I Décidabilité

Définition : Algorithme

Un algorithme est une suite d'instructions déterministes (pas d'aléatoire) qui prend une entrée finie.
On suppose que cet algorithme s'exécute sur un ordinateur avec une quantité illimité de mémoire.
Sur une entrée, un algorithme peut :

- renvoyer un résultat en temps fini
- ne pas renvoyer de résultat, soit parce qu'il ne termine pas (boucle infinie...), soit parce qu'il plante (dépassement de tableau...)

Définition : Machine universelle

Une machine universelle est un programme U prenant en entrée le code source d'un programme f et un argument x, et simulant l'exécution de f sur x. Autrement dit :

- si $f(x)$ renvoie un résultat y en temps fini, alors $U(f,x)$ renvoie y en temps fini
- si $f(x)$ déclenche une erreur, $U(f,x)$ déclenche une erreur
- si $f(x)$ ne termine pas, $U(f,x)$ ne termine pas.

Exemple : l'interpréteur OCaml (utop) est une machine universelle pour les programmes OCaml.

Définition : Problème de décision

Un problème de décision est un couple (I, I^+) tel que :

- I est l'ensemble des instances du problème
- $I^+ \subset I$ est l'ensemble des instances positives du problème

On peut aussi définir un problème sous forme d'une question binaire sur une instance.

Exemples de problèmes de décision :

SAT

- Instance : une formule logique φ
- Question : φ est-elle satisfiable ?

$I = \{\varphi \mid \varphi \text{ formule logique}\}$, $I^+ = \{\varphi \mid \varphi \text{ satisfiable}\}$

APPARTIENT

- Instance : un mot w et un automate A
- Question : $w \in L(A)$?

$I = \{(w, A) \mid w \text{ mot et } A \text{ automate}\}$, $I^+ = \{(w, A) \mid w \in L(A)\}$

Définition : Décidabilité

Un problème de décision (I, I^+) est dit décidable s'il existe un algorithme A qui :

- prend une instance $i \in I$ du problème en entrée
- renvoie `true` en temps fini si i est une instance positive ($i \in I^+$)
- renvoie `false` en temps fini si i est une instance négative ($i \notin I^+$)

Sinon, le problème est dit indécidable.

Remarques :

- A doit terminer en temps fini sur toutes les instances.
- Pour montrer qu'un problème est décidable, il suffit d'exhiber un algorithme qui le décide. Montrer qu'il est indécidable est a priori plus difficile : il faut montrer qu'aucun algorithme ne peut le résoudre.
- Quand on s'intéresse à la décidabilité d'un problème, seule l'existence d'un algorithme compte : sa complexité n'a aucune importance. Il n'est donc pas non plus nécessaire de préciser les structures de données utilisées, tant que celles-ci sont calculables.
- Si l'ensemble des instances positives est de cardinal fini, le problème est trivialement décidable : il suffit d'énumérer toutes les instances positives et tester si l'une d'entre elles est égale à l'entrée.

Exercice 1.
Montrer que le problème SAT est décidable.

Définition : Fonction calculable

Une fonction $f : E \to F$ est calculable s'il existe un algorithme A qui, pour tout élément $x \in E$, termine en temps fini et renvoie $f(x)$.

ARRET

- Instance : le code source d'un programme f et un argument x
- Question : f termine-t-il sur l'entrée x ?

En OCaml, cela revient à écrire une fonction `arret : string -> string -> bool` qui termine sur toute entrée et telle que si `f` est une fonction et `x` une chaîne de caractères, `arret f x` renvoie `true` si $f(x)$ termine, `false` sinon.

Théorème

Le problème de l'arrêt est indécidable.

Preuve :

Définition : Réduction

On dit qu'un problème de décision $P_1 = (I_1, I_1^+)$ se réduit à un problème de décision $P_2 = (I_2, I_2^+)$, noté $P_1 \leq P_2$, s'il existe une fonction calculable $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1, \quad i \in I_1^+ \iff f(i) \in I_2^+$$

Exercice 2.
Montrer que ACCESSIBLE $\leq$ CHEMIN.

ACCESSIBLE

- Instance : un graphe $G = (S, A)$ et deux sommets $s, t \in S$
- Question : existe-t-il un chemin de s à t dans G ?

CHEMIN

- Instance : un graphe $G = (S, A)$, $s, t \in S$, $k \in \mathbb{N}$
- Question : existe-t-il un chemin de s à t dans G de longueur au plus k ?

Théorème

Soient P_1 et P_2 deux problèmes de décision tels que $P_1 \leq P_2$. Alors :

- Si P_1 est indécidable, alors P_2 est indécidable.
- Si P_2 est décidable, alors P_1 est décidable.

Preuve :

Méthode pour montrer qu'un problème P_1 est indécidable :

1. Supposer par l'absurde que P_1 est décidable par une fonction f_1.
2. En déduire une fonction f_2 qui décide un problème P_2 connu comme indécidable (par exemple ARRET).
3. Conclure que P_2 est indécidable.

Exercice 3.
Montrer que le problème ARRET-VIDE est indécidable.

ARRET-VIDE

- Instance : une fonction f
- Question : f termine-t-il sur l'entrée vide ?

II Classes de complexité

Définition : Taille d'une instance

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Remarques :

- Un entier n est codé en base 2, donc sa taille est $\log_2(n)$. On pourrait aussi le coder en unaire ce qui donnerait une taille n, mais ce n'est pas « raisonnable ».
- On s'intéresse seulement à l'ordre de grandeur de la taille (O(...)).

Exemples :

- La taille d'un entier n est $\log_2(n)$.
- Un ensemble de p entiers dans $[\![1, n]\!]$ a une taille de $p \log_2(n)$.
- Un graphe à n sommets représenté par une matrice d'adjacence a une taille de n^2 (car il y a n^2 cases contenant chacune un bit).

II.1 P

Définition : Classe P

La classe P est l'ensemble des problèmes de décision qui admettent un algorithme de complexité polynomiale en la taille de l'entrée (c'est-à-dire $O(n^k)$ pour une constante k, où n est la taille de l'entrée).

Exemple :

PGCD

- Instance : entiers a, b, d
- Question : d est-il le PGCD de a et b ?

On peut calculer le PGCD de a et b en utilisant l'algorithme d'Euclide en complexité $O(\log_2(a) + \log_2(b))$, polynomiale en la taille de l'entrée.

Définition : Classe EXP (HP)

La classe EXP est l'ensemble des problèmes de décision qui admettent un algorithme de complexité exponentielle en la taille de l'instance (c'est-à-dire $O(2^{n^k})$ pour une constante k, où n est la taille de l'entrée).

Remarque : P $\subset$ EXP $\subset$ Décidables.

Exemple :

PREMIER

- Instance : un entier n
- Question : n est-il premier ?

On peut énumérer les entiers de 2 à $\sqrt{n}$ pour tester si n est divisible par l'un d'entre eux, en complexité $O(\sqrt{n})$. Ceci est polynomial en n mais exponentielle en la taille $\log_2(n)$ de n (car $\sqrt{n} = 2^{\frac{\log_2(n)}{2}}$), donc cela montre PREMIER $\in EXP$.
Remarque : L'algorithme AKS découvert en 2002 montre que PREMIER $\in$ P.

Exercice 4.

1. Soit k un entier fixé. Montrer que k-CLIQUE $\in$ P.
2. Montrer que CLIQUE $\in$ EXP.

k-CLIQUE

- Instance : un graphe G
- Question : G contient-il une clique de taille k, c'est-à-dire un sous-graphe complet de G de taille k ?

CLIQUE

- Instance : un graphe G et un entier k
- Question : G contient-il une clique de taille k ?

II.2 NP

Définition : Classe NP

Un problème de décision (I, I^+) appartient à la classe NP s'il existe :

- un algorithme de décision A prenant en entrée un couple (x, c) où $x \in I$ et $c \in \{0,1\}^*$
- un polynome Q

tels que :

- A s'exécute en temps polynomial en $|x| + |c|$
- $\forall x \in I,\ x \in I^+ \iff \exists c, |c| \leq Q(|x|), A(x,c) = \texttt{true}$

Remarques :

- c est appelé certificat et peut représenter un entier, un ensemble de valeurs...
- A est appelé vérificateur.
- NP (*Nondeterministic Polynomial*) ne veut pas dire « non polynomial ».

En pratique : 99% du temps, si le problème est de la forme « existe t-il S tel que ... ? », S peut-être choisi comme certificat. Il faut alors justifier que S est de taille polynomiale et qu'on peut vérifier que c'est une solution en complexité polynomiale.

Exercice 5.
Montrer que les problèmes suivants appartiennent à NP :

CLIQUE

- Instance : un graphe G et un entier k
- Question : G contient-il une clique de taille k ?

SAT

- Instance : une formule logique φ en forme normale conjonctive
- Question : φ est-elle satisfiable ?

co-FACTORISATION

- Instance : deux entiers n et p
- Question : n ne possède t-il aucun diviseur d tel que $1 < d < p$?

(En admettant que PREMIER $\in$ P)

Théorème

P $\subset$ NP.

Preuve :

Remarque : La question « P = NP ? » est un des problèmes ouverts les plus célèbres en informatique.

Exercice 6.
Montrer que NP $\subset$ EXP.

II.3 Réduction polynomiale

Définition : Réduction polynomiale

On dit qu'un problème de décision $P_1 = (I_1, I_1^+)$ se réduit polynomialement à un problème de décision $P_2 = (I_2, I_2^+)$, noté $P_1 \leq_p P_2$, s'il existe une fonction calculable en complexité polynomiale $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1, \quad i \in P_1 \iff f(i) \in P_2$$

Intuitivement : P_1 se réduit polynomialement à P_2 si un algorithme pour résoudre P_2 permet de résoudre P_1 en complexité polynomiale. Pour cela, il faut pouvoir transformer une instance de P_1 en une instance de P_2 en complexité polynomiale.

Exercice 7.
Montrer que STABLE $\leq_p$ CLIQUE.

STABLE

- Instance : un graphe G et un entier k
- Question : G contient-il un ensemble stable de taille k, c'est-à-dire un ensemble de k sommets deux à deux non adjacents ?

Exercice 8.
Montrer que $\leq_p$ est réflexive et transitive.

Théorème

Soient P_1 et P_2 deux problèmes de décision tels que $P_1 \leq_p P_2$.

- Si $P_2 \in$ P alors $P_1 \in$ P.
- Si $P_2 \in$ NP alors $P_1 \in$ NP.

Preuve :

II.4 NP-complétude

Définition : Réduction polynomiale

Soit P_1 un problème de décision.

- P_1 est dit NP-difficile si $\forall P_2 \in \text{NP}$, $P_2 \leq_p P_1$.
- P_1 est dit NP-complet si P_1 est NP-difficile et $P_1 \in \text{NP}$.

Intuitivement : un problème est NP-difficile s'il est au moins aussi difficile à résoudre que tous les problèmes de NP.

Théorème

Supposons $P_1 \leq_p P_2$. Si P_1 est NP-difficile alors P_2 est NP-difficile.

Preuve :

Exercice 9.
Soit P_1 un problème NP-difficile. Montrer que si $P_1 \in \text{P}$ alors $\text{P} = \text{NP}$.

Remarque : Il est conjecturé que $\text{P} \neq \text{NP}$, donc qu'il est impossible de résoudre un problème NP-complet en temps polynomial.

Théorème : Cook-Levin (admis)

SAT est NP-complet.

SAT

- Instance : une formule logique φ
- Question : φ est-elle satisfiable ?

Méthode pour montrer qu'un problème P_1 est NP-complet :

1. Montrer que $P_1 \in \text{NP}$.
2. Montrer que $P_2 \leq_p P_1$, où P_2 est un problème NP-complet connu (par exemple SAT).

Théorème : 3-SAT est NP-complet (HP)

3-SAT est NP-complet.

k-SAT

- Instance : une formule logique φ en forme normale conjonctive (FNC) avec au plus k littéraux par clause.
- Question : φ est-elle satisfiable ?

Preuve :

Remarques :

- On en déduit que k-SAT est NP-complet pour $k \geq 3$, car 3-SAT $\leq_p$ k-SAT.
- Par contre 1-SAT et 2-SAT sont dans P (voir X-ENS 2016 MP)

Exercice 10.
Transformer $\varphi = (x_1 \wedge x_2) \vee \neg x_3$ en formule 3-SAT comme dans la preuve précédente (transformation de Tseytin).

Exercice 11.
Notons P_1 = CYCLE-HAMILTONIEN et P_2 = CHEMIN-HAMILTONIEN.

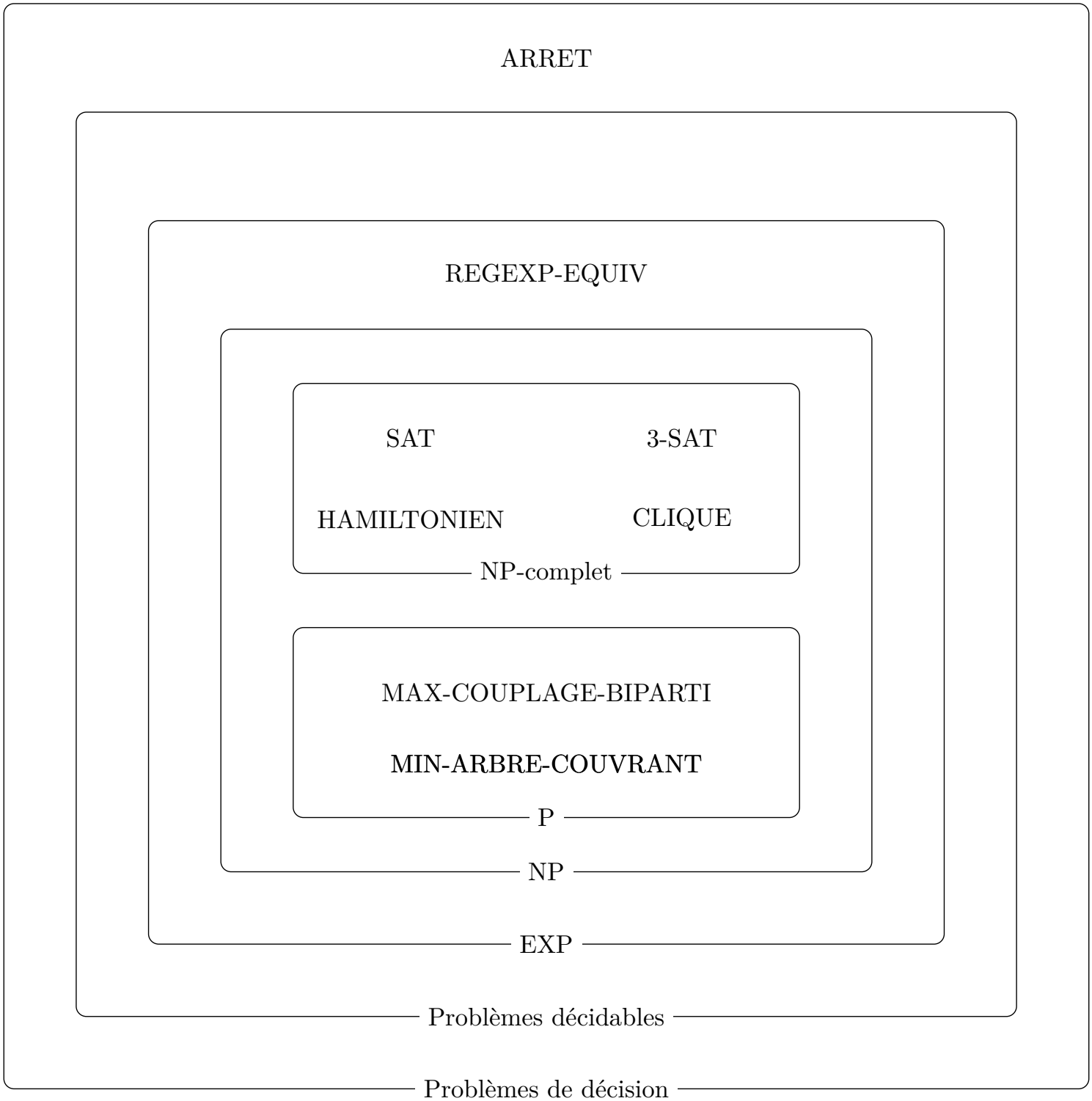
1. Montrer que P_1 est NP-complet, en trouvant une réduction de 3-SAT à P_1.
2. Montrer que P_2 est NP-complet.

CYCLE-HAMILTONIEN

• Instance : un graphe orienté G
• Question : G contient-il un cycle hamiltonien, c'est-à-dire un cycle passant une et une seule fois par chaque sommet ?

CHEMIN-HAMILTONIEN

• Instance : un graphe G orienté
• Question : G contient-il un chemin hamiltonien, c'est-à-dire un chemin passant une et une seule fois par chaque sommet ?


ARRET
REGEXP-EQUIV
SAT
3-SAT
HAMILTONIEN
CLIQUE
NP-complet
MAX-COUPLAGE-BIPARTI
MIN-ARBRE-COUVRANT
P
NP
EXP
Problèmes décidables
Problèmes de décision