

Calculabilité, logique et complexité

Chapitre 7

Classes de complexité

Chapitre 7 : Classes de complexité

1. Réduction et ensemble complet
2. Modèles de calcul
3. Classes de complexité
4. Relations entre classes de complexité
5. NP-complétude
- ~~6. NP-complétude (2)~~
7. Théorème de Cook: SAT est NP-complet
8. Quelques problèmes NP-complets
9. $P = NP$?
10. Interpréter la NP-complétude
11. Autres classes de complexité

Acquis d'apprentissage

A l'issue de ce chapitre, les étudiants seront capables de

- Comprendre, expliquer et appliquer les réductions algorithmique et fonctionnelle
- Montrer et expliquer les propriétés ainsi que les différences entre la réduction algorithmique et fonctionnelle
- Comprendre, expliquer et appliquer les concepts d'ensemble A-complet et A-difficile
- Comprendre et expliquer les classes de complexité $DTIME$, $DSPACE$, $NTIME$ et $NSPACE$; expliquer les relations entre ces classes de complexité
- Définir et expliquer les classes P et NP
- Comprendre, expliquer et justifier la réduction polynomiale
- Définir, comprendre et expliquer ce qu'est un problème NP -complet
- Comprendre et justifier les relations entre les classes P et NP
- Comprendre et expliquer l'énoncé du théorème de Cook ainsi que ces conséquences
- Comprendre les techniques pour montrer si un problème est NP -complet ; les appliquer à certains problèmes spécifiques
- Illustrer par des exemples concrets la NP -complétude
- Expliquer comment aborder dans la vie professionnelle les problèmes NP -complets

1. Réduction

Objectif

Déduire un algorithme pour un **problème** P à partir d'un algorithme d'un autre problème P'

Déduire un algorithme de décision pour un **ensemble** E à partir d'un algorithme de décision d'un autre ensemble E'

- permet de prouver la calculabilité
- permet de prouver la non calculabilité
- permet d'analyser le degré de non calculabilité
- permet de déduire la complexité
- permet d'analyser le degré de complexité

} calculabilité

} complexité

Existence de plusieurs méthodes de réduction

- réduction algorithmique
- réduction fonctionnelle
- réduction polynomiale

1. Réduction

Réduction algorithmique

Un ensemble A est *algorithmiquement réductible* à un ensemble B si à partir d'un algorithme permettant de reconnaître B , on peut construire un algorithme permettant de reconnaître A

Définition

Un ensemble A est *algorithmiquement réductible* à un ensemble B ($A \leq_a B$) si en supposant B récursif, alors A est récursif



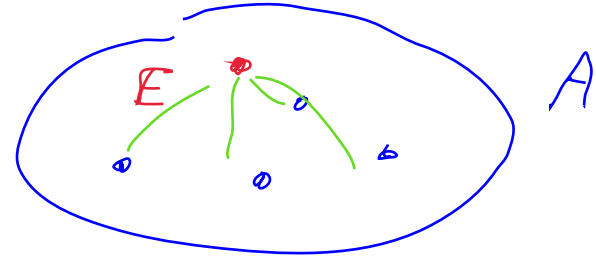
Relation de réductibilité

$A \leq B$: A réductible à B

- \leq : relation réflexive et transitive
- induit des classes d'équivalence


$$\begin{aligned} A &\leq A \\ A &\leq B \text{ et } B \leq C \\ \text{alors } A &\leq C \end{aligned}$$

1. Réduction



Ensemble complet

Un ensemble complet d'une classe d'ensembles est l'ensemble le “plus difficile” à décider parmi tous les ensembles de la classe

Si on trouve un algorithme pour cet ensemble, alors on peut reconnaître tous les autres ensembles de la classe 

Définition

Soit A une classe de problèmes

Un problème E est A -**complet** par rapport à une relation de réduction \leq ssi

1. $E \in A$
2. $\forall B \in A : B \leq E$

Un problème E est A -**difficile** par rapport à une relation de réduction \leq ssi

1. $\forall B \in A : B \leq E$

1. Réduction

Propriétés

- Si $A \leq_a B$ et B récursif, alors A récursif
- Si $A \leq_a B$ et A non récursif, alors B non récursif
- $A \leq_a \bar{A}$
- $A \leq_a B \Leftrightarrow \bar{A} \leq_a \bar{B}$
- Si A récursif, alors pour tout B , $A \leq_a B$
- Si $A \leq_a B$ et B récursivement énumérable, alors A *pas nécessairement* récursivement énumérable

$$\bar{K} \leq_a K$$

K récurs. énum.

\bar{K} non récurs. énum.

Exemples

- $DIAG \leq_a HALT$
- $HALT \leq_a DIAG$
- $HALT$ est r.e.-complet par rapport à \leq_a (r.e. = classes des ensembles récursivement énumérables)
- $DIAG$ est r.e.-complet par rapport à \leq_a

$$A \leq_a B$$

$$B \text{ est } O(n^2)$$

$$A \text{ est } O(?) \rightarrow \text{mon}$$

$$\text{obtain } A(x) \equiv \left[\begin{array}{l} \text{obtain } B(y) \\ O(n^2) \end{array} \right]$$



1. Réduction

Réduction fonctionnelle

Définition

Un ensemble A est **fonctionnellement réductible** à un ensemble B ($A \leq_f B$) ssi il existe une **fonction totale calculable** f telle que

$$a \in A \Leftrightarrow f(a) \in B$$

$$f(a) \in B \Rightarrow a \in A$$

$$f(a) \notin B \Rightarrow a \notin A$$

Pour décider si un élément appartient à A , il suffit de

- calculer $f(a)$
- et tester si $f(a) \in B$

$$\underbrace{}_{O(n^2)}$$

$$B \quad O(n^2)$$

$$A \quad O(?)$$

↳ complexité du calcul de f
+ complexité de B

1. Réduction

Propriétés

- Si $A \leq_f B$ et B récursif, alors A récursif
- Si $A \leq_f B$ et A non récursif, alors B non récursif
- Si $A \leq_f B$ et B récursivement énumérable, alors A récursivement énumérable
- $A \leq_f B \Leftrightarrow \bar{A} \leq_f \bar{B}$
- Si A récursif, alors pour tout B , $A \leq_f B$
- Pas nécessairement $A \leq_f \bar{A}$
- Si $A \leq_f B$, alors $A \leq_a B$
- $A \leq_a B$ n'implique *pas nécessairement* $A \leq_f B$

+ restrictif car
1) obligatⁿ d'appartenance
à B : $f(x) \in B$

2) 1 seule fois
↑
on exécute l'algo

Exemples

- $DIAG \leq_f HALT$
- $HALT \leq_f DIAG$
- $DIAG$ est r.e.-complet par rapport à \leq_f

1. Réduction

Différences entre \leq_a et \leq_f
 → résoluc^o algo
 → résoluc^o fonctionnelle

- $A \leq_a B$: point de vue *calculabilité*
Pour décider si $a \in A$, on peut utiliser quand on veut et autant de fois que l'on veut, le fait que B soit récursif
- $A \leq_f B$: point de vue *complexité*
Pour décider si $a \in A$, on doit utiliser le schéma d'algorithme suivant :
 - input a
 - $a' := f(a)$
 - si $a' \in B$ alors output OUI ($a \in A$)
sinon output NON ($a \notin A$)

On doit donc utiliser une seule fois le test $f(a) \in B$ et en dernier lieu

On ne peut rien faire après ce test : pas de transformation de résultat permise

Si on connaît la complexité de la fonction f et du problème de décision de l'ensemble B , alors on peut déduire des informations sur la complexité de A

2. Modèles de calcul

Théorie “classique” de la complexité:

- utilisation du modèle des machines de Turing
- permet une définition précise de complexité temporelle (nombre de transitions)
- permet une définition précise de complexité spatiale (nombre de cases utilisées)

Mais

- modèle peu intuitif pour un informaticien
- complexité d'une machine de Turing est précise mais éloignée d'une complexité “en pratique”
- on s'intéresse au ordre de grandeur (grand O)
- on s'intéresse à la frontière : problèmes intrinsèquement complexes
- modèle des machines de Turing simple à utiliser dans certaines preuves

2. Modèles de calcul

Possibilité d'utiliser le modèle des **programmes Java**

- déterministes
- non déterministes

En pratique, utiliser le modèle le plus adéquat à la situation à traiter

- Machine de Turing pour les preuves complexes
- Langage Java pour les définitions et les intuitions

Thèse

Tous les modèles de complexité (avec des hypothèses raisonnables de mesure de coût) ont entre eux des complexités spatiales et temporelles reliées de façon *polynomiale*.

Conséquence

Si un problème est pratiquement faisable dans un modèle (non déterministe), alors il est pratiquement faisable dans tous les modèles (non déterministes)

3. Classes de complexité

*Classes basées
sur un modèle déterministe*

DTIME(f)

Famille des ensembles *rékursifs* pouvant être décidés
par un programme Java de complexité temporelle $O(f)$

$$\text{DTIME}(n!) \neq \text{DTIME}(n^2)$$

Java M. Turing

Tous les programmes se terminent toujours

DSPACE(f)

Famille des ensembles *rékursifs* pouvant être décidés
par un programme Java de complexité spatiale $O(f)$



3. Classes de complexité

*Classes basées
sur un modèle non-déterministe*

NTIME(f)

Famille des ensembles *ND-réursifs* pouvant être décidés par un programme Java *non déterministe* de complexité temporelle $O(f)$

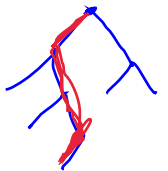
On considère la complexité de la branche d'exécution la plus longue

Toutes les branches sont donc finies et de complexité $O(f)$

NSPACE(f)

Famille des ensembles *ND-réursifs* pouvant être décidés par un programme Java *non déterministe* de complexité spatiale $O(f)$

Même remarque que ci-dessus pour le calcul de la complexité



3. Classes de complexité

Classe P \rightarrow indépendante du modèle de calcul

$$P = \bigcup_{i \geq 0} \text{DTIME}(n^i)$$

Famille des ensembles *rékursifs* pouvant être décidés
par un programme (Java) de complexité temporelle *polynomiale*
 \mathcal{PT}

Classe NP

$$NP = \bigcup_{i \geq 0} \text{NTIME}(n^i)$$

Famille des ensembles *ND-rékursifs* pouvant être décidés
par un programme Java *non déterministe* de complexité temporelle *polynomiale*

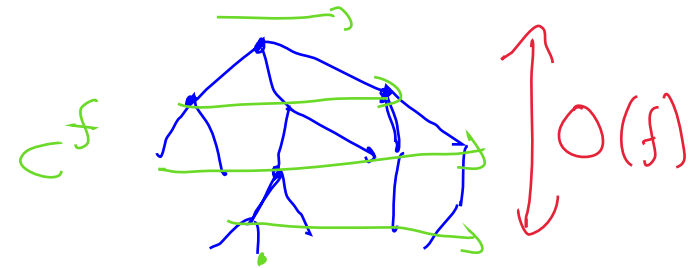
4. Relations entre classes de complexité

Déterministe vs non déterministe

Si $A \in \text{NTIME}(f)$

Alors $A \in \text{DTIME}(c^f)$

$\text{NTIME}(n^2)$
 $\text{DTIME}(7^{n^2})$



Si $A \in \text{NSPACE}(f)$

Alors $A \in \text{DSPACE}(f^2)$

(Savitch theorem)

4. Relations entre classes de complexité

Time vs Space

Si $A \in \text{NTIME}(f)$

Alors $A \in \text{NSPACE}(f)$

Si $A \in \text{DTIME}(f)$

Alors $A \in \text{DSPACE}(f)$



Si $A \in \text{NSPACE}(f)$

Alors $A \in \text{NTIME}(c^f)$

Si $A \in \text{DSPACE}(f)$

Alors $A \in \text{DTIME}(c^f)$

Existe-t-il une borne inférieure dans ce dernier cas ?

4. Relations entre classes de complexité

Hiérarchie de complexités

Pour toute fonction totale calculable f , il existe un ensemble A récursif tel que $A \notin \text{DTIME}(f)$

- Preuve par diagonalisation
- Mêmes résultats pour NTIME , DSPACE et NSPACE

Il existe donc toujours des problèmes plus complexes qu'une complexité donnée

5. NP-complétude

- Problèmes intrinsèquement complexes ont une complexité non polynomiale
- En pratique, beaucoup de problèmes intéressants s'avèrent être de complexité temporelle non déterministe polynomiale

Question fondamentale de la théorie de la complexité :

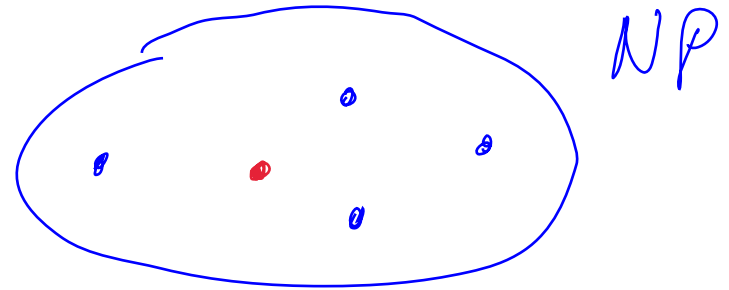
Si il existe un algorithme *non déterministe polynomiale*, existe-t-il alors un algorithme *déterministe polynomial* résolvant ce même problème ?

Si un ensemble A appartient à NP

A appartient-il alors à P ?

$P = NP ?$

5. NP-complétude



On sait que $P \subseteq NP$

Comment démontrer

- $NP \subseteq P$
- Ou non ($NP \subseteq P$)

Prendre un élément E “représentatif” de NP (le plus difficile) et essayer de démontrer

- $E \in P \rightarrow$ alors $P = NP$
- Ou $E \notin P \rightarrow$ alors $P \neq NP$

Choisir un élément E qui soit NP -complet par rapport à une relation de réduction \leq

- $E \in NP$
- $\forall B \in NP : B \leq E$

Quelles relation de réduction choisir ?

5. NP-complétude

Réduction algorithmique \leq_a

- $A \leq_a B$: en supposant B récursif, A est récursif
- Si on arrive à montrer que $E \in P$, on ne sait pas si $P = NP$
car pour les autres éléments de NP , la réduction ne permet pas d'affirmer que ceux-ci sont dans P (utilisation non bornée de tests d'appartenance à E)

Réduction fonctionnelle \leq_f

- $A \leq_f B$ si il existe une fonction totale calculable f telle que $a \in A \Leftrightarrow f(a) \in B$
- Si on arrive à montrer que $E \in P$, on ne sait pas si $P = NP$
car pour les autres éléments de NP , la réduction ne permet pas d'affirmer que ceux-ci sont dans P (le calcul de $f(a)$ peut prendre un temps non polynomial)

Nécessité d'affiner la réduction fonctionnelle

imposer calcul polynomial →

① calculer $f(a)$
② tester si $f(a) \in B$

✓ poly $B \in P$ →

5. NP-complétude

Réduction polynomiale

Un ensemble A est **polynomialement réductible** à un ensemble B ($A \leq_p B$) si il existe une **fonction totale calculable** f de *complexité temporelle polynomiale* telle que

$$a \in A \Leftrightarrow f(a) \in B$$

Propriétés

- \leq_p est une relation réflexive et transitive
- Si $A \leq_p B$ et $B \in P$ Alors $A \in P$
- Si $A \leq_p B$ et $B \in NP$ Alors $A \in NP$

5. NP-complétude

NP-complétude

Un problème E est *NP-complet* (par rapport à \leq_p) si

1. $E \in NP$
2. $\forall B \in NP : B \leq_p E$

Un problème E est *NP-difficile* (par rapport à \leq_p) si

1. $\forall B \in NP : B \leq_p E$

5. NP-complétude

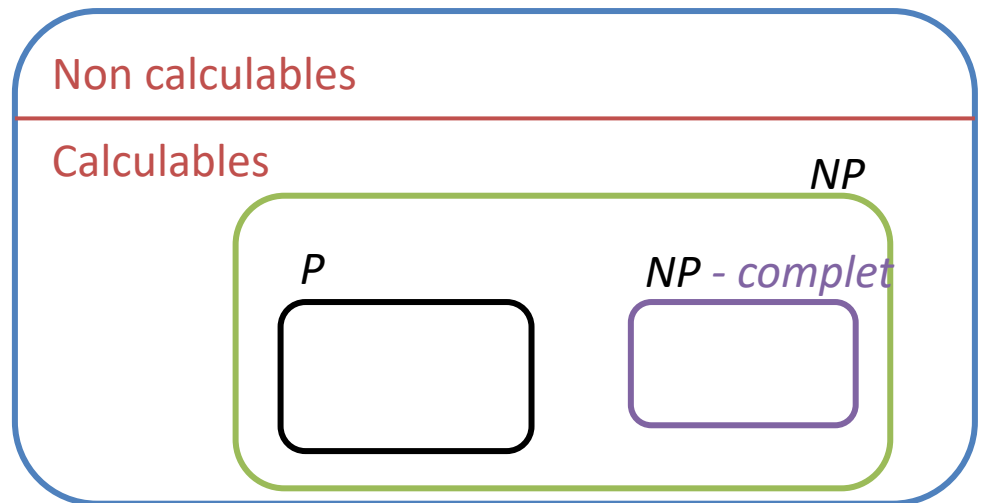
Propriétés

Soit E un ensemble NP -complet

- $E \in P$ ssi $P = NP$
- $E \notin P$ ssi $P \neq NP$
- Si $E \leq_p B$ et $B \in NP$ alors B est NP -complet

En supposant $P \neq NP$

Problèmes



5. NP-complétude

Questions

- Que recouvre la classe NP ?
- Peut-on déterminer des ensembles NP-complets ?
- Est-ce que $P = NP$?
- Quelle est la structure de NP ?

6. NP complétude (2)



On peut définir et présenter la classe *NP* d'une manière différente

Problème de décision

Soit A un ensemble récursif

Le problème de décision associé à A consiste à

décider, étant donné une donnée x ,

- si $x \in A$ (réponse OUI)
- ou si $x \notin A$ (réponse NON)

ce problème est dénoté $A(x)$

Un problème de décision peut être interprété comme un prédicat $A(x)$

Exemples

$SAT(x)$: la formule x est-elle satisfaisable ?

$TS(g, B)$: le graphe g possède-t-il un circuit de longueur inférieure ou égale à B

6. NP complétude (2)

Classe P

La classe P est la classe des problèmes de décision pouvant être calculé par un algorithme polynomial

Classe NP

La classe NP est la classe des problèmes de décision $A(x)$ pouvant s'exprimer sous la forme

$$\exists y B(x,y)$$

tel que

- $B(x,y) \in P$
- le domaine de y est fini et peut être généré, de manière non déterministe, en un temps polynomial

6. NP complétude (2)

Calcul d'un problème NP

Pour décider $A(x)$

1. calculer y (de manière non déterministe)
2. déterminer $B(x,y)$

Les deux définitions de la classe NP sont équivalentes

6. NP complétude (2)

Décision vs résultat

Problème de décision $A(x) = \exists y B(x,y)$

Pour **décider** $A(x)$

1. calculer y (de manière non déterministe)
2. si $B(x,y)$ alors output OUI
sinon output NON

Comment obtenir comme résultat la valeur y qui vérifie le prédicat $B(x,y)$?

Comme la taille de y est polynomiale en la taille de x , la seconde ligne de l'algorithme suivant a une complexité polynomiale en x

Pour **calculer** $A(x)$

1. calculer y (de manière non déterministe)
2. si $B(x,y)$ alors output y
sinon output NON

6. NP complétude (2)

Résultat vs optimisation

Soit le problème de décision suivant de NP : $A(x,b) = \exists y (B(x,y) \text{ et } f(x,y) \leq b)$

La fonction f est une *fonction d'évaluation*

A ce problème correspond un problème **d'optimisation** :

$OPT_A(x)$: y_m tel que

- $B(x, y_m)$
- $\forall y : B(x, y) \Rightarrow f(x, y_m) \leq f(x, y)$

ou encore

- $B(x, y_m)$
- $f(x, y_m) = \min \{ f(x, y) \mid B(x, y) \}$

Sous l'hypothèse

$$\forall x : \exists M : \forall y : B(x, y) \Rightarrow f(x, y) \leq M$$

(la fonction d'évaluation est bornée par une valeur M “pas trop grande”)

Le problème d'optimisation peut lui aussi être calculé en un temps (non déterministe) polynomial. Comment ?

7. Théorème de Cook : SAT est NP-complet

SAT: satisfaction des formules propositionnelles

Soit $W(A_1, \dots, A_n)$ une formule propositionnelle dont les variables sont A_1, \dots, A_n

Existe-t-il des valeurs logiques (true, false) pour les variables A_1, \dots, A_n telles que $W(A_1, \dots, A_n)$ soit vraie ?

Exemple

Soit la formule propositionnelle

$$(\neg A \vee B) \Rightarrow (A \wedge B)$$

Cette formule est satisfaite pour $A = \text{true}$ et $B = \text{true}$

7. Théorème de Cook : SAT est NP-complet

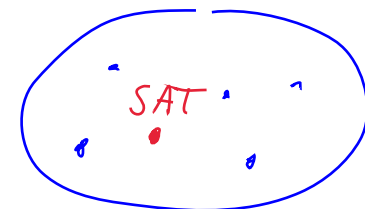
Formalisation

- Connecteurs logiques : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)$
- Variables A_i représentée par l'entier i (décimal ou binaire)
- Une formule $W(A_1, \dots, A_m)$ est **satisfaisable** si il existe des valeurs logiques (true, false) pour les variables A_1, \dots, A_m tel que $W(A_1, \dots, A_m)$ soit vraie
- Longueur d'une formule W :
 - soit n = nombre d'occurrence des variables
 - $\text{taille}(W) = O(n \log n)$
↪ représ. les variables propositionnelles

SAT = ensemble des formules propositionnelles satisfaisables

7. Théorème de Cook : SAT est NP-complet

NP





Théorème : SAT est NP-complet

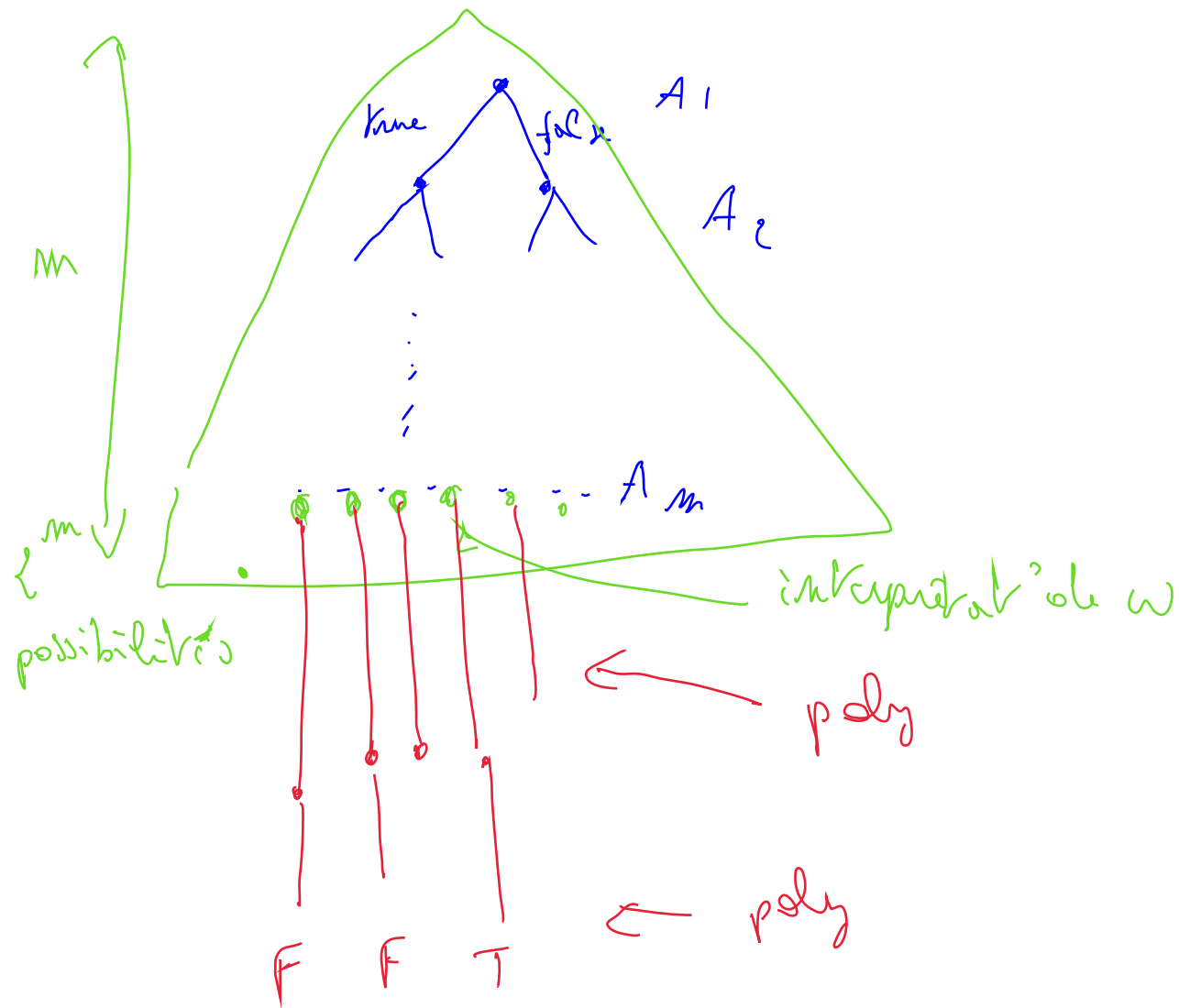


(1) $SAT \in NP$

Il existe un programme non déterministe de complexité polynomiale capable de reconnaître si une formule $W(A_1, \dots, A_m)$ est satisfaisable

(avec W contenant n occurrences de variables, $\text{taille}(W) = O(n \log n)$)

- Générer (de façon non déterministe) une séquence de m valeurs logiques
Complexité: $O(m)$ avec $m \leq n$ 
- Substituer les occurrences des variables A_i par leur valeur
Complexité : $O(n \log n)$ 
- Evaluer l'expression (par technique de réduction)
Complexité polynomiale



7. Théorème de Cook : SAT est NP-complet

(2) $\forall B \in NP : B \leq_p SAT$

Soit $B \in NP$

Il existe une machine de Turing non déterministe M qui reconnaît en un temps polynomial $p(n)$, si un élément x appartient à B , avec

- p est un polynôme
- n est la longueur de la donnée x

Dans toutes ses branches, la machine M s'arrête donc avant $p(n)$ mouvements

(2.1) Réalisation d'une transformation

La donnée x est transformée en une formule propositionnelle W_x

- taille de W_x : $O(p(n))$ symboles
- transformation polynomiale : la construction de W_x peut se faire en un temps polynomial par rapport à n
- $x \in B \Leftrightarrow W_x \in SAT$

7. Théorème de Cook : SAT est NP-complet

(2.2) Construction de la formule W_x

Principe: construction (automatique), à partir d'une donnée x et d'une machine de Turing non déterministe M , d'une formule W_x dont les variables caractérisent le fonctionnement de la machine M sur la donnée x

8. Quelques problèmes NP-complets

Comment prouver la NP-complétude ?

Deux techniques :

- Réaliser une preuve semblable à celle pour SAT (approche difficile)
- Utiliser la propriété : si $E \leq_p B$ avec E NP-complet et $B \in NP$
alors B est NP-complet

Les problèmes considérés sont des problèmes de *décision*

En pratique, on est intéressé par des algorithmes qui construisent des *solutions*

Si un problème de décision est NP-complet, le problème associé consistant à trouver une solution est au moins aussi complexe

“la formule est-elle satisfaisable”

vs. “déterminer une valeur des variables telle que la formule soit vraie”

En fait, trouver une solution ou trouver la meilleure solution (problème d'optimisation) n'est pas plus complexe (sous certaines conditions)

8. Quelques problèmes NP-complets

Problème du circuit hamiltonien HC

HC = ensemble des graphes possédant un circuit hamiltonien
(circuit hamiltonien = circuit parcourant une et une seule fois tous les sommets du graphe)

Le problème de la recherche d'un circuit eulerien est dans P

Problème du voyageur de commerce TS

Etant donnés

- un graphe de n noeuds (villes)
- les distances associée à chaque arc (paires de villes)
- B : un entier positif

Existe-t-il un circuit reliant les n noeuds (villes) et de longueur $\leq B$?

On peut montrer que **HC** \leq_p **TS**

8. Quelques problèmes NP-complets

Chemin le plus long entre deux sommets

Etant donnés

- un graphe où a et b sont parmi les nœuds
- les distances associées à chaque arc
- B : un entier positif

Existe-t-il un chemin reliant a et b et de longueur $\geq B$?

Si on change “et de longueur $\leq B$ ” alors le problème est dans P

3SAT

Problème de la satisfaisabilité de formules de la forme $C_1 \wedge C_2 \wedge \dots \wedge C_n$

Avec

- $C_i = L_{i1} \vee L_{i2} \vee L_{i3}$
- $L_{i1} = A_k$ ou $L_{i1} = \neg A_k$
- A_k est une variable propositionnelle

(forme normale conjonctive avec 3 variables par clause)

8. Quelques problèmes NP-complets

Couverture de sommets VC

Etant donnés

- un graphe
- B : un entier positif

Existe-t-il un sous-ensemble de sommets de taille $\leq B$ qui couvre tous les arcs du graphe (i.e. chaque arc a au moins une extrémité dans ce sous-ensemble)

Le problème de la couverture des arcs est dans P

Problème de la clique

Etant donnés

- un graphe
- B : un entier positif

Existe-t-il un sous-ensemble des sommets de taille $\geq B$ qui soit une clique (i.e. chaque paire de noeuds de ce sous-ensemble est reliée par un arc)

8. Quelques problèmes NP-complets

Nombre chromatique d'un graphe

Etant donnés

- un graphe
- B : un entier positif

Est-il possible de colorier les nœuds du graphe en utilisant au plus B couleurs, de telle sorte que deux nœuds adjacents (i.e. reliés par un arc) soient de couleurs différentes.

Problème de partition

Etant donné un ensemble A et une taille $s(a)$ pour chacun de ses éléments,

Existe-t-il un sous-ensemble $B \subseteq A$ tel que

$$\sum_{a \in B} s(a) = \sum_{a \in A - B} s(a)$$

8. Quelques problèmes NP-complets

Programmation entière

Programmation linéaire :

Etant donnés

- une matrice A $m \times n$ entiers
- un vecteur b de m entiers
- un vecteur d de n entiers
- c un entier
- un vecteur x de n variables rationnelles

Déterminer x tel que

$$d'.x \geq c$$
$$A.x \leq b$$

La programmation linéaire (à coefficients entiers ou rationnels) est dans P

Programmation entière: même problème, mais où la solution doit être entière

Existe-t-il une valeur de x telle que $A.x \leq b$

Problème *NP*-complet

9. $P = NP$?

Similitude avec thèse de Church-Turing

- beaucoup d'efforts pour trouver des algorithmes polynomiaux pour les problèmes NP -complets, mais échec

Fortes présomptions que de tels algorithmes n'existent pas

Fortes présomptions que $P \neq NP$

Fortes présomptions que les problèmes NP -complets soit effectivement intrinsèquement complexes

Une *preuve* de $P \neq NP$ n'est pas à exclure

10. Interpréter la NP-complétude

Que signifie qu'un problème est NP-complet (sous l'hypothèse $P \neq NP$)

- Le problème n'a pas de solution algorithmique polynomiale
- Mais il s'agit de la complexité dans le pire des cas.
Possible que la complexité soit polynomiale pour la plupart des données

Comment résoudre un problème NP-complet ?

- Changer le problème en un problème plus simple. Particulariser le problème pour certaines instances
- Utiliser un algorithme exponentiel si la plupart des instances à résoudre sont de complexité polynomiale
- Utiliser la technique d'exploration, mais en se limitant à un nombre polynomial de cas (heuristique). Algorithme incomplet.
- Si problème d'optimisation, calculer une solution approximative.

10. Interpréter la NP-complétude

Transition de phase

Beaucoup de problèmes NP-complets sont caractérisés par un paramètre de contrôle

Valeurs du paramètre de contrôle induit deux grandes régions :

- Instances où presque tout est solution (réponse est (presque) toujours *oui*)
- Instances où solution quasi inexistante (réponse (presque) toujours *non*)

La transition entre ces deux régions est souvent brusque : une très petite variation du paramètre sépare ces deux régions.

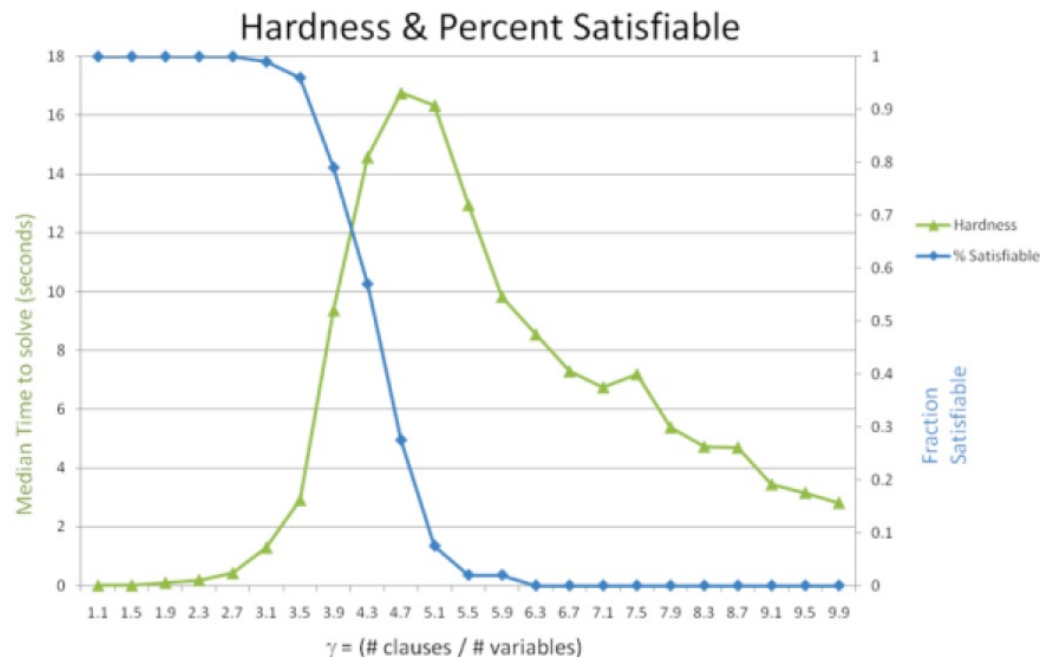
Semblable à la transition de phase en physique.

10. Interpréter la NP-complétude

Transition de phase : SAT

Paramètre de SAT : ratio entre nombre de clauses et nombre de variables

- Ratio faible pour SAT : instances (presque) toujours satisfaisables
- Ratio élevé pour SAT : instances (presque) toujours non satisfaisables



$\frac{\text{nb clause}}{\text{nb variables}}$



<https://arxiv.org/pdf/1304.0145>

10. Interpréter la NP-complétude

Transition de phase

Les instances difficiles d'un problème NP-complet se situent dans la zone de transition de phase

- Pic de difficulté : petite variation du paramètre, passage d'instances faciles à des instances très difficiles
- Pic de difficulté indépendant de l'algorithme choisi
- Information sur transition de phase permet d'adapter des heuristiques de résolution

11. Autres classes de complexité

Classe co-NP

Famille des ensembles *ND-récurifs* dont le **complément** peut être reconnu par un programme Java *non déterministe* de complexité temporelle *polynomiale*

e. $A \in \text{co-NP}$ iff $\bar{A} \in \text{NP}$

Attention:

si $A \in \text{NP}$, on n'a pas nécessairement $\bar{A} \in \text{NP}$

Propriétés

- $P \subseteq \text{co-NP}$
- Si $P = \text{NP}$, alors $P = \text{co-NP}$

Classe EXPTIME

Famille des ensembles *récurifs* pouvant être décidés par un programme Java de complexité temporelle $O(2^p)$, où p est un polynôme.

11. Autres classes de complexité

Classe LOGSPACE

Famille des ensembles *rékursifs* pouvant être décidés par un programme Java de complexité spatiale $O(\log)$

Classe PSPACE

Famille des ensembles *rékursifs* pouvant être décidés par un programme Java de complexité spatiale $O(p)$, où p est un polynôme

Le problème de la satisfaisabilité des formules Booléennes quantifiées est PSPACE-complet wrt \leq_p

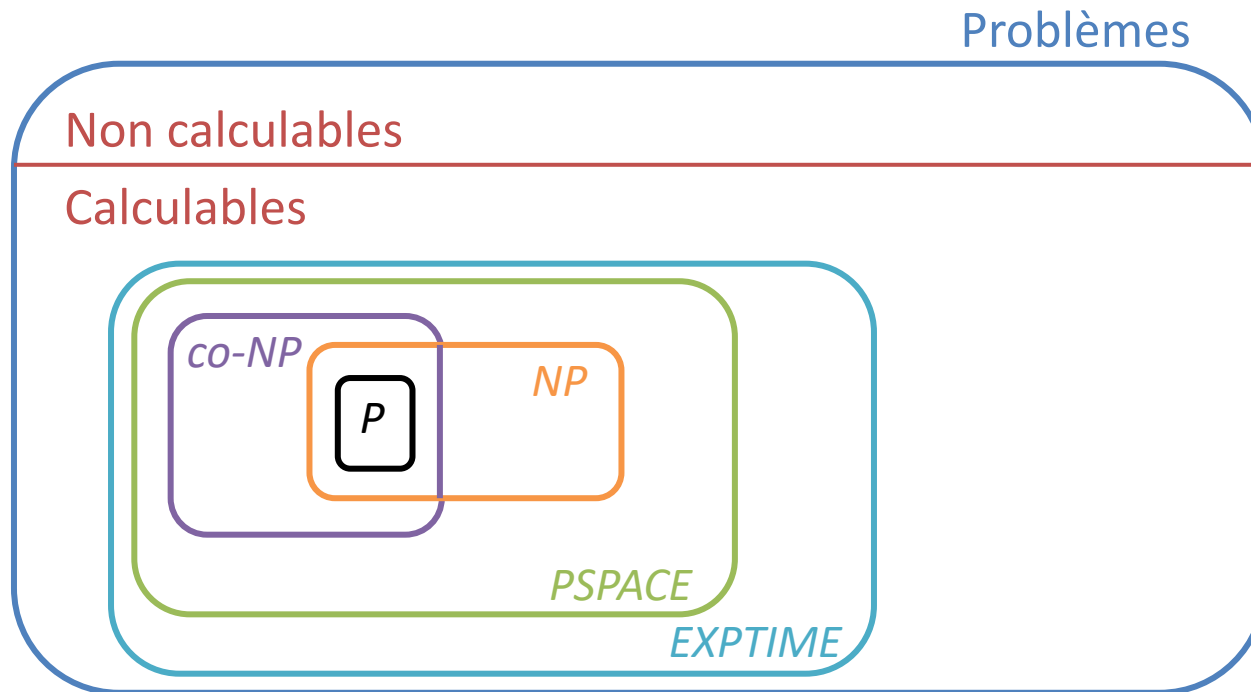
Classe NPSPACE

Famille des ensembles *ND-rékursifs* pouvant être reconnus par un programme Java *non déterministe* de complexité spatiale $O(p)$, où p est un polynôme

Propriété: PSPACE = NPSPACE

11. Autres classes de complexité

Relations entre classes



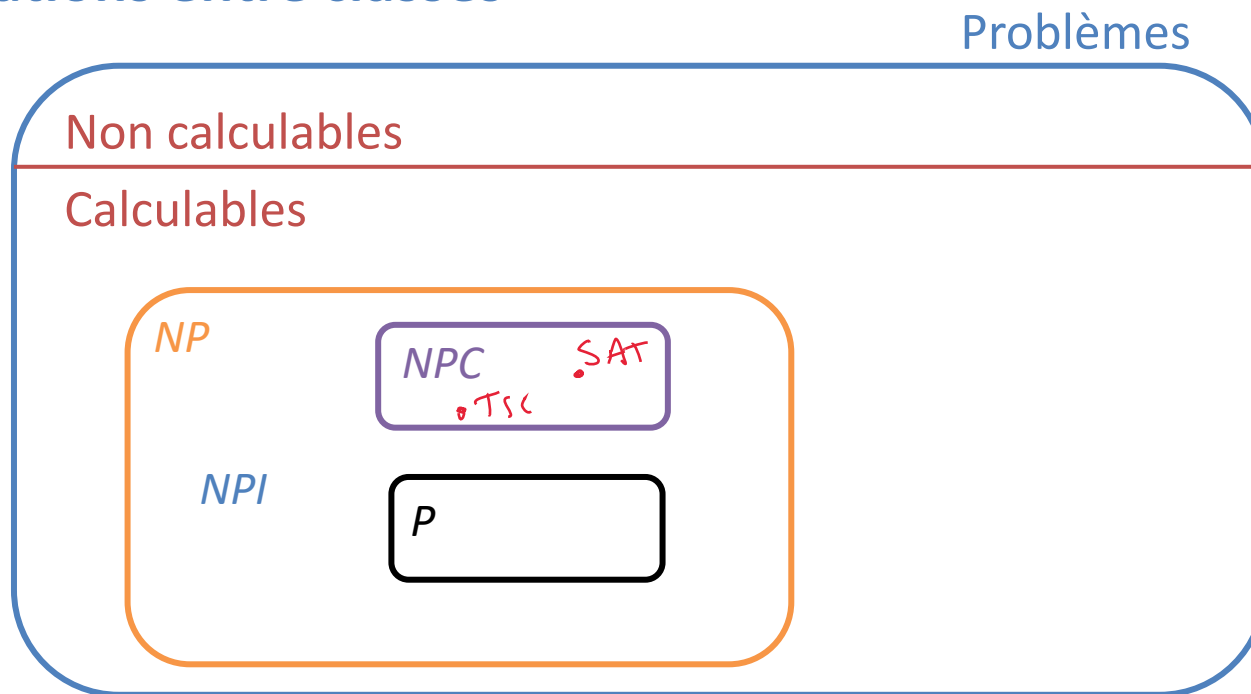
$NP = PSPACE$? problème ouvert

On sait que $P \neq EXPTIME$

Si $P=NP$, alors $EXPTIME=NEXPTIME$

11. Autres classes de complexité

Relations entre classes



NPC : problèmes NP-complets

$NPI = NP \setminus NPC \setminus P$

On peut montrer que NPI est non vide (supposant $P \neq NP$)

11. Autres classes de complexité

The Complexity Zoo

Il existe de *très* nombreuses classes de complexité (443)

<https://perso.ensta-paris.fr/~diam/ro/online/www.complexityzoo.com/index.html>

0-1-NPC - 1NAuxPDAp - #AC0 - #L - #L/poly - #GA - #P - #W[t] - \oplus EXP - \oplus L - \oplus L/poly - \oplus P - \oplus SAC0 - \oplus SAC1 - AOPP - AC - AC0 - AC1 - AC0[m] - ACC0 - AH - AL - ALL - AlgP/poly - Almost-NP - Almost-P - Almost-PSpace - AM - AMEXP - AM intersect coAM - AM[polylog] - AmpMP - AmpP-BQP - AP - APP - APX - AUC-SPACE(f(n)) - AuxPDA - AVBPP - AvE - AvP - AW[P] - AWPP - AW[SAT] - AW[*] - AW[t] - AxP - AxPP - β P - BH - BPE - BPEE - BPHSPACE(f(n)) - BPL - BP•NP - BPP - BPPcc - BPPKT - BPP//log - BPP-OBDD - BPPpath - BPQP - BPSpace(f(n)) - BPTIME(f(n)) - BQNC - BQNP - BQP - BQP/log - BQP/poly - BQP/qlog - BQP/qpoly - BQP-OBDD - BQPSpace - BQPtt/poly - BQTIME(f(n)) - k-BWBP - C=AC0 - C=L - C=P - CFL - CLOG - CH - Check - CkP - CNP - coAM - coC=P - cofrIP - Coh - coMA - coModkP - compIP - compNP - coNE - coNEXP - coNL - coNP - coNPcc - coNP/poly - coNQP - coRE - coRNC - coRP - coSL - coSPARSE - coUCC - coUP - CP - CSIZE(f(n)) - CSL - CZK - D#P - DCFL - Δ 2P - δ -BPP - δ -RP - DET - DiffAC0 - DisNP - DistNP - DP - DQP - DSPACE(f(n)) - DTIME(f(n)) - DTISP(t(n),s(n)) - Dyn-FO - Dyn-ThC0 - E - EE - EEE - EESpace - EEXP - EH - ELEMENTARY - ElkP - EPTAS - k-EQBP - EQP - EQTIME(f(n)) - ESspace - ExistsBPP - ExistsNISZK - EXP - EXP/poly - EXPSPACE - FBQP - Few - FewP - FH - FNL - FNL/poly - FNP - FO(t(n)) - FOLL - FP - FPNP[log] - FPR - FPRAS - FPT - FPTnu - FPTsu - FPTAS - FQMA - frIP - F-TAPE(f(n)) - F-TIME(f(n)) - GA - GAN-SPACE(f(n)) - GapAC0 - GapL - GapP - GC(s(n),C) - GCSL - GI - GLO - GPCD(r(n),q(n)) - G[t] - HeurBPP - HeurBPTIME(f(n)) - HkP - HP - HVSZK - IC(log,poly) - IP - IPP - IP[polylog] - L - LIN - LkP - LOGCFL - LogFew - LogFewNL - LOGNP - LOGSNP - L/poly - LWPP - MA - MA' - MAC0 - MAE - MAEXP - mAL - MaxNP - MaxPB - MaxSNP - MaxSNP0 - mcoNL - MinPB - MIP - MIP*[2,1] - MIPEXP - (Mk)P - mL - mNC1 - mNL - mNP - ModkL - ModkP - ModP - ModZkL - mP - MP - MPC - mP/poly - mTC0 - NAuxPDAp - NC - NC0 - NC1 - NC2 - NE - NE/poly - Nearly-P - NEE - NEEE - NEEXP - NEXP - NEXP/poly - NISZK - NISZKh - NL - NL/poly - NLIN - NLOG - NONE - NP - NPC - NPcc - NPC - NPI - NP intersect coNP - (NP intersect coNP)/poly - NP/log - NPMV - NPMV-sel - NPMVt - NPMVt-sel - NPO - NPOPB - NP/poly - (NP,P-samplable) - NPR - NPSPACE - NPSV - NPSV-sel - NPSVt - NPSVt-sel - NQP - NSpace(f(n)) - NT - NTIME(f(n)) - OCQ - OptP - P - P/log - P/poly - P#P - P#P[1] - PAC0 - PBP - k-PBP - PC - Pcc - PCD(r(n),q(n)) - P-close - PCP(r(n),q(n)) - PermUP - PEXP - PF - PFCHK(t(n)) - PH - PHcc - Φ 2P - PhP - Π 2P - PINC - PIO - PK - PKC - PL - PL1 - PLinfinity - PLF - PLL - PLS - PNP - P | NP - PNP[k] - PNP[log] - PNP[log^2] - P-OBDD - PODN - polyL - PostBQP - PP - PP/poly - PPA - PPAD - PPADS - PPP - PPP - PPSpace - PQUERY - PR - PR - PrHSPACE(f(n)) - PromiseBPP - PromiseBQP - PromiseP - PromiseRP - PrSpace(f(n)) - P-Sel - PSK - PSPACE - PT1 - PTAPE - PTAS - PT/WK(f(n),g(n)) - PZK - Q - QAC0 - QAC0[m] - QACC0 - QACf0 - QAM - QCFL - QCMA - QH - QIP - QIP[2] - QMA - QMA+ - QMA(2) - QMalog - QMAM - QMIP - QMIPl - QMIPne - QNC - QNC0 - QNCf0 - QNC1 - QP - QPLIN - QPSpace - QRG - QS2P - QSZK - R - RE - REG - RevSpace(f(n)) - RG - RG[1] - RHL - RHSPACE(f(n)) - RL - RNC - RP - RPP - RSPACE(f(n)) - S2P - S2-EXP•PNP - SAC - SAC0 - SAC1 - SAPTIME - SBP - SC - SE - SEH - SelfNP - SFk - Σ 2P - SKC - SL - SLICEWISE PSPACE - SNP - SO-E - SP - SP - span-P - SPARSE - SPL - SPP - SQG - SUBEXP - symP - SZK - SZKh - TALLY - TC0 - TFNP - Θ 2P - TreeBQP - TREE-REGULAR - UAP - UCC - UE - UL - UL/poly - UP - US - VNCK - VNPk - VPk - VQPk - W[1] - WAPP - W[P] - WPP - W[SAT] - W[*] - W[t] - W*[t] - XOR-MIP*[2,1] - XP - Xpuniform - YACC - ZPE - ZPP - ZPTIME(f(n))