Théorème de Mahaney

Marc Jeanmougin

14 janvier 2010

Notion de circuits

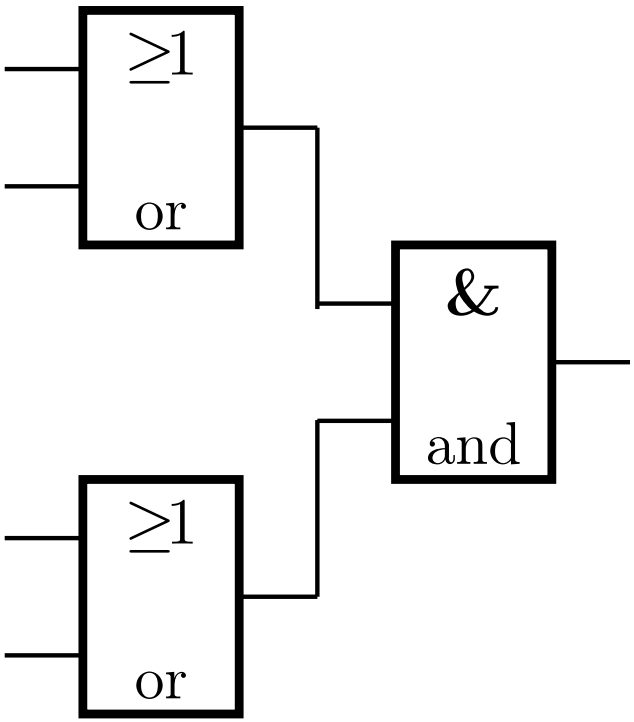
Définition

Un *circuit* est un graphe acyclique orienté avec des portes appartenant à une base de portes.

Exemples de bases

La base la plus répandue et couramment utilisée est $B_0 = \{\neg, \wedge, \vee\}$, une base minimale (pour obtenir toutes les fonctions logiques) est $B_{min} = \{\neg\wedge\}$, une autre base utile est $B_1 = \{\neg, (\wedge^n)_{n\in\mathbb{N}}, (\vee^n)_{n\in\mathbb{N}}\}$.

Exemple



Familles de circuits

Définition

Une famille de circuits $C = \{C_i\}_{i \in \mathbb{N}}$ est un ensemble infini de circuits tels que C_i a i portes d'entrées.

Exemple

$\{\{\bigwedge_n\}, n \in \mathbb{N}\}$ est une famille de circuits.

Familles de circuits

Définition

Une famille de circuits $C = \{C_i\}_{i \in \mathbb{N}}$ est un ensemble infini de circuits tels que C_i a i portes d'entrées.

Exemple

$\{\{\bigwedge_n\}, n \in \mathbb{N}\}$ est une famille de circuits.

Classes de complexité uniformes

Les familles *uniformes* de circuit sont celles pour lesquelles il existe une machine de Turing qui pour une entrée i a pour sortie une description de C_i.

Equivalences

Equivalences

Circuits à n entrées $\longleftrightarrow$ $f : \{0,1\}^n \to \{0,1\}$

Familles de circuits $\longleftrightarrow$ $L : \{0,1\}^* \to \{0,1\}$

Equivalences

Equivalences

Circuits à n entrées $\longleftrightarrow$ $f : \{0,1\}^n \to \{0,1\}$

Familles de circuits $\longleftrightarrow$ $L : \{0,1\}^* \to \{0,1\}$

Un bon modèle ?

Equivalences

Equivalences

Circuits à n entrées $\longleftrightarrow$ $f : \{0,1\}^n \to \{0,1\}$

Familles de circuits $\longleftrightarrow$ $L : \{0,1\}^* \to \{0,1\}$

Un bon modèle ?

L non forcément calculable...

Complexité

Deux mesures de la complexité

- La profondeur

Complexité

Deux mesures de la complexité

- La profondeur
- La taille

Complexité

Deux mesures de la complexité

- La profondeur
- La taille

ATTENTION ! Ceci est très dépendant de la base choisie. Traditionnellement, on choisit B_0

Complexités triviales

Complexité

Deux mesures de la complexité

- La profondeur
- La taille

ATTENTION ! Ceci est très dépendant de la base choisie. Traditionnellement, on choisit B_0

Complexités triviales

- profondeur 3 sur B_1

Complexité

Deux mesures de la complexité

- La profondeur
- La taille

ATTENTION ! Ceci est très dépendant de la base choisie. Traditionnellement, on choisit B_0

Complexités triviales

- profondeur 3 sur B_1
- taille $\theta(2^n)$ sur B_1

Complexité

Deux mesures de la complexité

- La profondeur
- La taille

ATTENTION ! Ceci est très dépendant de la base choisie. Traditionnellement, on choisit B_0

Complexités triviales

- profondeur 3 sur B_1
- taille $\theta(2^n)$ sur B_1
- profondeur $\theta(n)$ sur B_0

Complexité

Deux mesures de la complexité

- La profondeur
- La taille

ATTENTION ! Ceci est très dépendant de la base choisie. Traditionnellement, on choisit B_0

Complexités triviales

- profondeur 3 sur B_1
- taille $\theta(2^n)$ sur B_1
- profondeur $\theta(n)$ sur B_0
- taille $\theta(n2^n)$ sur B_0

La classe $P/Poly$

Définition

On peut définir la classe $P/Poly$ de deux manières équivalentes :

- $P/Poly = \bigcup_{i \in \mathbb{N}} SIZE(n^i)$

La classe $P/Poly$

Définition

On peut définir la classe $P/Poly$ de deux manières équivalentes :

- $P/Poly = \bigcup_{i \in \mathbb{N}} SIZE(n^i)$
- $P/Poly = \{L | \exists$ une machine de Turing et une suite α_n de "conseils", $|\alpha_n| < p(n)$ et $M(x, \alpha_{|x|}) = \chi_L(x)\}$

A quoi ça sert ?

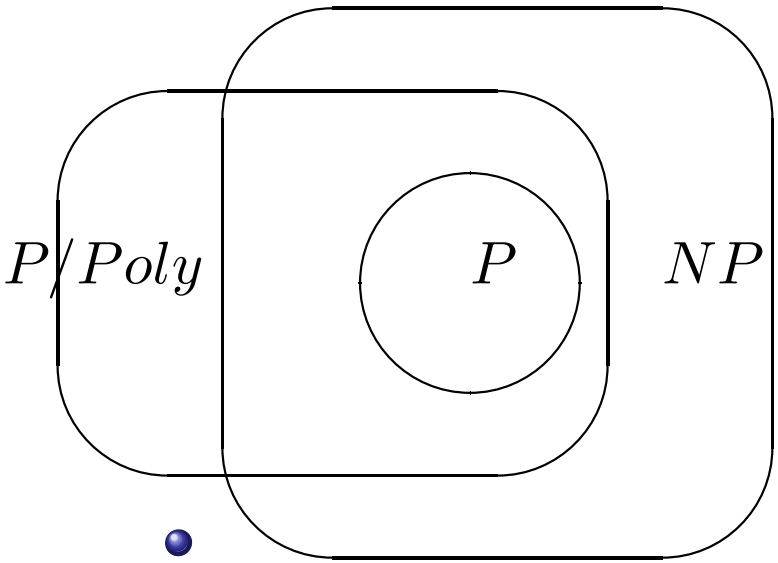
- $P \subset P/Poly$

A quoi ça sert ?

- $P \subset P/Poly$
- $NP \not\subset P/Poly$?

A quoi ça sert ?

- $P \subset P/Poly$
- $NP \not\subset P/Poly$?



?

Langage peu dense

Définition

Un langage est dit peu dense si $|L \cap \{0,1\}^n| < p(n)$ pour un certain polynôme p.

Premier résultat

Théorème

Un langage NP-complet L est réductible à un langage peu dense ssi $NP \subset P/Poly$

$\Rightarrow$

On réduit L à un langage peu dense que l'on peut décrire par une famille polynomiale de circuits ...

$\Leftarrow$

Rapide : L, NP-Complet, est dans P/Poly, donc réductible à un langage peu dense.

Théorème de Mahaney

Théorème de Mahaney

Un langage $NP - Complet$ est réductible en temps polynomial à un langage peu dense si et seulement si $P = NP$

Demonstration $\Leftarrow$

Si $P = NP$, un langage $NP - complet$ est dans P... et on peut donc le ramener de manière polynomiale au problème d'appartenance au langage non-dense le plus simple, par exemple $\{1\}$.

Schéma de la preuve

Supposons qu'un langage $NP - Complet$ soit réductible en temps polynomial à un langage peu dense.
Soit LSAT un langage $NP - Complet$ que nous définirons plus loin.
Nous pouvons alors réduire LSAT à un langage peu dense en temps polynomial (en passant par L en temps polynomial).
On construira alors un algorithme... polynomial qui résoudra le problème SAT, d'où $P = NP$.

Preuve

On définit LSAT comme suit : Soit ϕ une expression booléenne à n variables, $x \in \{0,1\}^n$.
$(\phi, x) \in LSAT \iff \exists x' | x' \leq_{lexicographique} x$ et $\phi(x)$.
On a immédiatement $(\phi, 1^n) \in LSAT \iff \phi \in SAT$, d'où LSAT est $NP - Complet$. Comme précisé dans le schéma de la preuve,

on a LSAT réductible en temps polynomial à un langage peu dense S, par la fonction f.
Puisque f est calculable en temps polynomial, on a une borne polynomiale (polynôme $p(n)$) sur la longueur de la sortie de f.
On pose $q(n) = |S \cup \{0,1\}^{\leq p(n)}|$, polynômial car S est peu dense et p polynomial (un polynôme de polynôme est un polynôme) .
Soit l'arbre ayant pour racine ϵ et à chaque noeud étiqueté par v ayant pour fils $v0$ et $v1$, de profondeur n.

résoudre SAT en temps polynomial

$LIVE_0 \leftarrow \{\epsilon\}$
for $i = 1$ à $n-1$ **do**
  $LIVE_i \leftarrow \{$tous les fils de $LIVE_{i-1}\}$
  if $|LIVE_i| > q(n)$ **then**
    "élaguer" $LIVE_i$ jusqu'à ce qu'il contienne moins de $q(n)$ éléments.
  end if
end for
Renvoyer 1 ssi un des fils de $LIVE_{n-1}$ satisfait le problème.

L'élagage d'un ensemble de nœuds V est fait de la manière suivante : Calculer $Z_V = \{f(v)|v \in V\}$ puis répéter les étapes suivantes :

- Si $f(v_1) = f(v_2)$, retirer le plus grand des deux dans l'ordre lexicographique.
- Si Z_v contient plus de $q(n)$ valeurs, retirer le plus petit élément de V dans l'ordre lexicographique.

Fin de la démonstration : complexité

Il y a n niveaux de boucle et au plus $2q(n)$ noeuds considérés à chaque niveau, donc on "élague" au plus $2nq(n)$ fois, ce qui est polynomial. l'opération elle-même l'est également trivialement.
Il reste à prouver la correction de l'algorithme :
Montrons par induction qu'à chaque itération i, $LIVE_i$ contient un ancêtre du plus petit (dans l'ordre lexicographique) mot satisfaisant ϕ. C'est vrai pour $LIVE_0$.
Supposons la proposition au rang $i-1$. $LIVE_i$, avant l'élagage, satisfait toujours la condition...

Fin de la démonstration (2)

On a ensuite deux possibilités :

- $f(v_1) = f(v_2)$ et $v_1 < v_2$. On retire v_2 mais si v_2 avait eu un descendant satisfaisant le problème, alors v_1 également, d'où v_2 ne peut pas être ancêtre du plus petit dans l'ordre lexicographique. On conserve donc la propriété par cette opération.
- Si Z_{LIVE_i} contient plus de $q(n)$ valeurs, alors nous savons qu'il en existe au moins une valeur dans Z qui n'est pas ancêtre d'une solution (il y en a au plus $q(n)$). Donc il y a une valeur dans $LIVE_i$ qui n'est pas ancêtre d'une solution de LSAT. Donc puisque v_α n'est pas satisfaite, si v_0 est le plus petit mot de V, v_0 n'est évidemment pas solution de LSAT. Donc ce n'est pas un ancêtre de la plus petite solution de ϕ et on peut donc le retirer.

Conclusion

Ainsi, on conserve la propriété à toutes les étapes de l'algorithme. Donc $LIVE_{n-1}$ contient le père d'une solution de SAT, et il n'y en a qu'un nombre polynomial, qui sont chacun polynomialement testables.

Donc...

On a donc un algorithme polynomial qui résout SAT, donc

$$P = NP$$

.

On a donc démontré le théorème.□