CH.10 $\mathcal{P} = \mathcal{NP}$?

- 10.1 La déterminisation des machines
- 10.2 Les langages $\mathcal{P}$ et $\mathcal{NP}$
- 10.3 Les problèmes $\mathcal{NP}$-complets

10.1 La déterministation des machines

La puissance de calcul des machines de Turing déterministes est la même que celle des machines non déterministes.
On peut le démontrer en déterminisant une machine non déterministe.
Si un mot $x_1x_2...x_n$ est reconnu par une machine de Turing M non déterministe, il y a des choix faits pour chaque mouvement de M.
La suite de ces choix peut être codée par un mot. Ce mot est écrit sur bande (ou sur une deuxième bande) et cette suite de choix permet de rendre le fonctionnement déterministe. La machine déterminisée fonctionne comme la machine M mais, quand elle a un choix à faire, elle consulte la mot codant les choix.

Dans le cas des automates finis, le nombre d'états de l'automate déterminisé croît de façon exponentielle. Le même phénomène se produit ici.

Les seuls algorithmes utilisables sont ceux de complexité polynomiale.
On s'intéresse donc aux machines de Turing qui peuvent résoudre un problème en un temps polynomial de la taille du problème. Ce problème est codé par une suite de symboles. Ce codage n'est en en général pas unique. Mais pour les problèmes usuels, on peut passer d'un codage à un autre au moyen d'un algorithme de complexité polynomiale. D'un point de vue complexité, tous ces codages sont alors équivalents. La taille du problème est la taille d'un des codages. On peut alors parler de complexité polynomiale d'un problème. Ce caractère polynomial ne dépend pas en effet du codage choisi.

Pour un certain nombre de problèmes (chemins hamiltoniens, coloriage des graphes, problèmes d'emplois du temps, factorisation des nombres entiers), une solution peut être difficile à trouver, mais, si on en propose une, il est facile (= polynomial) de vérifier si elle convient.

La plupart des problèmes utiles sont soit résolubles au moyen d'un algorithme polynomial, soit on peut vérifier si une solution proposée en est bien une en un temps polynomial.
Un examen attentif de ce deuxième type de problèmes a donné lieu, en 1971, à un travail de Cook, à Toronto, intitulé "The Complexity of Theorem Proving Procedures" où est introduite la notion de $\mathcal{NP}$-complétude. De façon inattendue, cette propriété abstraite d'un problème très artificiel se trouve être satisfaite par plusieurs centaines ou milliers de problèmes concrets.
La question de savoir si $\mathcal{P} = \mathcal{NP}$ est le problème non résolu le plus important de l'informatique théorique, figurant parmi les 7 problèmes mathématiques non résolus jugés les plus importants par l'institut Clay et dotés chacun d'un prix d'un million de dollars.

10.2 Les langages $\mathcal{P}$ et $\mathcal{NP}$

Plus précisément, un langage L est **polynomial**, noté $L \in \mathcal{P}$, lorsqu'il existe une machine **déterministe** M qui détermine si un mot x est dans L en un nombre de mouvements qui est une fonction polynomiale P(|x|) de la longueur de x.

Un langage est **non déterministe polynomial**, noté $L \in \mathcal{NP}$, lorsqu'il existe une machine **non déterministe** M s'il existe une suite de choix qui déterminent si un mot x est dans L en un nombre de mouvements qui est une fonction polynomiale P(|x|) de la longueur de x.

Si on a $L \in \mathcal{NP}$, la suite de choix possibles est une fonction polynomiale Q(|x|) de la longueur du mot x qu'on cherche à tester. On fabrique donc pour x toutes les suites de choix possibles, de longueur au plus Q(|x|). On teste les paires formées de x et d'un choix, de façon déterministe, en un temps P(|x| + Q(|x|)), donc polynomial.

On a donc démontré le théorème suivant.

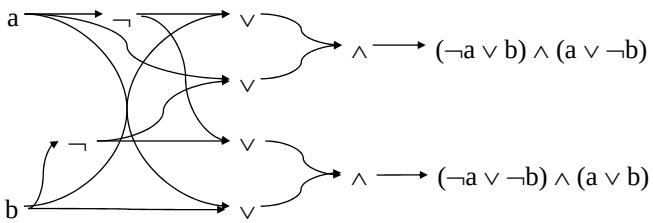
Théorème : Un langage $L \in A^*$ est dans $\mathcal{NP}$ si et seulement si il est la projection d'un langage $L' \in A^* \times C^*$ qui est dans $\mathcal{P}$.

Exemple historique de problème $\mathcal{NP}$: **circuits booléens**.

On fabrique m fonctions booléennes de n variable booléennes.
Opérations permises, non $\neg$, et $\wedge$, ou $\vee$.

Elles peuvent être représentées par un graphe sans cycle avec n entrées et m sorties (une fonction pouvant être représentée par un arbre abstrait).
La taille est le nombre de sommets du graphe.

Par exemple, les 2 fonctions de variables
$((\neg a \vee b) \wedge (a \vee \neg b), (\neg a \vee \neg b) \wedge (a \vee b))$ sont représentées par le graphe suivant :



Problème : existe-t-il une suite de valeurs pour les n variables telle que chacune des m fonctions prenne la valeur 1 ?

On peut tester les 2^n valeurs possibles (complexité exponentielle). Lorsqu'une solution est proposée, on peut la tester en un temps polynomial.

Ce problème peut être codé par une suite des valeurs de toutes les étapes intermédiares. La machine de Turing vérifie que le calcul se déroule bien.

Le problème se ramène donc à savoir si le code correspondant à un tel ensemble de **circuits booléens** est **satisfaisable**.

Si une solution est proposée, il est facile (polynomial) de vérifier si elle convient.

L'ensemble des codes des circuits booléens satisfaisables est donc un langage $\mathcal{NP}$.

10.3 Les problèmes $\mathcal{NP}$-complets

Le résultat surprenant est le suivant. Si un langage L est $\mathcal{NP}$, on a donc une machine non déterministe acceptant les mots x de L.

En considérant la machine de Turing déterministe qui accepte en temps polynomial le langage dont L est la projection (*cf*. plus haut), on peut fabriquer, en un temps polynomial, une suite de fonctions booléennes associées. Si ces fonctions booléennes peuvent être satisfaites simultanément, alors x est dans L.

On dit que le problème initial a été **réduit** au problème de la satisfaisabilité d'un ensemble de circuits booléens : tout algorithme efficace pour celle-ci permet d'obtenir un algorithme efficace pour le problème initial.

On dit que le problème de la satisfaisabilité d'un ensemble de circuits booléens est $\mathcal{NP}$-complet : c'est un problème $\mathcal{NP}$ auquel tous les problèmes $\mathcal{NP}$ peuvent être polynomialement réduits.

C'est le résultat démontré par Cook en 1971.

Une des conséquences est que, si l'on connaît un algorithme polynomial pour le problème de la satisfaisabilité d'un ensemble de circuits booléens, on en déduit que tout problème $\mathcal{P}$ est $\mathcal{NP}$:

Dans ce cas, $\mathcal{P} = \mathcal{NP}$. Et les problèmes utiles, qui sont $\mathcal{NP}$, peuvent être résolus par un algorithme polynomial. Par exemple, l'existence d'un circuit hamiltonien, le nombre chromatique d'un graphe, le problème des emplois du temps, la factorisation d'un nombre entier.

Mais le plus surprenant est que beaucoup de ces problèmes sont en fait équivalents au problème de la satisfaisabilité et donc sont $\mathcal{NP}$-complets.

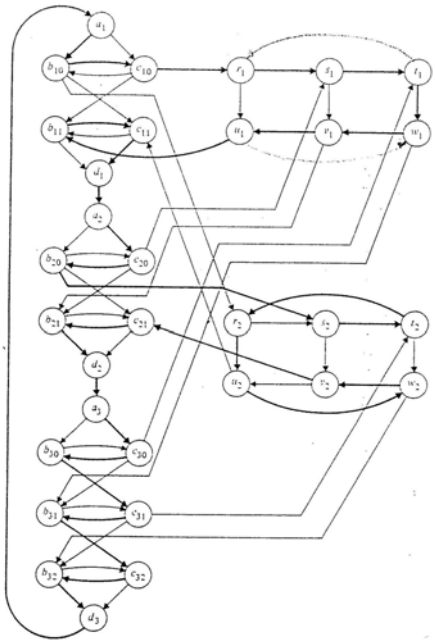
On montre ainsi que le problème de la satisfaisabilité d'un ensemble de circuits booléens dont chacun a trois entrées (problème 3FNC) est équivalent au problème initial de satisfaisabilité (assez facile).

Par exemple, le problème de l'existence d'un chemin hamiltonien est $\mathcal{NP}$-complet.

Il est clairement $\mathcal{NP}$. Si on propose une suite d'arcs, il est facile de vérifier qu'ils constituent un chemin passant une et une seule fois par tous les sommets du graphe.

On peut aussi montrer que le problème 3FNC se réduit à la recherche d'un circuit hamiltonien.

Un exemple est présenté sur la page suivante ;
les clauses sont $(a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c)$, le chemin hamiltonien en gras correspond à a = 1, b = c = 0.



Extrait de "Introduction to Automata Theory, Languages and Computation" de John E. Hopcroft et Jeffrey D. Ullman

Plusieurs centaines, peut-être plusieurs milliers de problèmes sont connus pour être $\mathcal{NP}$-complets. La découverte d'un algorithme polynomial pour l'un quelconque d'entre eux impliquerait que tous peuvent être résolus par un algorithme polynomial.

Parmi ces problemes, outre ceux déjà mentionnés, se trouvent
-le problème du voyageur de commerce ;
-le problème du recouvrement d'un graphe par des sommets (trouver un ensemble de k sommets tels que chaque arête a une extrémité dans cet ensemble) ;
-le problème du nombre chromatique (peut-on colorier les sommets d'un graphe avec k couleurs de façon que deux sommets adjacents soient de couleurs distinctes) ;
-le problème du sac à dos (on se donne un ensemble S d'entiers positifs ; peut-on écrire l'entier t comme somme d'éléments de S) ;

-le problème de l'emploi du temps ;
-le problème de la programmation linéaire en nombres entiers (optimiser une fonction coût linéaire dont les variables sont liées par des contraintes linéaires).

Par contre, le problème de la factorisation des nombres entiers est $\mathcal{NP}$, mais sans doute pas $\mathcal{NP}$-complet.

Problème ouvert, $\mathcal{P}$ est-il égal à $\mathcal{NP}$?
Récompense d'un million de dollars sans limite de temps pour une réponse, quelle qu'elle soit.
Pour une réponse oui, il suffit de trouver un algorithme polynomial pour n'importe lequel des problèmes $\mathcal{NP}$-complets. Pour une réponse non, la plus probable, on n'a pas encore l'outil permettant d'attaquer le problème.