Décidabilité et classes de complexité

Quentin Fortier

February 16, 2026

Décidabilité

Définition

Un algorithme est une suite d'instructions déterministes (sans aléatoire) avec une entrée finie.
On suppose que cet algorithme s'exécute sur un ordinateur avec une quantité illimitée 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...).

Décidabilité

Définition

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.

Décidabilité

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

Attention : `let universel f x = f x` n'est pas une machine universelle, car ce n'est pas le code source de `f` qui est donné en argument mais une fonction. Cependant, par simplicité de notation, on se permettra parfois d'écrire `f x` au lieu de `universel f x`.

Décidabilité

Définition

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

- I est l'ensemble des instances (entrées) du problème
- $I^+ \subset I$ est l'ensemble des instances positives du problème (celles pour lesquelles la réponse est « oui »)

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

Décidabilité

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 : $w \in L(A)$?

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

Décidabilité

Définition

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

Décidabilité

Remarques :

- L'algorithme 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.
- Seule l'existence d'un algorithme importe pour montrer la décidabilité : 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 I^+ est fini, le problème est trivialement décidable : il suffit d'énumérer les instances positives et tester si l'une d'entre elles est égale à l'entrée.

Décidabilité

Exercice

Montrer que le problème SAT est décidable.

Décidabilité

ARRET

- Instance : une fonction f et un argument x.
- Question : f termine-t-elle sur l'entrée x ?

Décidabilité

ARRET

- Instance : une fonction f et un argument x.
- Question : f termine-t-elle sur l'entrée x ?

En OCaml, cela revient à écrire une fonction `arret : ('a -> 'b) -> 'a -> bool` qui termine sur toute entrée et telle que `arret f x` renvoie `true` si `f x` termine, `false` sinon.

Décidabilité

ARRET

- Instance : une fonction f et un argument x.
- Question : f termine-t-elle sur l'entrée x ?

En OCaml, cela revient à écrire une fonction `arret : ('a -> 'b) -> 'a -> bool` qui termine sur toute entrée et telle que `arret f x` renvoie `true` si `f x` termine, `false` sinon.

Théorème

Le problème de l'arrêt est indécidable.

Décidabilité : Réduction

Définition

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 $f(x)$.

Décidabilité : Réduction

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq \Pi_2$, s'il existe une fonction calculable $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \Longleftrightarrow f(i) \in I_2^+$$

Décidabilité : Réduction

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq \Pi_2$, s'il existe une fonction calculable $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \Longleftrightarrow f(i) \in I_2^+$$

Intuitivement : $\Pi_1 \leq_p \Pi_2$ signifie qu'on peut transformer une instance de Π_1 en une instance de Π_2 en complexité polynomiale, en préservant la réponse (oui/non). Un algorithme polynomial pour Π_2 permet alors de résoudre Π_1 en complexité polynomiale.

Décidabilité : Réduction

Exercice

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 ?

Décidabilité : Réduction

Théorème

Soient Π_1 et Π_2 deux problèmes de décision tels que $\Pi_1 \leq \Pi_2$. Alors :

- Si Π_1 est indécidable, alors Π_2 est indécidable.
- Si Π_2 est décidable, alors Π_1 est décidable.

Décidabilité : Réduction

Théorème

Soient Π_1 et Π_2 deux problèmes de décision tels que $\Pi_1 \leq \Pi_2$. Alors :

- Si Π_1 est indécidable, alors Π_2 est indécidable.
- Si Π_2 est décidable, alors Π_1 est décidable.

Pour montrer qu'un problème est indécidable, on peut donc trouver une réduction depuis un autre problème connu comme indécidable (par exemple `ARRET`).

Décidabilité : Réduction

Exercice

Montrer que le problème `ARRET-VIDE` est indécidable.

`ARRET-VIDE`

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

Classes de complexité

Définition

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

Classes de complexité

Définition

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(...)).

Classes de complexité

Définition

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 :

- Un ensemble de p entiers dans $[\![1, n]\!]$ est de taille _____

Pour un graphe à n sommets et p arêtes :

- Sa représentation par liste d'adjacence est de taille _____
- Sa représentation par matrice d'adjacence est de taille _____

Classes de complexité : P

Définition

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).

Classes de complexité : P

Définition

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).

Définition (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.

Classes de complexité : P

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.

Classes de complexité : P

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.

Classes de complexité : P

Exercice

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 ?

Classes de complexité : NP

Définition

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

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

tels que :

- A s'exécute en temps polynomial en $|x| + |c|$
- $\forall x \in I$:

$$x \in I^+ \iff \exists c, |c| \leq Q(|x|), A(x,c) = \texttt{true}$$

Autrement dit : Π appartient à NP si, pour chaque instance positive x de Π, il existe un certificat c de taille polynomiale en $|x|$ qui permet de vérifier en temps polynomial en $|x| + |c|$ que x est une instance positive de Π.

Classes de complexité : NP

Remarques :

- c est appelé certificat et peut représenter un entier, un ensemble de valeurs, un graphe... On peut le voir comme une preuve concise que x est une instance positive.
- P est l'ensemble des problèmes résolubles en temps polynomial alors que NP est l'ensemble des problèmes vérifiables en temps polynomial.

Classes de complexité : NP

Remarques :

- c est appelé certificat et peut représenter un entier, un ensemble de valeurs, un graphe... On peut le voir comme une preuve concise que x est une instance positive.
- P est l'ensemble des problèmes résolubles en temps polynomial alors que NP est l'ensemble des problèmes vérifiables en temps polynomial.

En pratique : très souvent, 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.

Classes de complexité : NP

Exercice

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 $1 < d < p$?

Classes de complexité : NP

Théorème

$P \subset NP$.

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

Classes de complexité : NP

Théorème

$P \subset NP$.

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

Exercice

Montrer que $NP \subset EXP$.

Classes de complexité : Réduction polynomiale

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit polynomialement à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq_p \Pi_2$, s'il existe une fonction calculable en complexité polynomiale $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \Longleftrightarrow f(i) \in I_2^+$$

Classes de complexité : Réduction polynomiale

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit polynomialement à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq_p \Pi_2$, s'il existe une fonction calculable en complexité polynomiale $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \Longleftrightarrow f(i) \in I_2^+$$

Intuitivement : $\Pi_1 \leq_p \Pi_2$ signifie qu'on peut transformer une instance de Π_1 en une instance de Π_2 en complexité polynomiale, en préservant la réponse (oui/non).

Classes de complexité : Réduction polynomiale

Exercice

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 ?

CLIQUE

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

Classes de complexité : Réduction polynomiale

Exercice

Montrer que $\leq_p$ est réflexive et transitive. Est-elle antisymétrique ?

Classes de complexité : Réduction polynomiale

Théorème

Soient Π_1 et Π_2 deux problèmes de décision tels que $\Pi_1 \leq_p \Pi_2$.

- Si $\Pi_2 \in$ P alors $\Pi_1 \in$ P.
- Si $\Pi_2 \in$ NP alors $\Pi_1 \in$ NP.

Classes de complexité : NP-complétude

Définition

Soit Π un problème de décision.

- Π est NP-difficile si : $\forall \Pi' \in$ NP, $\Pi' \leq_p \Pi$.
- Π est NP-complet si Π est NP-difficile et $\Pi \in$ NP.

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

Classes de complexité : NP-complétude

Théorème

Supposons $\Pi_1 \leq_p \Pi_2$.
Si Π_1 est NP-difficile alors Π_2 est NP-difficile.

Classes de complexité : NP-complétude

Exercice

Montrer que le problème suivant est NP difficile :

TAUTOLOGIE

- Instance : une formule logique φ.
- Question : φ est-elle une tautologie ?

Remarque : Il est conjecturé que TAUTOLOGIE n'est pas dans NP.

Classes de complexité : NP-complétude

Exercice

Soit Π un problème NP-difficile.
Montrer que si $\Pi \in \mathrm{P}$ alors $\mathrm{P} = \mathrm{NP}$.

Classes de complexité : NP-complétude

Exercice

Soit Π un problème NP-difficile.
Montrer que si $\Pi \in$ P alors P = NP.

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

Classes de complexité : NP-complétude

Théorème de Cook-Levin (admis)

`SAT` est NP-complet.

Classes de complexité : NP-complétude

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

1. Montrer que $\Pi \in$ NP.
2. Montrer que $\Pi' \leq_p \Pi$, où Π' est un problème NP-complet connu (par exemple `SAT`).

Classes de complexité : NP-complétude

Théorème (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 ?

Classes de complexité : NP-complétude

Théorème (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 ?

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)

Classes de complexité : NP-complétude

Exercice

Transformer $\varphi = (x \wedge y) \vee \neg z$ en formule 3-SAT comme dans la preuve précédente (transformation de Tseytin).

Classes de complexité : NP-complétude

Exercice

1. Montrer que STABLE $\in$ NP.
2. Pour $\varphi = \bigwedge_{k=1}^{p} C_k$ une instance de 3-SAT, on définit $G_\varphi = (S, A)$ où
   • S contient un sommet par littéral, autant de fois qu'il apparaît dans φ.
   • A contient une arête entre deux sommets s'ils sont dans la même clause ou s'ils sont la négation l'un de l'autre.
   Dessiner G_φ si $\varphi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y})$.
3. Montrer que si G_φ contient une clique de taille p alors φ est satisfiable.
4. Montrer que si φ est satisfiable alors G_φ contient une clique de taille p. Conclure.
5. Montrer que CLIQUE est NP-complet.

Classes de complexité : NP-complétude

Exercice

Donner un exemple de problème NP-difficile qui n'est pas dans NP, en le démontrant.