Complexité, logique et matroïdes

Yann Strozecki

Séminaire internationale des thésards de logique et apparentés

27 octobre 2008

Plan de l'exposé

1931 Kurt Gödel démontre que certains énoncés sont indémontrables

1931 Kurt Gödel démontre que certains énoncés sont indémontrables

1936 Alonzo Church introduit le lambda calcul

1931 Kurt Gödel démontre que certains énoncés sont indémontrables

1936 Alonzo Church introduit le lambda calcul

1936 Alan Turing introduit la machine du même nom

1931 Kurt Gödel démontre que certains énoncés sont indémontrables

1936 Alonzo Church introduit le lambda calcul

1936 Alan Turing introduit la machine du même nom

Question : qu'est ce que le calcul, une fonction effective, un algorithme ?

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection
   - les fonctions constantes

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection
   - les fonctions constantes
2. **des schémas**

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection
   - les fonctions constantes
2. **des schémas**
   - le schéma de composition

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection
   - les fonctions constantes
2. **des schémas**
   - le schéma de composition
   - le schéma de récurrence

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection
   - les fonctions constantes
2. **des schémas**
   - le schéma de composition
   - le schéma de récurrence
3. **un autre schéma**

Les fonctions de Church-Russer sont définies inductivement de $\mathbb{N}^k \longrightarrow \mathbb{N}$ par :

1. **des fonctions de base**
   - le successeur
   - la projection
   - les fonctions constantes
2. **des schémas**
   - le schéma de composition
   - le schéma de récurrence
3. **un autre schéma**
   - le schéma μ qui introduit des fonctions partielles

Une machine de turing est définie par :

- un alphabet Σ disons ici $\{0, 1, B\}$

Une machine de turing est définie par :

- un alphabet Σ disons ici $\{0, 1, B\}$
- une bande semi-infinie dont chaque case peut prendre une valeur de l'alphabet

Une machine de turing est définie par :

- un alphabet Σ disons ici $\{0, 1, B\}$
- une bande semi-infinie dont chaque case peut prendre une valeur de l'alphabet
- une tête de lecture qui pointe sur une case

Une machine de turing est définie par :

- un alphabet Σ disons ici $\{0, 1, B\}$
- une bande semi-infinie dont chaque case peut prendre une valeur de l'alphabet
- une tête de lecture qui pointe sur une case
- un ensemble d'état

Une machine de turing est définie par :

- un alphabet Σ disons ici $\{0, 1, B\}$
- une bande semi-infinie dont chaque case peut prendre une valeur de l'alphabet
- une tête de lecture qui pointe sur une case
- un ensemble d'état
- une fonction de transition qui définit la dynamique de cette machine

Une machine de turing est définie par :

- un alphabet Σ disons ici $\{0, 1, B\}$
- une bande semi-infinie dont chaque case peut prendre une valeur de l'alphabet
- une tête de lecture qui pointe sur une case
- un ensemble d'état
- une fonction de transition qui définit la dynamique de cette machine

On peut remplacer la fonction de transition par une relation, c'est à dire qu'à chaque étape il y a plusieurs choix de transition possible. On a alors une machine non déterministe.

On distingue un état qu'on appelle état d'arrêt.

On distingue un état qu'on appelle état d'arrêt.

On définit la fonction partielle $\Sigma^* \to \Sigma^* \cup \{undef\}$ calculée par la machine de Turing T par :
$T(u) =$ l'état de la bande vu comme un mot de Σ^* quand la machine arrive à l'état d'arrêt après avoir démarré sur u.

On distingue un état qu'on appelle état d'arrêt.

On définit la fonction partielle $\Sigma^* \to \Sigma^* \cup \{undef\}$ calculée par la machine de Turing T par :
$T(u) =$ l'état de la bande vu comme un mot de Σ^* quand la machine arrive à l'état d'arrêt après avoir démarré sur u.

Si la fonction est toujours définie on dit qu'elle est totale.

On peut facilement décrire une bijection entre $\mathbb{N}$ et les machines de Turing qu'on notera $i \longrightarrow \phi_i$.

On peut facilement décrire une bijection entre $\mathbb{N}$ et les machines de Turing qu'on notera $i \longrightarrow \phi_i$.

Théorème

Il existe une machine de Turing T telle que $T(\langle i, u \rangle) = \phi_i(u)$.

On suppose que la machine de Turing a deux états d'arrêt distingués q_{oui} et q_{non}.
On appelle un sous-ensemble L de Σ^* un langage ou un problème.
On dit que la machine T décide le langage L si T s'arrête sur q_{oui} quand $u \in L$ et s'arrête sur q_{non} quand $u \notin L$.

On suppose que la machine de Turing a deux états d'arrêt distingués q_{oui} et q_{non}.
On appelle un sous-ensemble L de Σ^* un langage ou un problème.
On dit que la machine T décide le langage L si T s'arrête sur q_{oui} quand $u \in L$ et s'arrête sur q_{non} quand $u \notin L$.

Exemple

On représente les invités potentiels d'une soirée par les sommets d'un graphe et on met une arête entre deux personnes qui ne se supportent pas. Trouver une assistance de taille k ou tout le monde s'apprécie revient à trouver un sous-ensemble indépendant du graphe de taille k.
Le langage (problème) correspondant est noté $\mathit{INDSET} \subseteq \Sigma^*$.

Plan de l'exposé

On définit une mesure du temps de calcul d'une machine de Turing.
$t(u, M) =$ nombre d'étape avant que la machine M s'arrête en partant de u.

On définit une mesure du temps de calcul d'une machine de Turing.
$t(u, M) =$ nombre d'étape avant que la machine M s'arrête en partant de u.

On dit que la machine M calcule en temps $t(n) = max_{|x|=n} t(x, M)$.

Definition

$TIME(f(n)) = \{L \mid \exists M$ qui décide L en temps $\bigcirc(f(n))\}$

On définit une mesure du temps de calcul d'une machine de Turing. $t(u, M) =$ nombre d'étape avant que la machine M s'arrête en partant de u.

On dit que la machine M calcule en temps $t(n) = max_{|x|=n} t(x, M)$.

Definition

$TIME(f(n)) = \{L \mid \exists M$ qui décide L en temps $\bigcirc(f(n))\}$

Si la machine est non déterministe on note l'ensemble $NTIME(f(n))$.

On définit une mesure du temps de calcul d'une machine de Turing. $t(u, M) =$ nombre d'étape avant que la machine M s'arrête en partant de u.

On dit que la machine M calcule en temps $t(n) = max_{|x|=n} t(x, M)$.

Definition

$TIME(f(n)) = \{L \mid \exists M$ qui décide L en temps $\bigcirc(f(n))\}$

Si la machine est non déterministe on note l'ensemble $NTIME(f(n))$.

Cette définition dépends du modèle choisi, ici les machines de Turing. En pratique tout les modèles habituels varient d'au plus d'un facteur polynomial.

Definition

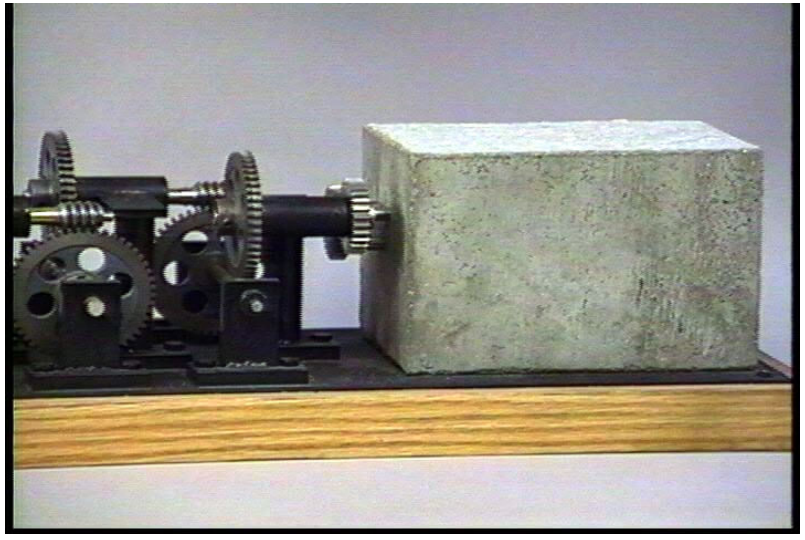
La classe P est l'ensemble $\bigcup_k TIME(n^k)$.

Definition

La classe EXP est l'ensemble $\bigcup_k TIME(2^{n^k})$.

On dit généralement qu'un problème dans P est facile ou "tractable" alors qu'un problème dans EXP est difficile.

The MIT museum contains a kinetic sculpture by Arthur Ganson called *Machine with concrete*. It consists of 13 gears connected to one another in a series such that each gear moves 50 times slower than the previous one. The fastest gear is constantly rotated by an engine at a rate of 212 rotations per minute. The slowest gear is fixed to a block of concrete and so apparently cannot move at all.



Definition (NP machine)

La classe NP est l'ensemble $\bigcup_k NTIME(n^k)$.

Definition (NP machine)

La classe NP est l'ensemble $\bigcup_k NTIME(n^k)$.

Definition (NP logique)

NP est l'ensemble des langages L tel qu'il existe un prédicat $P(x, y)$ décidable en temps polynomial et un polynome Q, tel que $x \in L \Leftrightarrow \exists y \in \Sigma^{Q(|x|)}, P(x, y)$.

Definition (NP machine)

La classe NP est l'ensemble $\bigcup_k NTIME(n^k)$.

Definition (NP logique)

NP est l'ensemble des langages L tel qu'il existe un prédicat $P(x, y)$ décidable en temps polynomial et un polynome Q, tel que $x \in L \Leftrightarrow \exists y \in \Sigma^{Q(|x|)}$, $P(x, y)$.

Lemme

Les deux définitions sont équivalentes.

- Le problème *INDSET* est dans NP.

- Le problème *INDSET* est dans NP.
- Le problème *PRIME*, c'est à dire le problème de tester si un nombre est premier est dans P, grâce à un théorème récent de Manindra Agrawal, Neeraj Kayal et Nitin Saxena.

- Le problème *INDSET* est dans NP.
- Le problème *PRIME*, c'est à dire le problème de tester si un nombre est premier est dans P, grâce à un théorème récent de Manindra Agrawal, Neeraj Kayal et Nitin Saxena.
- Le problème $\mathit{EULERIAN} - \mathit{PATH}$ est dans P.

Definition

On dit que A est réductible en temps polynomial à B si il existe f calculable en temps polynomial telle que $x \in A \Leftrightarrow f(x) \in B$. On écrit $A \leq_p B$ car A est moins général que B.

C'est la notion fondamentale inventé par Karp en vers 1970, elle permet de faire des transferts de résultat.

Definition

On dit que A est réductible en temps polynomial à B si il existe f calculable en temps polynomial telle que $x \in A \Leftrightarrow f(x) \in B$. On écrit $A \leq_p B$ car A est moins général que B.

C'est la notion fondamentale inventé par Karp en vers 1970, elle permet de faire des transferts de résultat.

Definition

On dit qu'un problème B est complet pour la classe $\mathcal{C}$ si $B \in \mathcal{C}$ et $\forall A \in \mathcal{C}$, $A \leq_p B$.

Le problème SAT est de décider si pour une formule du premier ordre $\phi(\vec{x})$ il existe une valuation de $\vec{x}$ la satisfaisant.

Le problème *SAT* est de décider si pour une formule du premier ordre $\phi(\vec{x})$ il existe une valuation de $\vec{x}$ la satisfaisant.

Théorème (COOK)

Le problème SAT est NP*-complet.*

And now for something completely different !

Vous voulez gagner 1 million de dollars ?

Alors résolvez la question :

$$\mathrm{P} = \mathrm{NP}?$$

$\mathrm{P} = \mathrm{NP}$ Le monde ou les bisounours mangent les mathématiciens. On peut démontrer les théorèmes de manière automatique. On peut trouver une solution à la plupart des problèmes d'optimisation ou de design. Même les physiciens sont contents car ils peuvent facilement trouver la théorie minimale qui explique des données expérimentales (rasoir d'Occam). Il n'existe pas de cryptographie sûre, pas de transactions par internet.

$\mathrm{P} \neq \mathrm{NP}$ C'est le monde réel, quand un problème est NP complet, on change de problème. Si on veut quand même faire quelque chose, on contourne l'obstacle : solution approchée, restriction du problème, temps moyen, heuristique, algorithme probabiliste, algorithme quantique .....

Peut être que $\mathrm{P} = \mathrm{NP}$ n'est pas démontrable dans *ZFC* ...

Il y a des problèmes de séparation qu'on sait résoudre :

Il y a des problèmes de séparation qu'on sait résoudre :

Théorème

Si f et g sont des fonctions satisfaisant $f(n)\log f(n) = o(g(n))$, alors $TIME(f(n)) \subsetneq TIME(g(n))$.

Il y a des problèmes de séparation qu'on sait résoudre :

Théorème

Si f et g sont des fonctions satisfaisant $f(n)\log f(n) = o(g(n))$, alors $TIME(f(n)) \subsetneq TIME(g(n))$.

Théorème

$TISP(n^{1,2}, n^{0,2}) \subsetneq NTIME(n)$

Une machine à oracle B peut n'importe quand pendant son calcul tester l'appartenance d'un élément x au langage B en temps 1.

Une machine à oracle B peut n'importe quand pendant son calcul tester l'appartenance d'un élément x au langage B en temps 1.

Théorème (Baker, Gill, Solovay)

Il y a des oracles A, B tels que $\mathrm{P}^A = \mathrm{NP}^A$ et $\mathrm{P}^B \neq \mathrm{NP}^B$.

Une machine à oracle B peut n'importe quand pendant son calcul tester l'appartenance d'un élément x au langage B en temps 1.

Théorème (Baker, Gill, Solovay)

Il y a des oracles A, B tels que $\mathrm{P}^A = \mathrm{NP}^A$ et $\mathrm{P}^B \neq \mathrm{NP}^B$.

Ce théorème utilise la technique de diagonalisation. Il a été montré que ces techniques ne peuvent pas suffire à montrer $\mathrm{P} \neq \mathrm{NP}$

- $U_B = \{1^n | \text{ un mot de longueur } n \text{ est dans } B\}$, cet ensemble est clairement dans NP^B

- $U_B = \{1^n | \text{ un mot de longueur } n \text{ est dans } B\}$, cet ensemble est clairement dans NP^B
- nous allons définir B de façon à ce que $U_B \notin \mathrm{P}^B$, ce qui implique $\mathrm{P}^B \neq \mathrm{NP}^B$.

- $U_B = \{1^n | \text{ un mot de longueur } n \text{ est dans } B\}$, cet ensemble est clairement dans NP^B
- nous allons définir B de façon à ce que $U_B \notin \mathrm{P}^B$, ce qui implique $\mathrm{P}^B \neq \mathrm{NP}^B$.
- on construit B et son complémentaire F par induction sur i en ajoutant un ensemble fini de mots à chaque étape

- $U_B = \{1^n | \text{ un mot de longueur } n \text{ est dans } B\}$, cet ensemble est clairement dans NP^B
- nous allons définir B de façon à ce que $U_B \notin \mathrm{P}^B$, ce qui implique $\mathrm{P}^B \neq \mathrm{NP}^B$.
- on construit B et son complémentaire F par induction sur i en ajoutant un ensemble fini de mots à chaque étape
- supposons B_i et F_i construit, soit $n \geq max\{|x||x \in B_i \cap F_i\}$, exécutons M_i^B sur le mot 1^n pendant $\frac{2^n}{10}$. Si la machine fait une requête sur un mot de statut inconnu répondre qu'il n'est pas dans B et l'ajouter à F_i

- si M_i accepte, ajouter tous les mots de taille n à F_i, ce qui implique $1^n \notin U_B$

- si M_i accepte, ajouter tous les mots de taille n à F_i, ce qui implique $1^n \notin U_B$
- si M_i rejette, ajouter un mot de taille n qui n'a pas été demandé à B_i, ce qui implique $1^n \in U_B$

- si M_i accepte, ajouter tous les mots de taille n à F_i, ce qui implique $1^n \notin U_B$
- si M_i rejette, ajouter un mot de taille n qui n'a pas été demandé à B_i, ce qui implique $1^n \in U_B$
- si M_i ne s'arrête pas alors on ne touche pas à B_i et F_i

- si M_i accepte, ajouter tous les mots de taille n à F_i, ce qui implique $1^n \notin U_B$
- si M_i rejette, ajouter un mot de taille n qui n'a pas été demandé à B_i, ce qui implique $1^n \in U_B$
- si M_i ne s'arrête pas alors on ne touche pas à B_i et F_i
- Conclusion ?

Par une technique similaire on a le théorème suivant.

Théorème (Ladner)

Supposons que $\mathrm{P} \neq \mathrm{NP}$. *Alors il existe un langage* $L \in \mathrm{NP} \setminus \mathrm{P}$ *qui n'est pas* NP*-complet.*

Fixons une signature σ, $\mathcal{L}$ une sous-classe des formules du premier ordre et $\mathcal{S}$ une sous-classe des σ-structures.

Fixons une signature σ, $\mathcal{L}$ une sous-classe des formules du premier ordre et $\mathcal{S}$ une sous-classe des σ-structures.

Le problème de model-checking est de décider si pour une σ structure $M \in \mathcal{S}$ et une formule ϕ de $\mathcal{L}$ on a $M \models \phi$.

Fixons une signature σ, $\mathcal{L}$ une sous-classe des formules du premier ordre et $\mathcal{S}$ une sous-classe des σ-structures.

Le problème de model-checking est de décider si pour une σ structure $M \in \mathcal{S}$ et une formule ϕ de $\mathcal{L}$ on a $M \models \phi$.

Théorème

Décider si une formule ϕ est vraie sur un modèle fini M peut se faire en temps $\bigcirc(|\phi|.|M|^d.|D|^k)$ ou D est le domaine de M, k la profondeur de quantification de ϕ et d le nombre de variables libres.

Quand le problème du model checking se fait sur un modèle fixé, généralement très simple comme $\{0, 1\}$, on parle de *CSP* pour contraint satisfaction problem.

Quand le problème du model checking se fait sur un modèle fixé, généralement très simple comme $\{0, 1\}$, on parle de *CSP* pour contraint satisfaction problem.

Théorème (Schaeffer : théorème dichotomique)

On peut tester la satisfiabilité des formules affines, 0-valide, 1-valide, Horn, anti-Horn, bijonctives en temps polynomial. Dans tout les autres cas c'est NP*-complet.*

Quand le problème du model checking se fait sur un modèle fixé, généralement très simple comme $\{0, 1\}$, on parle de *CSP* pour contraint satisfaction problem.

Théorème (Schaeffer : théorème dichotomique)

On peut tester la satisfiabilité des formules affines, 0-valide, 1-valide, Horn, anti-Horn, bijonctives en temps polynomial. Dans tout les autres cas c'est NP*-complet.*

Contre-intuitif avec le théorème de Ladner.

Une formule de la logique existentielle du second ordre s'écrit sur la signature σ :

$$\exists S_1 \ldots \exists S_k \phi(\sigma, S_1, \ldots, S_k)$$

ou $S_1, \ldots, S_k$ sont des symboles de fonction ou de relation et ϕ une formule du premier ordre.

Une formule de la logique existentielle du second ordre s'écrit sur la signature σ :

$$\exists S_1 \dots \exists S_k \phi(\sigma, S_1, \dots, S_k)$$

ou $S_1, \dots, S_k$ sont des symboles de fonction ou de relation et ϕ une formule du premier ordre.

Par exemple sur la signature $\varnothing$:
$\exists S, \exists Blanc, \exists Noir, successeur(S) \wedge (\forall xy, x = S(y) \Rightarrow (Noir(x) \Rightarrow Blanc(y) \wedge Blanc(x) \Rightarrow Noir(y))) \wedge (\forall xy, min(x) \wedge max(y) \Rightarrow Noir(x) \wedge Blanc(y))$

Une formule de la logique existentielle du second ordre s'écrit sur la signature σ :

$$\exists S_1 \dots \exists S_k \phi(\sigma, S_1, \dots, S_k)$$

ou $S_1, \dots, S_k$ sont des symboles de fonction ou de relation et ϕ une formule du premier ordre.

Par exemple sur la signature $\varnothing$:
$\exists S, \exists Blanc, \exists Noir, successeur(S) \wedge (\forall xy, x = S(y) \Rightarrow (Noir(x) \Rightarrow Blanc(y) \wedge Blanc(x) \Rightarrow Noir(y))) \wedge (\forall xy, min(x) \wedge max(y) \Rightarrow Noir(x) \wedge Blanc(y))$

Cette formule exprime la parité.

Théorème (Fagin 73)

Une propriété est $\exists SO$-définissable si et seulement si il est dans NP.

Théorème (Fagin 73)

Une propriété est $\exists SO$-définissable si et seulement si il est dans NP.

Premier théorème de logique descriptive.

Théorème (Fagin 73)

Une propriété est $\exists SO$-définissable si et seulement si il est dans NP.

Premier théorème de logique descriptive.
Plein d'autres langages logiques qui caractérisent une classe de complexité.
Bientôt une caractérisation de P par David pour le prix Turing.

Théorème (Fagin 73)

Une propriété est $\exists SO$-définissable si et seulement si il est dans NP.

Premier théorème de logique descriptive.
Plein d'autres langages logiques qui caractérisent une classe de complexité.

Permet de passer d'un problème de séparation de classes de complexité à un problème d'expressivité de logique.

Plan de l'exposé

Jusqu'à maintenant on a étudié des problèmes de décision.
On veut aussi étudier la difficulté de calculer des fonctions !

Jusqu'à maintenant on a étudié des problèmes de décision.
On veut aussi étudier la difficulté de calculer des fonctions !

Definition

Une fonction est dans *FP* si il existe une machine de Turing qui en partant de l'argument s'arrête sur le résultat de la fonction en temps polynomial.

Jusqu'à maintenant on a étudié des problèmes de décision.
On veut aussi étudier la difficulté de calculer des fonctions !

Definition

Une fonction est dans FP si il existe une machine de Turing qui en partant de l'argument s'arrête sur le résultat de la fonction en temps polynomial.

Definition

Une fonction f est dans $\#\mathrm{P}$ si il existe un prédicat $P(x, y)$ décidable en temps polynomial et un polynome Q, tel que $f(x) = \sharp \left\{ y \in \Sigma^{Q(|x|)} \mid P(x, y) \right\}$.

- Le déterminant se calcule en temps polynomial.

- Le déterminant se calcule en temps polynomial.
- Le permanent d'une matrice M est dans $\#\mathrm{P}$.

$$\mathrm{Per(M)} = \sum_{\sigma \in \mathfrak{S}_n} \prod_{i=1}^{n} m_{i,\sigma(i)}$$

- Le déterminant se calcule en temps polynomial.
- Le permanent d'une matrice M est dans $\#\mathrm{P}$.

$$\mathrm{Per(M)} = \sum_{\sigma \in \mathfrak{S}_n} \prod_{i=1}^{n} m_{i,\sigma(i)}$$

- On représente un groupe d'hommes par un ensemble de sommet E_1 et un groupe de femmes par un ensemble de sommets E_2. On trace une arête entre un homme et une femme si il veulent bien se "marier" ensemble. On veut décider s'il existe une manière que tout le monde se trouve un partenaire, et combien il y a de tels arrangements valides. Ce sont les problèmes $Bip - Matching$ et $\sharp.Bip - Matching$.

Definition

Soient f et g deux fonctions on dit que g se réduit à f si $g \in FP^f$, c'est à dire si on peut calculer g en temps polynomial en utilisant la fonction f comme un oracle.

Definition

Soient f et g deux fonctions on dit que g se réduit à f si $g \in FP^f$, c'est à dire si on peut calculer g en temps polynomial en utilisant la fonction f comme un oracle.

On peut définir une notion de complétude pour $\#\mathrm{P}$.
Tout les problèmes NP complets donnent naissance à des problèmes $\#\mathrm{P}$ complets.

Théorème (Valiant)

Le problème Bip − Matching est dans P *alors que* $\sharp$*.Bip − Matching est dans* $\#\mathrm{P}$

Théorème (Valiant)

Le problème Bip − Matching est dans P *alors que* $\sharp$*.Bip − Matching est dans* $\#\mathrm{P}$

Théorème (Kasteleyn)

Sur les graphes planaires $\sharp$*.Bip − Matching est dans FP.*

Soit P un prédicat tel que $P(x, y)$ est décidable en temps polynomial en $|x|$ et $|y| \leq Q(|x|)$ avec Q un polynome fixé.

Le problème est de déterminer l'ensemble $A(x) = \{y \mid P(x, y)\}$ Les problèmes de cette forme constituent **Enum**$\cdot\mathrm{P}$.

On veut trouver $A(x)$ aussi vite que possible..
Comme c'est un processus dynamique, on veut que le temps entre la sortie de deux solutions soit aussi court que possible.

Definition

Un problème $\textsc{Enum}\cdot A$ est décidable en temps incrémental polynomial **IncP**, si il y a un algorithme qui pour chaque instance x et chaque entier $k \leq |A(x)|$ renvoie la $(k+1)^e$ solution en temps polynomial en $|x|$ et k.

Definition

Un problème $\textsc{Enum}\cdot A$ est décidable en temps incrémental polynomial **IncP**, si il y a un algorithme qui pour chaque instance x et chaque entier $k \leq |A(x)|$ renvoie la $(k+1)^e$ solution en temps polynomial en $|x|$ et k.

$\textsc{Enum}\cdot\textsc{MatroidCircuit}$ est dans **IncP**.

Definition

Un problème $\text{Enum}\cdot\mathcal{A}$ est décidable avec délai polynomial **DelayP**, si il y a un algorithme qui sur chaque instance x renvoie $A(x)$ en un temps polynomial en $|x|$ entre deux solutions générées.

Definition

Un problème $\textsc{Enum}\cdot\mathcal{A}$ est décidable avec délai polynomial **DelayP**, si il y a un algorithme qui sur chaque instance x renvoie $A(x)$ en un temps polynomial en $|x|$ entre deux solutions générées.

$\textsc{Enum}\cdot\textsc{MaxIndependentSet}$ est dans **DelayP** et l'algorithme énumère les solutions dans l'ordre lexicographique.
Dans l'ordre anti-lexicographique, le problème n'est pas dans **DelayP** [David S. Johnson, Christos H. Papadimitriou and Mihalis Yannakakis].

Plan de l'exposé

Matroids have been design to abstract the notion of dependence.

Definition

A matroid is a pair $(E, \mathcal{I})$, E is a finite set and $\mathcal{I}$ is included in the power set of E. Elements of $\mathcal{I}$ are said to be independent sets, the others are dependent sets.
A matroid must satisfy the following axioms :

Matroids have been design to abstract the notion of dependence.

Definition

A matroid is a pair $(E, \mathcal{I})$, E is a finite set and $\mathcal{I}$ is included in the power set of E. Elements of $\mathcal{I}$ are said to be independent sets, the others are dependent sets.
A matroid must satisfy the following axioms :

1. $\varnothing \in \mathcal{I}$

Matroids have been design to abstract the notion of dependence.

Definition

A matroid is a pair $(E, \mathcal{I})$, E is a finite set and $\mathcal{I}$ is included in the power set of E. Elements of $\mathcal{I}$ are said to be independent sets, the others are dependent sets.
A matroid must satisfy the following axioms :

1. $\varnothing \in \mathcal{I}$
2. If $I \in \mathcal{I}$ and $I' \subseteq I$, then $I' \in \mathcal{I}$

Matroids have been design to abstract the notion of dependence.

Definition

A matroid is a pair $(E, \mathcal{I})$, E is a finite set and $\mathcal{I}$ is included in the power set of E. Elements of $\mathcal{I}$ are said to be independent sets, the others are dependent sets.
A matroid must satisfy the following axioms :

1. $\varnothing \in \mathcal{I}$
2. If $I \in \mathcal{I}$ and $I' \subseteq I$, then $I' \in \mathcal{I}$
3. If I_1 and I_2 are in $\mathcal{I}$ and $|I_1| < |I_2|$, then there is an element e of $I_2 - I_1$ such that $I_1 \cup e \in \mathcal{I}$.

The first concrete example of matroid is the vector matroid.

Let A be a matrix, the ground set E is the set of the columns and a set of columns is independent if the vectors are linearly independent.
In fact matroid have been designed from this example.

The first concrete example of matroid is the vector matroid.

Let A be a matrix, the ground set E is the set of the columns and a set of columns is independent if the vectors are linearly independent.
In fact matroid have been designed from this example.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

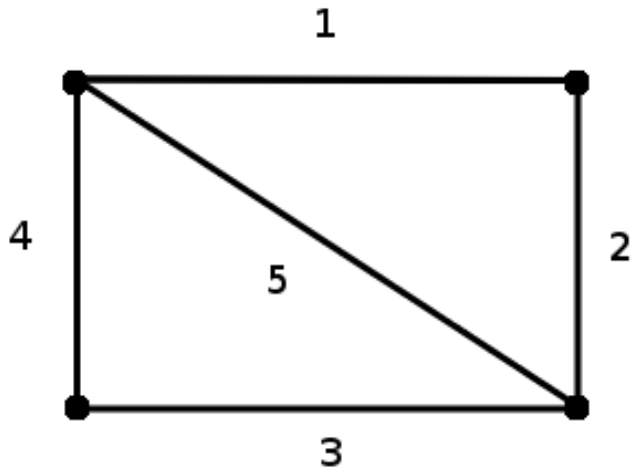
Here the set $\{1, 2, 4\}$ is independent and $\{1, 2, 3\}$ is dependent.

The second example is the cycle matroid of a graph.

Let G be a graph, the ground set of his cycle matroid is E the set of his edges.
A set is said to be dependent if it contains a cycle.

The second example is the cycle matroid of a graph.

Let G be a graph, the ground set of his cycle matroid is E the set of his edges.
A set is said to be dependent if it contains a cycle.



Here the set $\{1, 2, 4\}$ is independent and $\{1, 2, 3, 4\}$ or $\{1, 2, 5\}$ is dependent.

Lemme

Any cycle matroid is a representable matroid, i.e. it is isomorph to a vector matroid.

Label the vertices of a graph by $1, \ldots, n$ and the edges by $1, \ldots, m$.

Lemme

Any cycle matroid is a representable matroid, i.e. it is isomorph to a vector matroid.

Label the vertices of a graph by $1, \ldots, n$ and the edges by $1, \ldots, m$. We build the matrix A such as $A_{i,j} = 1$ iff the edge j is incident to the vertex i.

Lemme

Any cycle matroid is a representable matroid, i.e. it is isomorph to a vector matroid.

Label the vertices of a graph by $1, \ldots, n$ and the edges by $1, \ldots, m$.
We build the matrix A such as $A_{i,j} = 1$ iff the edge j is incident to the vertex i.
The dependence relation is the same over the edges and over the vectors representing the edges.

Lemme

Any cycle matroid is a representable matroid, i.e. it is isomorph to a vector matroid.

Label the vertices of a graph by $1, \ldots, n$ and the edges by $1, \ldots, m$. We build the matrix A such as $A_{i,j} = 1$ iff the edge j is incident to the vertex i.
The dependence relation is the same over the edges and over the vectors representing the edges.

This matrix represents the former graph :

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Definition

A circuit is a dependent set minimal for the inclusion.

Definition

A circuit is a dependent set minimal for the inclusion.

Théorème

Computing the circuits of a matroid is in **IncP**.

We can define the MSO_M logic over the matroids. It's the second order logic with signature $\{indep\}$, a unary relation over second order variables.
All the second order variables have to be unary (monadic).

We can define the MSO_M logic over the matroids. It's the second order logic with signature $\{indep\}$, a unary relation over second order variables.
All the second order variables have to be unary (monadic).

$$Circuit(X) = \neg indep(X) \cup \forall Y\,(Y \not\subseteq X \vee X = Y \vee indep(Y))$$

We can define a subclasses of the represented matroids called the matroids of branch-width k .

Théorème (Moi)

We can decide in polynomial time any MSO_M formulas over the matroids of branch-width k. We can also enumerate with polynomial delay all the assignation satisfying a MSO_M formula with free variables, if we are restricted to the matroid of branch-width k.

Merci !