

## I Décidabilité

### Définition : Algorithme

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...).

Un algorithme peut être écrit en pseudo-code ou dans un langage de programmation (OCaml, C...).

### 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.

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 notera `f x` au lieu de `universel f x`.

### Définition : Problème de décision

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.

Exemples de problèmes de décision :

#### SAT

- Instance : une formule logique  $\varphi$ .
- Question :  $\varphi$  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 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 :

- 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.

## Exercice 1.

Montrer que le problème SAT est décidable.

Le problème de l'arrêt est célèbre en informatique :

## 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.

Preuve :

## Définition : Fonction calculable

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

## Définition : Réduction ❤

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  $f : I_1 \rightarrow I_2$  telle que :

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

Pour réduire  $\Pi_1$  à  $\Pi_2$ , il faut donc pouvoir transformer une instance de  $\Pi_1$  en une instance de  $\Pi_2$ , en préservant la réponse (oui/non).

## Exercice 2.

Montrer que ACCESSIBLE < 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  $\Pi_1$  et  $\Pi_2$  deux problèmes de décision tels que  $\Pi_1 \leq \Pi_2$ . Alors :

- Si  $\Pi_1$  est indécidable alors  $\Pi_2$  est indécidable.
- Si  $\Pi_2$  est décidable alors  $\Pi_1$  est décidable.

Preuve :

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

Intuitivement,  $\Pi_1 \leq \Pi_2$  signifie que  $\Pi_2$  est au moins aussi difficile à résoudre que  $\Pi_1$  (un algorithme pour  $\Pi_2$  permettrait de résoudre  $\Pi_1$ ).

**Exercice 3.**

Montrer que le problème ARRET-VIDE est indécidable.

**ARRET-VIDE**

- Instance : une fonction  $f$ .
- Question :  $f$  termine-t-elle 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 ( $O(\dots)$ ).

Exemples :

- Un ensemble de  $p$  entiers dans  $\llbracket 1, n \rrbracket$  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 \_\_\_\_\_

## 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 :  $P \subset EXP \subset \text{Décidables}$ .

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  $\text{PREMIER} \in EXP$ .

Remarque : L'algorithme AKS découvert en 2002 montre que  $\text{PREMIER} \in P$ .

### Exercice 4.

1. Soit  $k$  un entier fixé. Montrer que  $k\text{-CLIQUE} \in P$ .

2. Montrer que  $\text{CLIQUE} \in EXP$ .

#### $k\text{-CLIQUE}$

- 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  $A$  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  $|x| + |c|$
- $\forall x \in I$  :

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

Remarques :

- $c$  est appelé certificat et peut représenter un entier, un ensemble de valeurs, un graphe... On peut le voir comme une preuve concise (de taille polynomiale) que  $x$  est une instance positive.
- $A$  est appelé vérificateur et peut utiliser le certificat  $c$  pour vérifier que  $x$  est une instance positive :  $x \in I^+$  si et seulement s'il existe un certificat  $c$  qui permet à  $A$  de renvoyer « Oui » en temps polynomial.
- NP (*Nondeterministic Polynomial*) ne veut pas dire « non polynomial » mais reconnaissable en temps polynomial par une machine de Turing non déterministe (HP) : un automate non déterministe muni d'une mémoire qui, à chaque étape, peut faire plusieurs choix et explorer tous les chemins possibles en parallèle. Si au moins un chemin mène à une réponse « oui » en temps polynomial, la machine renvoie « oui ».
- 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.

### 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  $\varphi$  en forme normale conjonctive.
- Question :  $\varphi$  est-elle satisfiable ?

#### co-FACTORISATION

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

**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  $\Pi_1 = (I_1, I_1^+)$  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  $f : I_1 \rightarrow I_2$  telle que :

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

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

**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. Est-elle antisymétrique ?

---

---

---

### Théorème

Soient  $\Pi_1$  et  $\Pi_2$  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  $\Pi_1 \in NP$ .

Preuve :

---

---

---

---

## II.4 NP-complétude

### Définition : NP-difficile, NP-complet ♡

Soit  $\Pi$  un problème de décision.

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

### Théorème : ♡

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

Preuve :

---

---

---

---

### Exercice 9.

Soit  $\Pi$  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.

### Théorème : Cook-Levin (admis) ♡

SAT est NP-complet.

♡ Méthode pour montrer qu'un problème  $\Pi$  est NP-complet :

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

Rappels :

- Un littéral est une variable propositionnelle ou sa négation.
- Une clause est une disjonction de littéraux. Par exemple,  $x_1 \vee \neg x_2 \vee x_1$  est une clause contenant 3 littéraux.
- Une formule en forme normale conjonctive (FNC) est une conjonction de disjonctions de littéraux. Par exemple,  $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$  est une formule en forme normale conjonctive.

On rappelle le problème  $k$ -SAT :

**$k$ -SAT**

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

**Théorème : 3-SAT est NP-complet (HP)**

3-SAT est NP-complet.

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.**

On considère les problèmes suivants :

**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  $k$  ?

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  $\varphi$ .
  - $A$  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  $G_\varphi$  si  $\varphi = (\bar{x} \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$ .

3. Montrer que si  $G_\varphi$  contient une clique de taille  $p$  alors  $\varphi$  est satisfiable.

4. Montrer que si  $\varphi$  est satisfiable alors  $G_\varphi$  contient une clique de taille  $p$ . Conclure.

5. Montrer que CLIQUE est NP-complet.

## Définition : Co-classes (HP)

- Si  $\Pi = (I, I^+)$  est un problème de décision, on définit son complémentaire  $\bar{\Pi} = (I, I^-)$  où  $I^- = I \setminus I^+$ .
  - On définit co-P et co-NP comme les classes des problèmes dont le complémentaire est dans P et NP respectivement.
  - On dit qu'un problème  $\Pi$  est co-NP complet si  $\Pi \in \text{co-NP}$  et  $\forall \Pi' \in \text{co-NP}, \Pi' \leq_p \Pi$ .

## Exercice 12.

- Montrer que  $\Pi$  est co-NP complet si et seulement si  $\bar{\Pi}$  est NP-complet.

2. Montrer que le problème suivant est co-NP complet :

TAUTOLOGIE

- Instance : une formule logique  $\varphi$ .
  - Question :  $\varphi$  est-elle une tautologie ?

---

---

---

---

