduction de π_1 à π_2 calculable en temps n^k et

r_2 une réduction de π_2 à π_3 calculable en temps n^ℓ.

On considère l'application $r : \mathscr{I}(\pi_1) \to \mathscr{I}(\pi_3)$ définie par :

$$\forall w \in \mathscr{I}(\pi_1) \ : \ r(w) = r_2 r_1(w).$$

Alors r est une réduction polynômiale de π_1 à π_2. En effet :

1. r préserve la positivité des instances :
   $w \in \mathscr{I}^+(\pi_1)$ ssi $r_1(w) \in \mathscr{I}^+(\pi_2)$
   puisque r_1 et une réduction
   ssi $r_2(r_1(w)) = r(w) \in \mathscr{I}^+(\pi_3)$
   puisque r_2 et une réduction
2. r est calculée en temps polynômial par l'algorithme :
   (i) calculer $r_1(w)$;
   (Ceci nécessite au plus $|w|^k$ pas de calcul, puisque $r_1 \in \mathsf{DTIME}(n^k)$.)
   (ii) calculer $r_2 r_1(w)$.
   (Ceci nécessite au plus $|w|^{k\ell}$ pas de calcul, puisque $r_2 \in \mathsf{DTIME}(n^\ell)$ et puisque $r_1(w)$, calculé en temps $|w|^k$, est de taille au plus $|w|^k$.)

   Cet algorithme est de complexité $n^k + n^{k\ell} \leq n^{k\ell+1}$ et r est bien polynômiale.

□

Quelques problèmes polynômiaux

ACCESSIBILITÉ

Entrée : Un graphe G et deux sommets s, t de G ;

Question : Existe-t-il un chemin de s à t dans G ?

PREMIERS 2 À 2

Entrée : Deux entiers n et p ;

Question : n et p sont-ils premiers entre eux ?

HORN-SAT

Entrée : Une conjonction Φ de clauses de horn ;

Question : Φ est-elle satisfaisable ?

VERIF-SAT

Entrée : Une formule propositionnelle Φ et une évaluation T ;

Question : T est-elle un modèle de Φ ?

Thèse d'Edmonds

Problèmes « faciles » = problèmes polynomiaux

Commentaires

– Le chercheur J. Edmonds a construit de nombreux algorithmes "efficaces" en théorie des graphes. C'est lui qui a introduit la classe P, en 1965, pour caractériser la notion de problème "faciles", "effectivement décidables".

– Bien sûr, il peut paraître illusoire de considérer comme efficace un algorithme de complexité n^{50}, par exemple. Mais l'expérience montre que les problèmes *naturels* appartenant à P sont le plus souvent de complexité au plus $O(n^5)$.

– D'autre part, pour qui cherche à établir l'existence de problèmes qui n'acceptent pas de traitement algorithmique efficace, il est tout à fait raisonnable de considérer comme tels les problèmes non polynômiaux.

Problèmes de recherche

Tous les problèmes abordés par la suite admettent un traitement algorithmique en temps exponentiel. Plus formellement, ils appartiennent à la classe :

$$\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathsf{DTIME}(2^{n^k}).$$

Parmi eux, un grand nombre sont des *problèmes de recherche*, c'est-à-dire des problèmes de la forme suivante : soit π une propriété "facile à vérifier" :

PROBLÈME DE RECHERCHE
Entrée : une structure E (une formule, un graphe, ...) ;
Question : Existe-t-il un objet O qui vérifie la propriété π vis à vis de E ?

Exemples : HAMILTON, ACCESSIBILITÉ, SAT, HORN-SAT, ...

À chaque fois, l'objet O est de taille polynomialement bornée par celle de E. Vérifier qu'un objet O donné satisfait π prend alors un temps polynômial en $|E|$.

Mais en général, les objets candidats sont en nombre exponentiel en $|E|$ et si l'on ne dispose pas d'un autre algorithme que l'algorithme "force brute" (*i.e.* examen systématique de tous les candidats) pour chercher la solution, on obtient une complexité exponentielle.

Nous allons maintenant introduire la classe NP qui rendra compte de cette structure commune des problèmes algorithmiques. Il nous faut, au préalable, définir une version “non déterministe” de la machine de Turing.

Machines de Turing non déterministes

Une *machine de Turing non déterministe* (MTN en abrégé) est un 7-uplet $\mathscr{M} = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$ où Q, Σ, Γ, Q, q_0, q_a, q_r sont définis comme pour une MT mais où δ est une partie de $(\Gamma \times Q) \times (\Gamma \times Q \times \{\leftarrow, \downarrow, \rightarrow\})$ plutôt qu'une application de $\Gamma \times Q$ vers $\Gamma \times Q \times \{\leftarrow, \downarrow, \rightarrow\}$.

Ainsi, à partir d'un symbole $a \in \Gamma$ lu par sa tête de lecture et d'un état $p \in Q$ de son unité de contrôle, la machine peut effectuer plusieurs transitions, correspondant à tous les triplets (b, q, m) de $\Gamma \times Q \times \{\leftarrow, \downarrow, \rightarrow\}$ tels que $(a, p, b, q, m) \in \delta$.

Des configurations distinctes peuvent de ce fait *dériver en un pas* d'une même configuration de départ. Un *calcul de* $\mathscr{M}$ est toujours une suite de configurations $C_1, C_2, \ldots$ telle que chacune dérive en un pas de la précédente mais désormais il n'y a plus *un* mais *des* calculs possibles de $\mathscr{M}$ sur une entrée w.

L'ensemble des configurations intervenant dans ces calculs, ordonné par la relation de transition en un pas, est appelé *arbre des calculs de* $\mathscr{M}$ *sur* w.

Il se peut que certains des calculs de $\mathscr{M}$ sur w convergent alors que d'autres divergent.

La classe NP

On dit que la MTN $\mathscr{M}$ *accepte* le mot w s'il existe un calcul acceptant de $\mathscr{M}$ sur w. L'ensemble des mots acceptés par $\mathscr{M}$ est appelé *langage accepté par* $\mathscr{M}$ et noté $L_{\mathscr{M}}$.

Soit $t : \mathbb{N} \to \mathbb{N}$. On dit que la MTN $\mathscr{M}$ *calcule en temps* $t(n)$ si pour tout mot $w \in L_{\mathscr{M}}$ de longueur n, il existe un calcul de $\mathscr{M}$ sur w de taille $t(n)$.

On appelle *classe de complexité en temps non déterministe* $t(n)$, et on note

$$\mathsf{NTIME}(t(n)),$$

la classe des langages acceptés en temps $t(n)$ par une machine de Turing non déterministe.

On appelle *classe de complexité en temps non déterministe polynômial* la classe :

$$\mathsf{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k).$$

– Comme la classe P, la classe NP ne dépend pas du modèle de calcul (non déterministe) utilisé pour la définir.
– Si un langage L codant un problème de décision π est dans NP, alors tout autre codage raisonnable de π est dans NP. Ceci permet de parler des problèmes appartenant à NP comme étant les problèmes dont les codages sont dans NP.

Propriétés de NP

Lemme 15 *La classe* NP *contient la classe* P.

PREUVE. Clair, puisqu'une MT polynômiale est un cas particulier de MTN polynômiale. □

Lemme 16

$$\mathsf{NTIME}(f(n)) \subseteq \bigcup_{c} \mathsf{DTIME}(c^{f(n)}).$$

PREUVE. Étant donnée une MTN $\mathscr{N}$ fonctionnant en temps polynômial $q(n)$, on construit une MT $\mathscr{D}$ qui simule les $r^{q(n)}$ calculs de $\mathscr{N}$ de longueur inférieure à $q(n)$ (r est le nombre maximal de choix possibles à chaque étape du calcul de $\mathscr{N}$). Cette simulation se fait en un temps polynômial en $r^{q(n)}$, donc exponentiel en une puissance de n. (Voir détails en TD.) □

Ce résultat montre au passage qu'un langage *accepté* par une machine de Turing non déterministe est *décidé* par une machine de Turing déterministe, dès lors que la complexité en temps de la MTN est bornée.

Corollaire 17 $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}(2^{n^k})$.

Lemme 18 *Si* $\pi_1 \leq_{pol} \pi_2$ *alors :* $\pi_2 \in \mathsf{NP} \Rightarrow \pi_1 \in \mathsf{NP}$.

PREUVE. En "composant" une MT polynomiale qui calcule la réduction et une MTN polynômiale qui reconnait π_2, on obtient une MTN polynômiale qui reconnait π_1. □

Certificat polynômial

Nous énonçons ici un résultat technique qui formalise l'idée selon laquelle la classe NP contient exactement les problèmes relevant de l'appélation "problèmes de recherche".

Soit Σ un alphabet. Une relation $R \subseteq \Sigma^* \times \Sigma^*$ est dite *polynômialement bornée* s'il existe $k \geq 1$ tel que

$$(x,y) \in R \Rightarrow |y| \leq |x|^k.$$

Par ailleurs, R est dite *polynômialement décidable* si le langage $\{x\mathrm{B}y \ : \ (x,y) \in R\}$ est dans P.

Théorème 19 *Un langage $L \subseteq \Sigma^*$ est dans* NP *ssi il existe une relation $R \subseteq \Sigma^* \times \Sigma^*$ polynômialement bornée et polynômialement décidable telle que :*

$$L = \{x \in \Sigma^* \ : \ \exists y \in \Sigma^* \ t.q. \ (x,y) \in R\}.$$

Chaque y ainsi associé à un élément x de L est appelé *certificat polynômial* de x.

Preuves d'appartenance à NP

Dans la pratique, prouver qu'un problème π est dans NP revient désormais à construire une procédure de décision pour π telle que pour chaque instance I de π, la procédure :

1. "Devine" un objet O de taille polynomiale en $|I|$, candidat à "certifier" la positivité de I.
2. Vérifie en temps polynômial en $|O|$ (et donc en $|I|$) que ce candidat est bien un certificat de I.

Ainsi, SAT est dans NP parce qu'il est résolu par la procédure suivante : pour une formule $\Phi \in \mathscr{I}(\text{SAT})$:

1. Deviner une évaluation v sur les variables de Φ ;
2. vérifier que v satisfait bien Φ.

Clairement, $|v|$ est polynomiale en $|\Phi|$ et la vérification de $v \models \Phi$ se fait bien, on l'a vu, en temps polynômial.

De même, HAMILTON $\in$ NP parce qu'il est résolu par la procédure suivante : pour $G \in \mathscr{I}(\text{HAMILTON})$:

1. Deviner un chemin $p = x_1 \ldots x_n$ dans G ;
2. vérifier que p est hamiltonien.

Là encore, $|C|$ est polynomiale en $|G|$ et la vérification de l'hamiltonicité de C se fait en temps polynômial.

Problèmes NP-complets

Nous avons établi les inclusions $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$. Mais on ne sait pas où se situe exactement NP entre ces deux bornes.

La question de l'égalité des classes P et NP, notamment, est encore une question ouverte aujourd'hui. Ses enjeux pratiques et théoriques en font l'une des interrogations majeures des mathématiques contemporaines.

L'une des manières d'appréhender ce problème consiste à isoler les problèmes "les plus difficiles" de la classe NP. La question de l'égalité $\mathsf{P} = \mathsf{NP}$ se ramène alors à prouver ou à infirmer, pour un tel problème, l'existence d'un algorithme polynômial qui le résoud.

Tel est l'objectif de la notion de NP-*complétude* :

Un problème π est dit NP-*complet* si :

1. $\pi \in \mathsf{NP}$;
2. pour tout problème $\pi' \in \mathsf{NP}$: $\pi' \leq_{\text{pol}} \pi$.

Propriétés des problèmes NP-complets

Théorème 20 *S'il existe un problème π appartenant à* P *et* NP*-complet alors* P $=$ NP.

PREUVE. Par hypothèse, $\pi \in \mathsf{P}$ et $\forall \pi' \in \mathsf{NP} : \pi' \leq_{\text{pol}} \pi$. D'où, par le Lemme 13 : $\pi' \in \mathsf{P}$ pour tout $\pi' \in \mathsf{NP}$. □

Théorème 21 *Soient* $\pi, \pi' \in$ NP.

(π est NP*-complet et* $\pi \leq_{pol} \pi'$*)* $\Rightarrow$ *(π' est* NP*-complet).*

PREUVE. Pour tout $\pi'' \in \mathsf{NP}$ on a $\pi'' \leq_{\text{pol}} \pi$ (puisque π est NP-complet). De $\pi \leq_{\text{pol}} \pi'$, on déduit, par transitivité de $\leq_{\text{pol}}$ (Lemme 14) : $\pi'' \leq_{\text{pol}} \pi'$ pour tout $\pi'' \in \mathsf{NP}$. Et puisque $\pi' \in \mathsf{NP}$, ceci entraîne bien la NP-complétude de π'. □

Théorème 22 (Cook-Levin, 1971) SAT *est* NP*-complet.*

PREUVE. Voir [Sip97] p. 254, [Pap94] p. 171 ou [HU79] Section 13.2. □

Commentaires sur la NP-complétude

La conjecture la plus communément admise affirme l'inégalité $\mathsf{P} \neq \mathsf{NP}$.

Ceci parce que d'immenses efforts ont été déployés, en vain, pour trouver des algorithmes efficaces pour chacun des nombreux problèmes NP-complets naturels.

Par conséquent, on estime que prouver la NP-complétude d'un problème revient à établir que ce problème n'a pas de solution efficace.

Lorsqu'un problème π est prouvé NP-complet, on dispose des recours suivants :

1. Se ramener à un sous-problème de π susceptible d'être traité efficacement (cf. HORN-SAT pour SAT).
2. Trouver une heuristique, ou un algorithme "assez efficace dans la pratique", même si dans le pire des cas cet algorithme prend un temps exponentiel (cf. procédure de Davis et Putnam pour CLAUSE-SAT).
3. Chercher un algorithme efficace résolvant π de façon approchée (cf. la recherche de trajet "aussi cours que possible" dans le problème du voyageur de commerce).

Ces trois démarches renvoient, respectivement, à l'algorithmique avancée, l'IA et l'optimisation.

Nouveaux problèmes NP-complets

On considère les problèmes suivants :

CLAUSE-SAT
Entrée : une FNC Φ ;
Question : Φ est-elle satisfaisable ?

3-SAT
Entrée : une FNC Φ dont les clauses s'écrivent avec 3 littéraux exactement ;
Question : Φ est-elle satisfaisable ?

CLIQUE
Entrée : un graphe G et un entier $k > 0$;
Question : G contient-t-il une clique de taille k ?

ANTI-CLIQUE
Entrée : un graphe G et un entier $k > 0$;
Question : G contient-t-il une anticlique de taille k ?

VERTEX-COVER
Entrée : un graphe G et un entier $k > 0$;
Question : Existe-t-il une couverture des sommets de G de taille k ?

Théorème 23 *Ces 5 problèmes sont* NP*-complets.*

PREUVE.

Appartenance à NP

- Pour CLAUSE-SAT et 3-SAT : clair puisque ce sont des sous-problème de SAT.
- Pour CLIQUE : ce problème est résolu par l'algorithme non déterministe suivant :
  1. "Deviner" un ensemble $\mathscr{X}$ de sommets de G ;
  2. vérifier que $\mathscr{X}$ est une clique de taille k.

  Cette procédure est bien (non déterministe) polynômiale puisque $|\mathscr{X}|$ est polynômialement borné en $|G|$ et puisque la phase 2 de l'algorithme se fait facilement en temps déterministe polynômial.
- Pour ANTI-CLIQUE (resp. VERTEX-COVER) : même argument en remplaçant partout "clique" par "anticlique" (resp. par "couverture des sommets").

Complétude

- Pour CLAUSE-SAT : par modification de la preuve du Théorème de Cook-Levin ;
- Pour les autres : par la suite de réductions :

  CLAUSE-SAT $\leq_{\text{pol}}$ 3-SAT $\leq_{\text{pol}}$ VERTEX-COVER $\leq_{\text{pol}}$ CLIQUE $\leq_{\text{pol}}$ ANTI-CLIQUE.

  (Voir détails en TD.) □

Théorème de Ladner

Contrairement à ce que pourrait laisser supposer l'éventail des problèmes NP évoqués jusqu'ici, la classe NP n'est pas partitionnée entre P et l'ensemble NPC des problèmes NP-complets. Plus précisemment :

Théorème 24 *Si* $\mathsf{P} \neq \mathsf{NP}$*, alors il existe des problèmes* NP *non* NP*-complets et n'appartenant pas à* P*. Autrement dit :*

$$\mathsf{P} \neq \mathsf{NP} \Rightarrow (\mathsf{NP} \setminus \mathsf{P}) \setminus \mathsf{NPC} \neq \varnothing.$$

PREUVE. [Pap94] Thm 14.1 p. 330 -[Gol99], Lecture 3, pp. 1-3. □

Jusqu'à présent, on n'a exhibé aucun problème naturel dans cette classe, mais le problème d'*isomorphisme de graphes*, qui est clairement NP, semble être un bon candidat.

GRAPH-ISOM
Entrée : deux graphes G et G' ;
Question : G et G' sont-ils isomorphes ?

Conjecture 25 *Si* $\mathsf{P} \neq \mathsf{NP}$*, alors* GRAPH-ISOM *n'est ni* NP*-dur, ni polynômial.*

La classe coNP

Pour toute classe de complexité $\mathscr{C}$, on appelle *classe complémentaire* de $\mathscr{C}$ l'ensemble :

$$\mathsf{co}\mathscr{C} = \{L : \overline{L} \in \mathscr{C}\}.$$

En particulier, la classe

$$\mathsf{coNP} = \{L : \overline{L} \in \mathsf{NP}\}$$

regroupe les problèmes qui admettent un *certificat de réfutation* polynômial.

Exemples

TAUTOLOGIE $\in$ coNP puisque l'assertion

$$\Phi \in \text{TAUTOLOGIE}$$

est réfutée par l'algorithme suivant :

1. Deviner une évaluation T ;
2. Vérifier que T ne satisfait pas Φ.

De même, PREMIER $\in$ coNP puisque l'assertion

$$n \in \text{PREMIER}$$

est réfutée par l'algorithme suivant :

1. Deviner un entier $d < n$;
2. Vérifier que d divise n.

Clôture par complémentarité

L'égalité $\mathscr{C} = \mathsf{co}\mathscr{C}$ est assurée pour toute classe de complexité $\mathscr{C}$ définie en bornant les ressources d'une MT *déterministe* (*e.g.* P, EXP, ou $\mathsf{DTIME}(t(n))$, pour toute $t : \mathbb{N} \to \mathbb{N}$).
Ceci parce que pour tout langage $L \in \mathscr{C}$ décidé par une MT déterministe $\mathscr{M}$, son complémentaire $\overline{L}$ est décidé par la MT obtenue à partir de $\mathscr{M}$ en inversant les réponse oui/non, *avec les même bornes sur les ressources*.

Par contre, pour les classes de complexité $\mathscr{C}$ définie à partir d'une MT *non-déterministe*, la question $\mathscr{C} = \mathsf{co}\mathscr{C}$? est en générale ouverte.

Proposition 26 *Soit L un langage. Si L est* NP*-complet, alors* $\overline{L}$ *est* coNP*-complet.*

Proposition 27 *S'il existe un langage* coNP*-complet dans* NP*, alors* $\mathsf{NP} = \mathsf{coNP}$.

Remarque 28 $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$.