Théorie de la complexité

Présentation et Motivations

Thomas Place, Marc Zeitoun

Master 1 CSI

Septembre 2023

Contenu du cours d'aujourd'hui

1. Objectifs de l'UE.
2. Modalités de l'UE.
3. Plan du cours.
4. Condensé du cours et vocabulaire important.

Objectifs de l'UE

À l'issue du cours ce semestre, vous devrez savoir :

- Ce qu'est un **problème** et un **algorithme** **en informatique**.

Objectifs de l'UE

À l'issue du cours ce semestre, vous devrez savoir :

- Ce qu'est un **problème** et un **algorithme** **en informatique**.
- Classifier les problèmes selon leur difficulté.

Objectifs de l'UE

À l'issue du cours ce semestre, vous devrez savoir :

- Ce qu'est un **problème** et un **algorithme** **en informatique**.
- Classifier les problèmes selon leur difficulté.
  - Prouver qu'un problème n'est pas résoluble par algorithme.

Objectifs de l'UE

À l'issue du cours ce semestre, vous devrez savoir :

- Ce qu'est un **problème** et un **algorithme** **en informatique**.
- Classifier les problèmes selon leur difficulté.
  - Prouver qu'un problème n'est pas résoluble par algorithme.
  - Minorer la complexité du meilleur algorithme résolvant un problème.

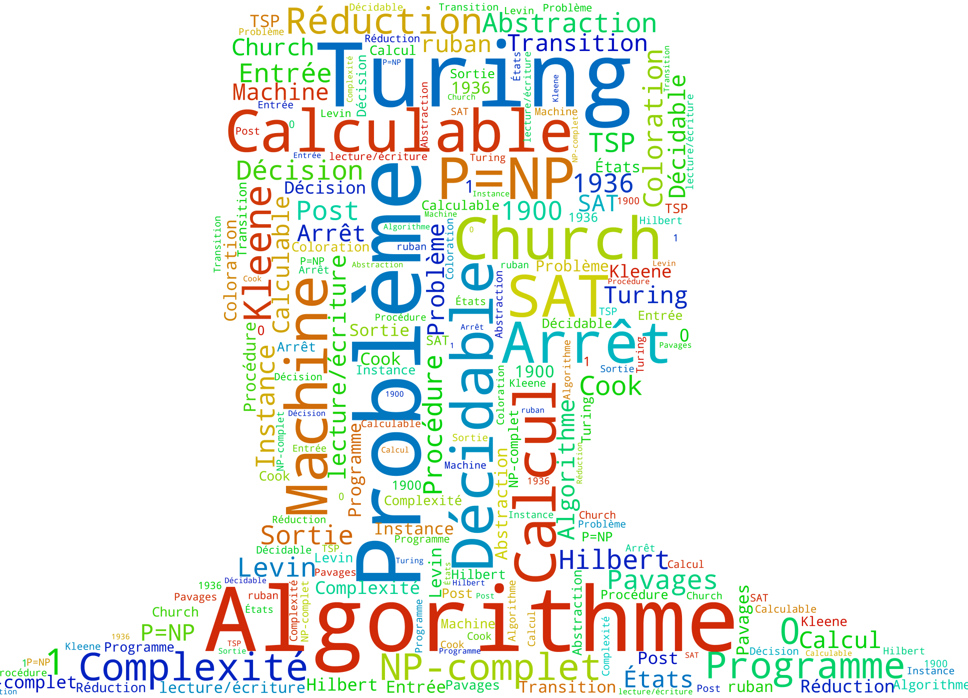
Objectifs de l'UE

À l'issue du cours ce semestre, vous devrez savoir :

- Ce qu'est un **problème** et un **algorithme** **en informatique**.
- Classifier les problèmes selon leur difficulté.
  - Prouver qu'un problème n'est pas résoluble par algorithme.
  - Minorer la complexité du meilleur algorithme résolvant un problème.

Quelques mots-clés

Problème, machine, algorithme, indécidabilité, NP-complétude, réduction.


Turing
Calculable
Problème
Algorithme
Décidable
Machine
Calcul
Church
SAT
P=NP
Arrêt
Réduction
Abstraction
Transition
Complexité
NP-complet
Programme
Hilbert
Kleene
Post
Cook
Levin
Instance
Sortie
Entrée
Décision
Procédure
Coloration
lecture/écriture
ruban
États
Pavages
TSP
1936
1900
0
1

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.
  - DM.

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.
  - DM.
  - Test « ensembles et raisonnement » 18/9 (noté mais pas comptabilisé),

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.
  - DM.
  - Test « ensembles et raisonnement » 18/9 (noté mais pas comptabilisé),
  - Test « ensembles et raisonnement » 25/9 (noté et comptabilisé).

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.
  - DM.
  - Test « ensembles et raisonnement » 18/9 (noté mais pas comptabilisé),
  - Test « ensembles et raisonnement » 25/9 (noté et comptabilisé).

    Prérequis à réviser en auto-formation.

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.
  - DM.
  - Test « ensembles et raisonnement » 18/9 (noté mais pas comptabilisé),
  - Test « ensembles et raisonnement » 25/9 (noté et comptabilisé).
    Prérequis à réviser en auto-formation.
  - Chaque TD :
    - Test éventuel en fin de TD sur la séance du jour.

Modalités de l'UE

- Examen 3h (en session 1 et 2).
- Contrôle continu :
  - DS 3h, date précisée ultérieurement.
  - DM.
  - Test « ensembles et raisonnement » 18/9 (noté mais pas comptabilisé),
  - Test « ensembles et raisonnement » 25/9 (noté et comptabilisé).
    Prérequis à réviser en auto-formation.
  - Chaque TD :
    - Test éventuel en fin de TD sur la séance du jour.
- Calcul de note identique en session 1 et 2 :

  50% Examen + 50% CC.

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.
- Les termes employés ne sont **pas interchangeables**, soyez rigoureux !

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.
- Les termes employés ne sont **pas interchangeables**, soyez rigoureux !
- La bonne compréhension du **vocabulaire** est une première étape clé.

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.
- Les termes employés ne sont **pas interchangeables**, soyez rigoureux !
- La bonne compréhension du **vocabulaire** est une première étape clé.

Quelques points pour réussir :

- Être actif/active en cours, ne laisser passer aucune ambiguïté.

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.
- Les termes employés ne sont **pas interchangeables**, soyez rigoureux !
- La bonne compréhension du **vocabulaire** est une première étape clé.

Quelques points pour réussir :

- Être actif/active en cours, ne laisser passer aucune ambiguïté.
- Lire, comprendre et mémoriser le cours, en nous sollicitant si besoin.

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.
- Les termes employés ne sont **pas interchangeables**, soyez rigoureux !
- La bonne compréhension du **vocabulaire** est une première étape clé.

Quelques points pour réussir :

- Être actif/active en cours, ne laisser passer aucune ambiguïté.
- Lire, comprendre et mémoriser le cours, en nous sollicitant si besoin.
- Faire soi-même les exercices à l'avance, et se tester sur les annales.

  *Corrections des exercices de TD détaillées **en cours**.*

Conseils de travail

Attention !

- Vous aurez à comprendre et prouver des propriétés.
- Les termes employés ne sont **pas interchangeables**, soyez rigoureux !
- La bonne compréhension du **vocabulaire** est une première étape clé.

Quelques points pour réussir :

- Être actif/active en cours, ne laisser passer aucune ambiguïté.
- Lire, comprendre et mémoriser le cours, en nous sollicitant si besoin.
- Faire soi-même les exercices à l'avance, et se tester sur les annales.

  *Corrections des exercices de TD détaillées **en cours**.*
- Ne pas essayer d'imiter des phrases du cours sans les comprendre.

Prérequis

- Savoir prendre des notes ($\neq$ copier le tableau).

Prérequis

- Savoir prendre des notes ($\neq$ copier le tableau).
- Pouvoir comprendre sans ambiguïté des définitions d'objets nouveaux.

Prérequis

- Savoir prendre des notes ($\neq$ copier le tableau).
- Pouvoir comprendre sans ambiguïté des définitions d'objets nouveaux.
- Savoir évaluer la complexité d'algorithmes simples.

Prérequis

- Savoir prendre des notes ($\neq$ copier le tableau).
- Pouvoir comprendre sans ambiguïté des définitions d'objets nouveaux.
- Savoir évaluer la complexité d'algorithmes simples.
- Être à l'aise en calcul (ordres de grandeur, fonctions log et exp, ...).

Prérequis

- Savoir prendre des notes ($\neq$ copier le tableau).
- Pouvoir comprendre sans ambiguïté des définitions d'objets nouveaux.
- Savoir évaluer la complexité d'algorithmes simples.
- Être à l'aise en calcul (ordres de grandeur, fonctions log et exp, ...).
- Connaître les bases de la **logique** et du **raisonnement** mathématique.
  - Théorie naïve des ensembles.
  - Langage logique (connecteurs, quantificateurs).
  - Lien entre logique et ensembles.
  - Méthodes de preuves (directes, par contraposée, par l'absurde, ...).

Prérequis

- Savoir prendre des notes ($\neq$ copier le tableau).
- Pouvoir comprendre sans ambiguïté des définitions d'objets nouveaux.
- Savoir évaluer la complexité d'algorithmes simples.
- Être à l'aise en calcul (ordres de grandeur, fonctions log et exp, ...).
- Connaître les bases de la **logique** et du **raisonnement** mathématique.
  - Théorie naïve des ensembles.
  - Langage logique (connecteurs, quantificateurs).
  - Lien entre logique et ensembles.
  - Méthodes de preuves (directes, par contraposée, par l'absurde, ...).
- *Connaissance en théorie des automates : peut aider mais non requise.*

Ressources

Disponibles sur la page web de l'UE :

`https://www.labri.fr/perso/zeitoun/enseignement/23-24/CSI/pmwiki.php`

- Diapos,
- Ressources bibliographiques,
- Annales,
- Notes de cours et exercices, **à faire vous-mêmes à l'avance**,
- ...

Plan du cours

1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

3. Réductions et NP-complétude

4. Complexité des problèmes – $P \stackrel{?}{=} NP$

Aujourd'hui : présentation haut niveau

Partie 1 Problèmes et algorithmes.

- Qu'est-ce qu'un **problème** en informatique ?
- Problèmes faciles et problèmes difficiles.
- Présentation historique.

Aujourd'hui : présentation haut niveau

Partie 1 Problèmes et algorithmes.

- Qu'est-ce qu'un **problème** en informatique ?
- Problèmes faciles et problèmes difficiles.
- Présentation historique.

Partie 2 Formalisation des notions intuitives algorithme, programme.

- Indécidabilité, problème de l'arrêt.
- Machines de Turing.
- Thèse de Church.

Aujourd'hui : présentation haut niveau

Partie 1 Problèmes et algorithmes.

- Qu'est-ce qu'un **problème** en informatique ?
- Problèmes faciles et problèmes difficiles.
- Présentation historique.

Partie 2 Formalisation des notions intuitives algorithme, programme.

- Indécidabilité, problème de l'arrêt.
- Machines de Turing.
- Thèse de Church.

Partie 3 Comment obtenir des résultats d'impossibilité ?

- Technique des réductions.
- Exemples de problèmes indécidables.

Aujourd'hui : présentation haut niveau

Partie 1 Problèmes et algorithmes.

- Qu'est-ce qu'un **problème** en informatique ?
- Problèmes faciles et problèmes difficiles.
- Présentation historique.

Partie 2 Formalisation des notions intuitives algorithme, programme.

- Indécidabilité, problème de l'arrêt.
- Machines de Turing.
- Thèse de Church.

Partie 3 Comment obtenir des résultats d'impossibilité ?

- Technique des réductions.
- Exemples de problèmes indécidables.

Partie 4 Introduction à la complexité des problèmes.

- Question $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$.
- Exemples de problèmes NP-complets.
- Problèmes encore plus difficiles.

Objectifs d'aujourd'hui

Introduction non technique (notions formalisées plus tard).

Objectifs d'aujourd'hui

Introduction non technique (notions formalisées plus tard).

À comprendre sans ambiguïté quand même !

Objectifs d'aujourd'hui

Introduction non technique (notions formalisées plus tard).

À comprendre sans ambiguïté quand même !

1. Introduire le vocabulaire important.

Objectifs d'aujourd'hui

Introduction non technique (notions formalisées plus tard).

À comprendre sans ambiguïté quand même !

1. Introduire le vocabulaire important.

2. Présenter quelques limites du calcul automatique.

Objectifs d'aujourd'hui

Introduction non technique (notions formalisées plus tard).

À comprendre sans ambiguïté quand même !

1. Introduire le vocabulaire important.

2. Présenter quelques limites du calcul automatique.

3. Donner un aperçu de problèmes informatiques difficiles :
   - qu'on ne peut pas résoudre par algorithme,
   - demandant trop de ressources pour une résolution exacte.

Objectifs d'aujourd'hui

Introduction non technique (notions formalisées plus tard).

À comprendre sans ambiguïté quand même !

1. Introduire le vocabulaire important.

2. Présenter quelques limites du calcul automatique.

3. Donner un aperçu de problèmes informatiques difficiles :
   - qu'on ne peut pas résoudre par algorithme,
   - demandant trop de ressources pour une résolution exacte.

4. Donner quelques jalons historiques.

Plan

1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

3. Réductions et NP-complétude

4. Complexité des problèmes – $P \stackrel{?}{=} NP$

Partie 1 : Rappel des objectifs

1. Introduire les notions de **problème informatique** et d'**algorithme**.

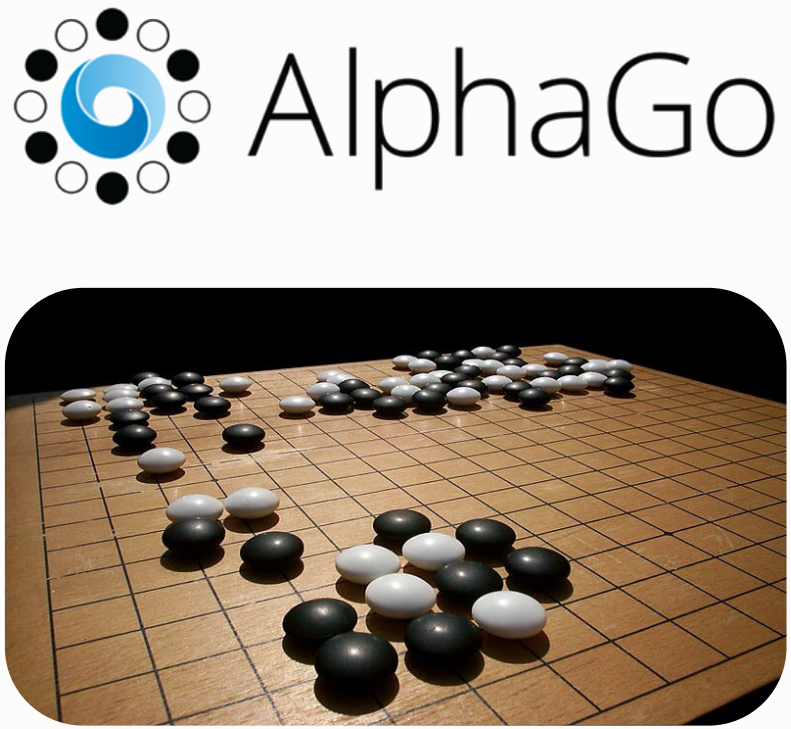
2. Présenter quelques **limites** du calcul automatique.

3. Donner un aperçu de **problèmes informatiques difficiles** :
   - qu'on ne peut pas résoudre par algorithme,
   - demandant trop de ressources pour une résolution exacte.

4. Donner quelques jalons historiques.

Des progrès impressionnants

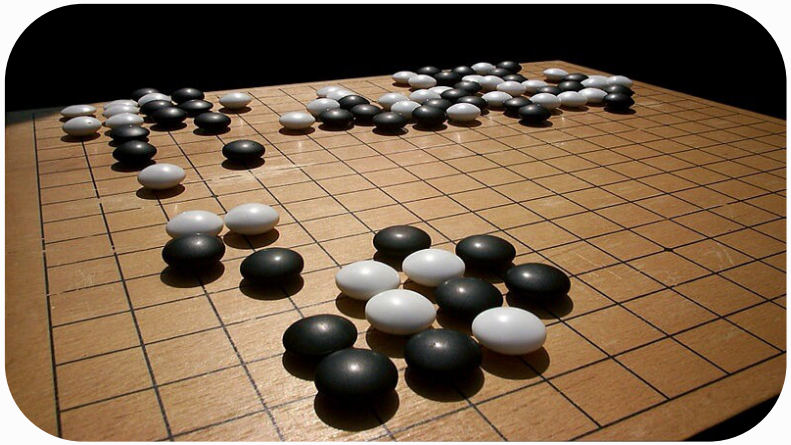
2016 : AlphaGo



Des progrès impressionnants

2016 : AlphaGo

AlphaGo



2015 : Gérard Berry

Gérard Berry : « L'ordinateur est complètement con »

« Fondamentalement, l'ordinateur et l'homme sont les deux opposés les plus intégraux qui existent. » Entretien avec Gérard Berry, informaticien et professeur au Collège de France, médaille d'or 2014 du CNRS.

Qu'est-ce qu'un problème informatique ?

Ceci **n'est pas** un problème informatique :

Résolution d'une équation du second degré

L'équation $42x^2 + 7x + 8 = 0$ a-t-elle au moins une solution dans $\mathbb{R}$?

Qu'est-ce qu'un problème informatique ?

Ceci **n'est pas** un problème informatique :

Résolution d'une équation du second degré

L'équation $42x^2 + 7x + 8 = 0$ a-t-elle au moins une solution dans $\mathbb{R}$?

Ceci **est** un problème informatique :

« Résolution » de toutes les équations du second degré

Existe-t-il une **méthode automatique** qui,

- étant **donnés** des entiers a, b, c (avec $a \neq 0$),
- détermine si $ax^2 + bx + c = 0$ a au moins une solution dans $\mathbb{R}$?

Qu'est-ce qu'un problème informatique ?

Ceci **n'est pas** un problème informatique :

Résolution d'une équation du second degré

L'équation $42x^2 + 7x + 8 = 0$ a-t-elle au moins une solution dans $\mathbb{R}$?

Ceci **est** un problème informatique :

« Résolution » de toutes les équations du second degré

Existe-t-il une **méthode automatique** qui,

- étant **donnés** des entiers a, b, c (avec $a \neq 0$),
- détermine si $ax^2 + bx + c = 0$ a au moins une solution dans $\mathbb{R}$?

Remarque

Une méthode automatique de résolution de toutes les équations du second degré permet en particulier de résoudre le premier exercice.

Qu'est-ce qu'un problème informatique ?

Take away : qu'est-ce qu'un problème informatique ?

Problème = **question** portant sur des **instances** (ou **entrées**).

Pour les problèmes intéressants, l'instance provient d'un **ensemble infini**.

Qu'est-ce qu'un problème informatique ?

Take away : qu'est-ce qu'un problème informatique ?

Problème = **question** portant sur des **instances** (ou **entrées**).

Pour les problèmes intéressants, l'instance provient d'un **ensemble infini**.

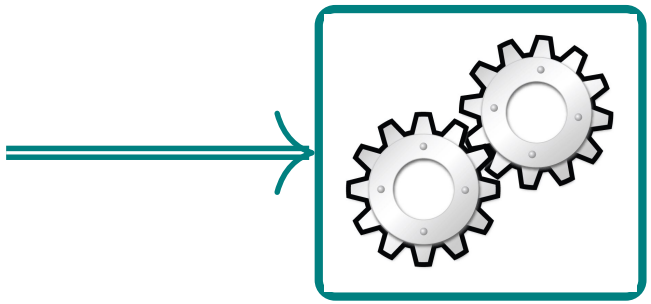
Si la réponse attendue est **oui** ou **non**, on parle de problème de **décision**.

Un algorithme « résout » un problème

Algorithme

Entrée

(venant d'un ensemble infini)

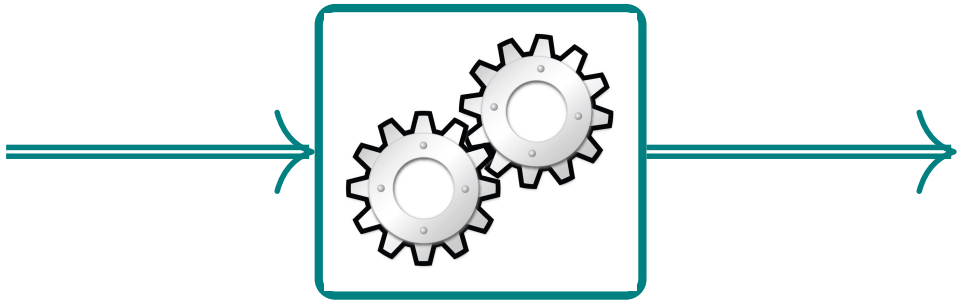


Un algorithme « résout » un problème

Algorithme

Entrée

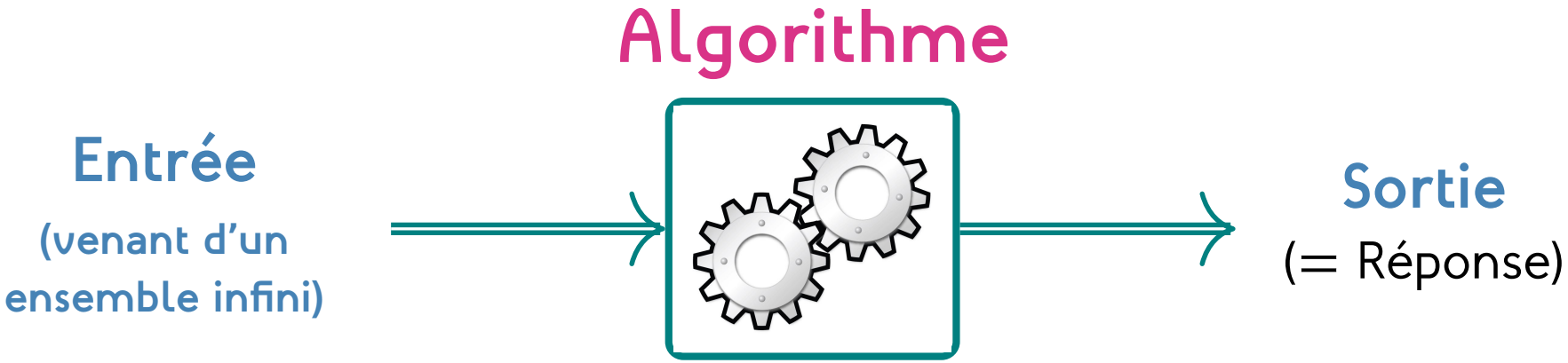
(venant d'un ensemble infini)



Sortie

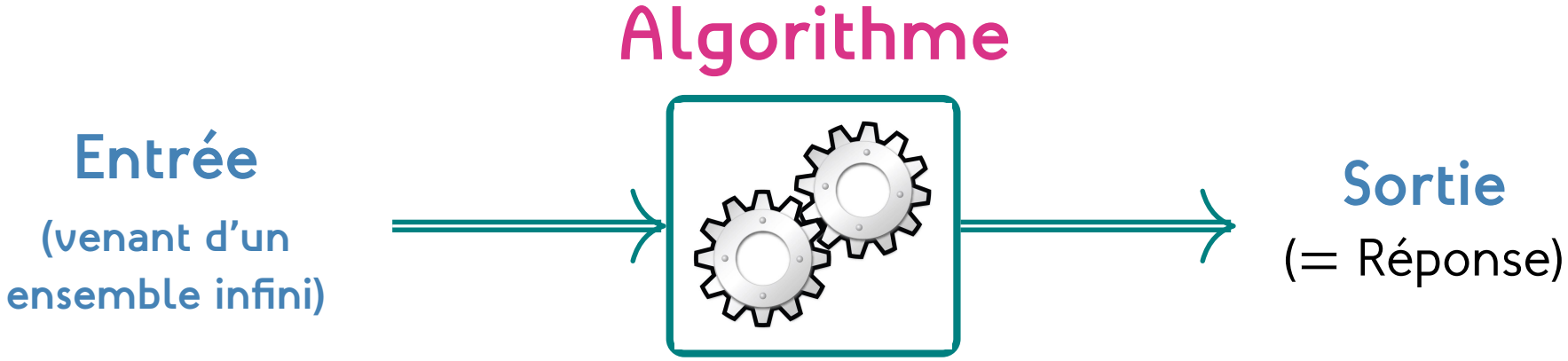
(= Réponse)

Un algorithme « résout » un problème



En particulier, un algorithme **s'arrête sur chaque entrée**.

Un algorithme « résout » un problème



En particulier, un algorithme **s'arrête sur chaque entrée**.

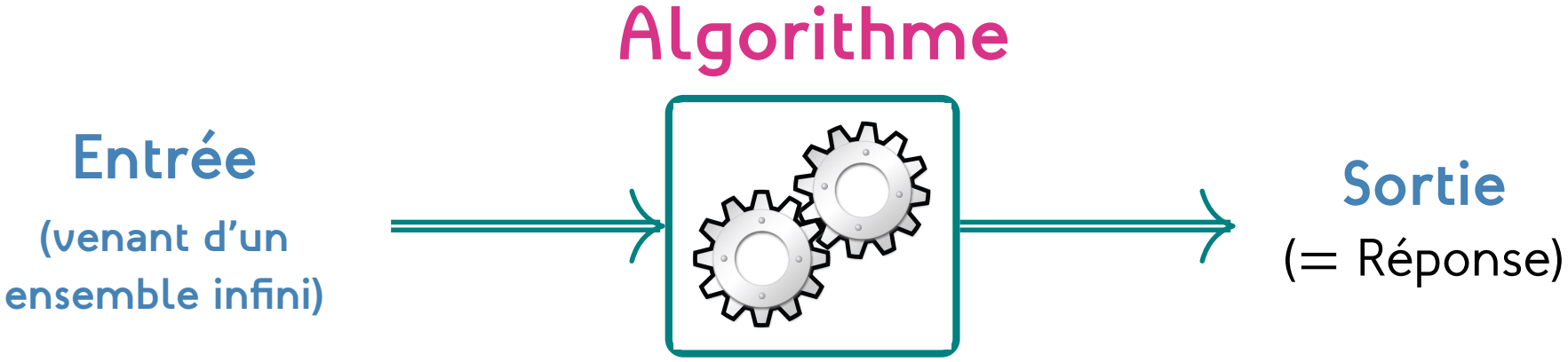
Exemples

- Algorithme de résolution d'équations du second degré.

  Entrée attendue : 3 entiers (a, b, c).

  Sortie attendue : Solutions réelles de $ax^2 + bx + c = 0$.

Un algorithme « résout » un problème



En particulier, un algorithme **s'arrête sur chaque entrée**.

Exemples

- Algorithme de résolution d'équations du second degré.

  Entrée attendue : 3 entiers (a, b, c).
  Sortie attendue : Solutions réelles de $ax^2 + bx + c = 0$.

- Algorithme de résolution d'équations **Diophantiennes** à **une** variable.

  Entrée attendue : Une liste d'entiers $a_0, \ldots, a_n$ (où $a_0 \neq 0$).
  Sortie attendue : Solutions **entières** de $\sum_{i=0}^{n} a_i x^i = 0$.

Équations Diophantiennes à une variable

Problème : résolution d'équations Diophantiennes à une variable

Instance : Une liste d'entiers $a_0, \ldots, a_n$ (avec $a_0 \neq 0$).

Sortie : Solutions **entières** de $\sum_{i=0}^{n} a_i x^i = 0$.

Équations Diophantiennes à une variable

Problème : résolution d'équations Diophantiennes à une variable

Instance : Une liste d'entiers $a_0, \ldots, a_n$ (avec $a_0 \neq 0$).

Sortie : Solutions **entières** de $\sum_{i=0}^{n} a_i x^i = 0$.

Une propriété simple

Les éventuelles solutions sont dans $[-|a_0|, |a_0|]$.

Équations Diophantiennes à une variable

Problème : résolution d'équations Diophantiennes à une variable

Instance : Une liste d'entiers $a_0, \ldots, a_n$ (avec $a_0 \neq 0$).

Sortie : Solutions **entières** de $\sum_{i=0}^{n} a_i x^i = 0$.

Une propriété simple

Les éventuelles solutions sont dans $[-|a_0|, |a_0|]$.

Démonstration. Si x est solution non nulle, on a :

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = 0,$$

c'est-à-dire,

$$a_0 = -(a_1 x + a_2 x^2 + \cdots + a_n x^n).$$

Comme x divise le membre droit, il divise a_0. Comme $a_0 \neq 0$, tout diviseur de a_0 (donc x) doit appartenir à l'intervalle $[-|a_0|, |a_0|]$. □

Équations Diophantiennes à une variable

Problème : résolution d'équations Diophantiennes à une variable

Instance : Une liste d'entiers $a_0, \ldots, a_n$ (avec $a_0 \neq 0$).

Sortie : Solutions **entières** de $\sum_{i=0}^{n} a_i x^i = 0$.

Une propriété simple

Les éventuelles solutions sont dans $[-|a_0|, |a_0|]$.

Algorithme de résolution basé sur cette propriété

1. Calculer $M = |a_0|$.
2. Initialiser une liste vide pour mémoriser les solutions.
3. Pour chaque entier z entre $-M$ et M,
   - 3.1 Tester si $\sum_{i=0}^{n} a_i z^i = 0$.
   - 3.2 Si oui, ajouter z à la liste des solutions.
4. Retourner la liste des solutions.

Équations Diophantiennes à une variable

Problème : résolution d'équations Diophantiennes à une variable

Instance : Une liste d'entiers $a_0, \ldots, a_n$ (avec $a_0 \neq 0$).

Sortie : Solutions **entières** de $\sum_{i=0}^{n} a_i x^i = 0$.

Une propriété simple

Les éventuelles solutions sont dans $[-|a_0|, |a_0|]$.

Algorithme de résolution basé sur cette propriété

1. Calculer $M = |a_0|$.
2. Initialiser une liste vide pour mémoriser les solutions.
3. Pour chaque entier z entre $-M$ et M,
   - 3.1 Tester si $\sum_{i=0}^{n} a_i z^i = 0$.
   - 3.2 Si oui, ajouter z à la liste des solutions.
4. Retourner la liste des solutions.

Algo « force brute »
(pas besoin de comprendre l'équation)

Récapitulatif

Vocabulaire important

- Problème (en informatique).
- Algorithme.
- Instance (ou entrée) d'un problème (ou d'un algorithme).
- Sortie.

Récapitulatif

Vocabulaire important

- Problème (en informatique).
- Algorithme.
- Instance (ou entrée) d'un problème (ou d'un algorithme).
- Sortie.

Problème « intéressant en info » ⇒ infinité d'instances

Les problèmes suivants n'entrent **pas** dans notre cadre.

- Conjecture de Syracuse « $3x + 1$ »,
- Conjecture de Goldbach,
- Conjecture des nombres premiers jumeaux.

Exemples de problèmes de décision

Problème 0

Instance Un nombre entier positif n.

Question n est-il pair ?

Exemples de problèmes de décision

Problème 0

Instance Un nombre entier positif n.

Question n est-il pair ?

```
def est_pair(n):
  return n % 2 == 0
```

Exemples de problèmes de décision

Problème 0

Instance Un nombre entier positif n.

Question n est-il pair ?

Problème 1

I. Un nombre entier positif n.

Q. n est-il premier ?

Exemples de problèmes de décision

Problème 0

Instance Un nombre entier positif n.

Question n est-il pair ?

Problème 1

I. Un nombre entier positif n.

Q. n est-il premier ?

```
def est_premier(n):
  return (n == 2 or
          # cas impair
          n > 2 and n % 2 != 0 and
          all(n % d != 0 for d in range(3, int(n**0.5)+1, 2)))
```

Exemples de problèmes de décision

Problème 0

Instance Un nombre entier positif n.

Question n est-il pair ?

Problème 1

I. Un nombre entier positif n.

Q. n est-il premier ?

```
def est_premier(n):
  return (n == 2 or
          # cas impair
          n > 2 and n % 2 != 0 and
          all(n % d != 0 for d in range(3, int(n**0.5)+1, 2)))
```

Remarque

En fait, cet algorithme naïf est **mauvais** (pourquoi ?).

Exemples de problèmes de décision

Problème 0

Instance : Un nombre entier positif n.

Question : n est-il pair ?

Problème 1

I. Un nombre entier positif n.

Q. n est-il premier ?

Problème 2

I. Un programme en Python.

Q. Le programme est-il syntaxiquement correct ?

Exemples de problèmes de décision

Problème 0

Instance : Un nombre entier positif n.

Question : n est-il pair ?

Problème 1

I. Un nombre entier positif n.

Q. n est-il premier ?

Problème 2

I. Un programme en Python.

Q. Le programme est-il syntaxiquement correct ?

Ce n'est pas immédiat, mais cela peut se faire en temps linéaire.

Exemples de problèmes de décision (2)

Problème 3 (Sudoku)

Instance : Une grille de Sudoku $n^2 \times n^2$.

Question : La grille a-t-elle une solution ?

Problème 4 (Eternity)

I. Un puzzle Eternity $n \times n$.

Q. Le puzzle a-t-il une solution ?

Problème 5 (3-COL)

I. Un graphe.

Q. Le graphe a-t-il une 3-coloration ?

Problème 6 (SAT)

I. Une formule propositionnelle φ.

Q. L'équation $\varphi =$ **True** a-t-elle une solution ?

Problème 7 (TSP)

I. Un entier k, des villes, les inter-distances.

Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par toutes les villes ?

Exemples de problèmes de décision (2)

Problème 3 (Sudoku)
Instance : Une grille de Sudoku $n^2 \times n^2$.
Question : La grille a-t-elle une solution ?

Problème 4 (Eternity)
I. Un puzzle Eternity $n \times n$.
Q. Le puzzle a-t-il une solution ?

Problème 5 (3-COL)
I. Un graphe.
Q. Le graphe a-t-il une 3-coloration ?

Problème 6 (SAT)
I. Une formule propositionnelle φ.
Q. L'équation $\varphi = \textbf{True}$ a-t-elle une solution ?

Problème 7 (TSP)
I. Un entier k, des villes, les inter-distances.
Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par toutes les villes ?

Problème 3 : Sudoku $n^2 \times n^2$

Instances = grilles partiellement remplies.

Deux exemples d'instances (pour $n = 2$ et $n = 3$) :

		1	
4			
			2
	3		

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Question : la grille d'entrée a-t-elle une solution ?

Exemples de problèmes de décision (2)

Problème 3 (Sudoku)
- Instance : Une grille de Sudoku $n^2 \times n^2$.
- Question : La grille a-t-elle une solution ?

Problème 4 (Eternity)
- I. Un puzzle Eternity $n \times n$.
- Q. Le puzzle a-t-il une solution ?

Problème 5 (3-COL)
- I. Un graphe.
- Q. Le graphe a-t-il une 3-coloration ?

Problème 6 (SAT)
- I. Une formule propositionnelle φ.
- Q. L'équation $\varphi = \textbf{True}$ a-t-elle une solution ?

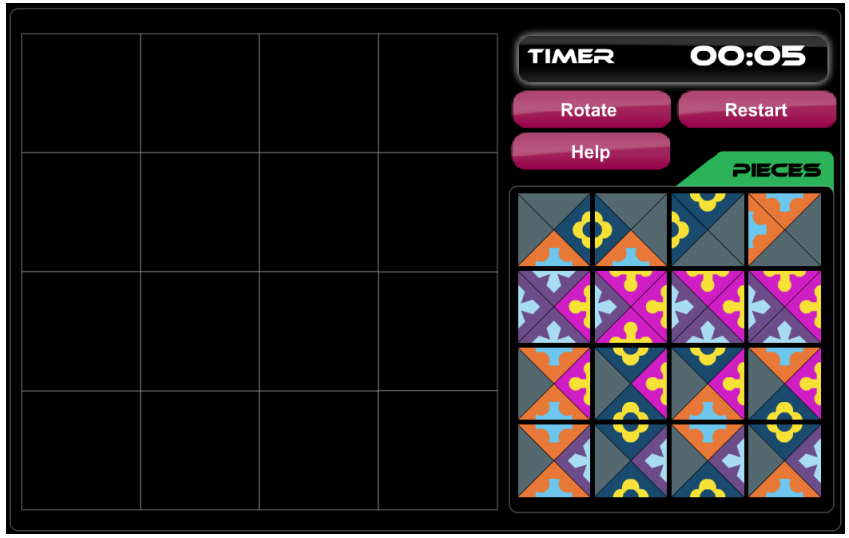
Problème 7 (TSP)
- I. Un entier k, des villes, les inter-distances.
- Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par toutes les villes ?

Problème 4 : Eternity II $n \times n$

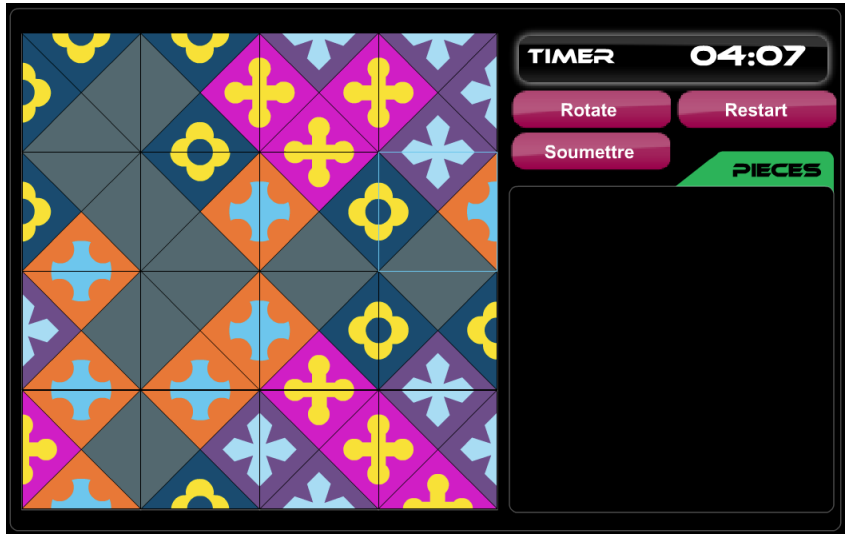
Puzzle $n \times n$ proposé vers 2010, 2M$ pour une solution 16×16 (Wikipedia).

Ci-dessous, un exemple d'instance en taille 4×4.

Au début du jeu



À la fin du jeu



Exemples de problèmes de décision (2)

Problème 3 (Sudoku)

Instance : Une grille de Sudoku $n^2 \times n^2$.

Question : La grille a-t-elle une solution ?

Problème 4 (Eternity)

I. Un puzzle Eternity $n \times n$.

Q. Le puzzle a-t-il une solution ?

Problème 5 (3-COL)

I. Un graphe.

Q. Le graphe a-t-il une 3-coloration ?

Problème 6 (SAT)

I. Une formule propositionnelle φ.

Q. L'équation $\varphi = \texttt{True}$ a-t-elle une solution ?

Problème 7 (TSP)

I. Un entier k, des villes, les inter-distances.

Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par toutes les villes ?

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Historique : 4-coloration de cartes géographiques (F. Guthrie, 1852).

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

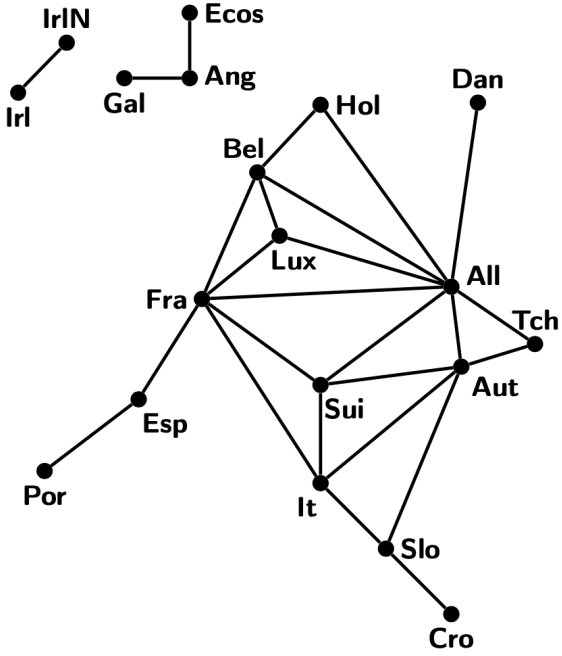
Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Historique : 4-coloration de cartes géographiques (F. Guthrie, 1852).



Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

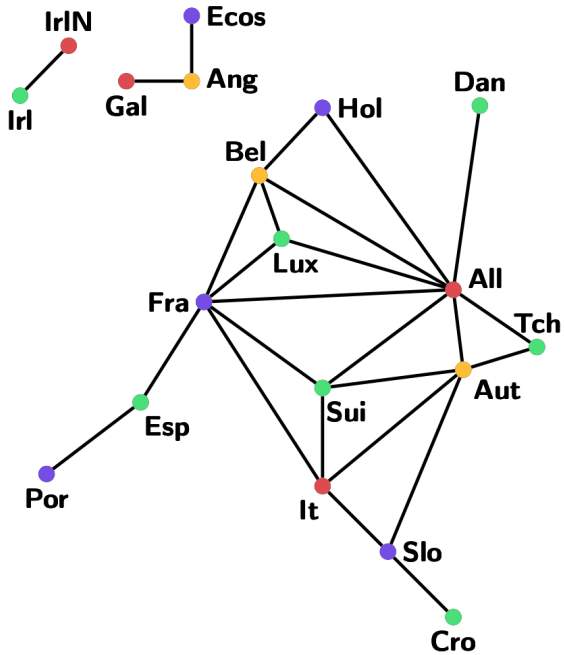
Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Historique : 4-coloration de cartes géographiques (F. Guthrie, 1852).



Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

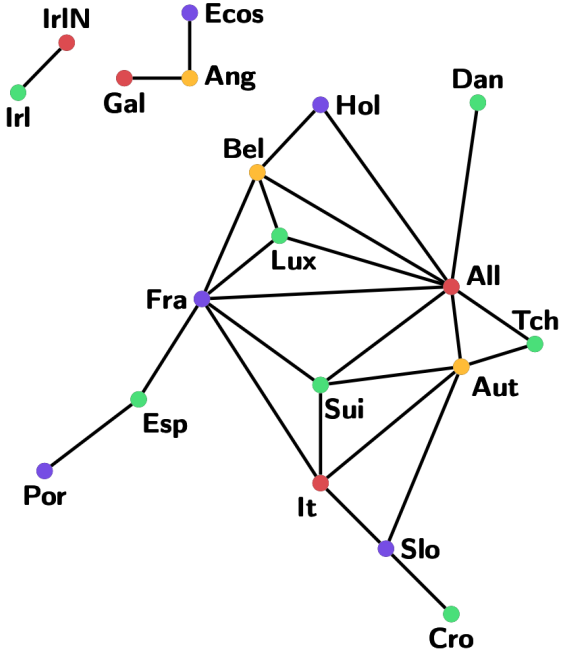
Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Historique : 4-coloration de cartes géographiques (F. Guthrie, 1852).



Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Théorème des 4 couleurs (Appel, Haken, 1976)

Tout graphe **planaire** peut être colorié avec 4 couleurs.

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Théorème des 4 couleurs (Appel, Haken, 1976)

Tout graphe **planaire** peut être colorié avec 4 couleurs.

Remarque

On ne peut **pas** colorier tout graphe planaire avec 3 couleurs (pourquoi ?).

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Théorème des 4 couleurs (Appel, Haken, 1976)

Tout graphe **planaire** peut être colorié avec 4 couleurs.

Remarque

On ne peut **pas** colorier tout graphe planaire avec 3 couleurs (pourquoi ?).

La remarque conduit à un autre problème informatique

Y a-t-il un algorithme pour **tester** si un graphe planaire est 3-colorable ?
C'est le problème 3-COL restreint aux graphes planaires.

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Remarque annexe

Sudoku $n^2 \times n^2$ revient à colorer un graphe partiellement coloré.

Pourquoi ?

Indication C'est un graphe à n^4 sommets, le nombre de couleurs est n^2.

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Problème d'emplois du temps et modélisation par des graphes

Instance

- des examens $e_1, \ldots, e_n$ (chacun de même durée).
- des conflits entre examens (ne pouvant occuper le même créneau).
- Un entier $k \geqslant 0$.

Question peut-on organiser les examens sur au plus k créneaux ?

Problème 5 : coloration de graphes

Problème 5 bis (k-COL)

Instance Un graphe et un entier $k > 0$.

Question Le graphe a-t-il une k-coloration ?

Coloration : des sommets voisins **ne doivent pas** avoir même couleur.

k-coloration : coloration utilisant au plus k couleurs.

Problème d'emplois du temps et modélisation par des graphes

Instance

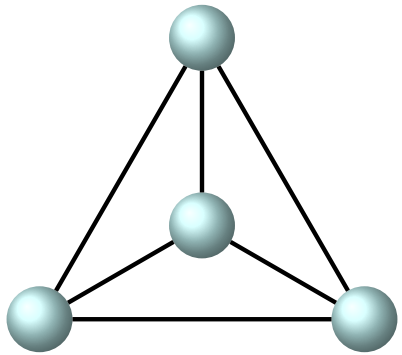
- des examens $e_1, \ldots, e_n$ (chacun de même durée).
- des conflits entre examens (ne pouvant occuper le même créneau).
- Un entier $k \geqslant 0$.

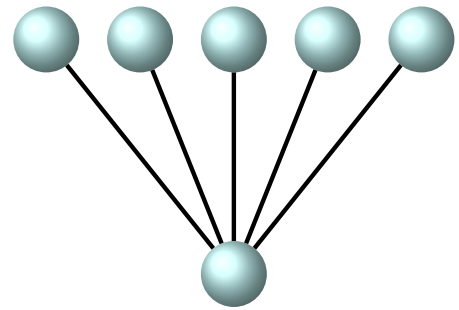
Question peut-on organiser les examens sur au plus k créneaux ?

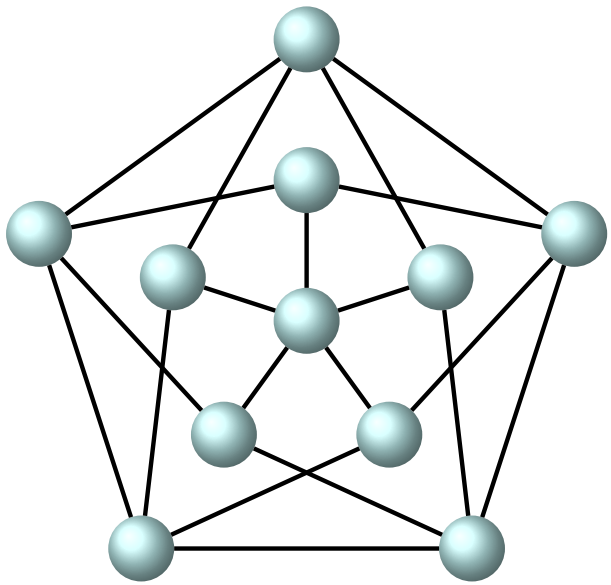
Revient à résoudre un problème de coloration de graphes (lequel ?).

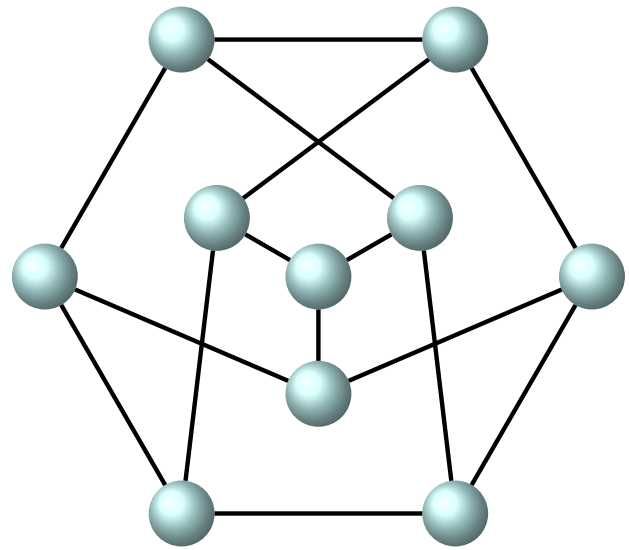
Problème 5 : 3-coloration de graphes

Parmi les graphes suivants, lesquels sont 3-colorables ?









Exemples de problèmes de décision (2)

Problème 3 (Sudoku)
- Instance : Une grille de Sudoku $n^2 \times n^2$.
- Question : La grille a-t-elle une solution ?

Problème 4 (Eternity)
- I. Un puzzle Eternity $n \times n$.
- Q. Le puzzle a-t-il une solution ?

Problème 5 (3-COL)
- I. Un graphe.
- Q. Le graphe a-t-il une 3-coloration ?

Problème 6 (SAT)
- I. Une formule propositionnelle φ.
- Q. L'équation $\varphi =$ **True** a-t-elle une solution ?

Problème 7 (TSP)
- I. Un entier k, des villes, les inter-distances.
- Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par toutes les villes ?

Problème 6 : SAT

Instance Une formule propositionnelle φ.

Question L'équation $\varphi = \textbf{True}$ a-t-elle une solution ?
Si oui, on dit que φ est **satisfaisable**.

Formule propositionnelle : définition

Formule construite à partir de variables $a, b, \ldots$ en utilisant les opérateurs

- $\wedge$ (et logique),
- $\vee$ (ou logique),
- $\neg$ (négation logique).

Problème 6 : SAT

Instance Une formule propositionnelle φ.

Question L'équation $\varphi = \textbf{True}$ a-t-elle une solution ?
Si oui, on dit que φ est **satisfaisable**.

Formule propositionnelle : définition

Formule construite à partir de variables $a, b, \ldots$ en utilisant les opérateurs

- $\wedge$ (et logique),
- $\vee$ (ou logique),
- $\neg$ (négation logique).

Exemple d'instance de SAT (forme CNF)

$$\varphi = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c)$$

Problème 6 : SAT

Instance Une formule propositionnelle φ.

Question L'équation $\varphi =$ **True** a-t-elle une solution ?
Si oui, on dit que φ est **satisfaisable**.

Formule propositionnelle : définition

Formule construite à partir de variables $a, b, \ldots$ en utilisant les opérateurs

- $\wedge$ (et logique),
- $\vee$ (ou logique),
- $\neg$ (négation logique).

Exemple d'instance de SAT (forme CNF)

$$\varphi = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c)$$

Quelle est la réponse à SAT sur cette instance ?

i.e., pour cette formule φ, l'équation $\varphi =$ **True** a-t-elle une solution ?

Exemple d'instance et algorithme

Un algorithme possible : la « table de vérité »

$\varphi = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c)$

a	b	c	$\neg a \vee b$	$\neg a \vee \neg b$	$a \vee \neg b$	$a \vee \neg c$	φ
F	F	F	T	T	T	T	T
F	F	T	T	T	T	F	F
F	T	F	T	T	F	T	F
F	T	T	T	T	F	F	F
T	F	F	F	T	T	T	F
T	F	T	F	T	T	T	F
T	T	F	T	F	T	T	F
T	T	T	T	F	T	T	F

La réponse attendue pour SAT sur l'instance φ est **OUI**.

Exemple d'instance et algorithme

Un algorithme possible : la « table de vérité »

$\varphi = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c) \wedge (a \vee b \vee c)$

a	b	c	$b \vee \neg a$	$\neg a \vee \neg b$	$a \vee \neg b$	$a \vee \neg c$	$a \vee b \vee c$	φ
F	F	F	T	T	T	T	F	F
F	F	T	T	T	T	F	T	F
F	T	F	T	T	F	T	T	F
F	T	T	T	T	F	F	T	F
T	F	F	F	T	T	T	T	F
T	F	T	F	T	T	T	T	F
T	T	F	T	F	T	T	T	F
T	T	T	T	F	T	T	T	F

La réponse attendue pour SAT sur l'instance $\varphi \wedge (a \vee b \vee c)$ est **NON**.

Exemple d'instance et algorithme

Un algorithme possible : la « table de vérité »

$\varphi = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c) \wedge (a \vee b \vee c)$

a	b	c	$b \vee \neg a$	$\neg a \vee \neg b$	$a \vee \neg b$	$a \vee \neg c$	$a \vee b \vee c$	φ
F	F	F	T	T	T	F	F	F
F	F	T	T	T	T	F	T	F
F	T	F	T	T	F	T	T	F
F	T	T	T	T	F	F	T	F
T	F	F	F	T	T	T	T	F
T	F	T	F	T	T	T	T	F
T	T	F	T	F	T	T	T	F
T	T	T	T	F	T	T	T	F

Algo « force brute »

Pas de compréhension de la formule

La réponse attendue pour SAT sur l'instance $\varphi \wedge (a \vee b \vee c)$ est **NON**.

Exemple d'instance et algorithme

Un algorithme possible : la « table de vérité »

$\varphi = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c) \wedge (a \vee b \vee c)$

a	b	c	$b \vee \neg a$	$\neg a \vee \neg b$	$a \vee \neg b$	$a \vee \neg c$	$a \vee b \vee c$	φ
F	F	F	T	T	T	F	F	F
F	F	T	T	T	T	F	T	F
F	T	F	T	T	F	T	T	F
F	T	T	T	T	F	F	T	F
T	F	F	F	T	T	T	T	F
T	F	T	F	T	T	T	T	F
T	T	F	T	F	T	T	T	F
T	T	T	T	F	T	T	T	F

Algo « force brute »

Pas de compréhension de la formule

Taille de l'espace de recherche ?

La réponse attendue pour SAT sur l'instance $\varphi \wedge (a \vee b \vee c)$ est **NON**.

Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).

Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.

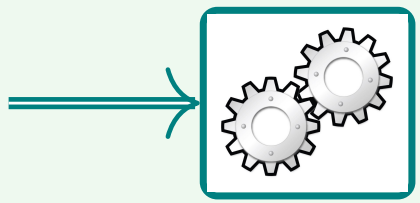
Instance G
de 3-COL

Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.

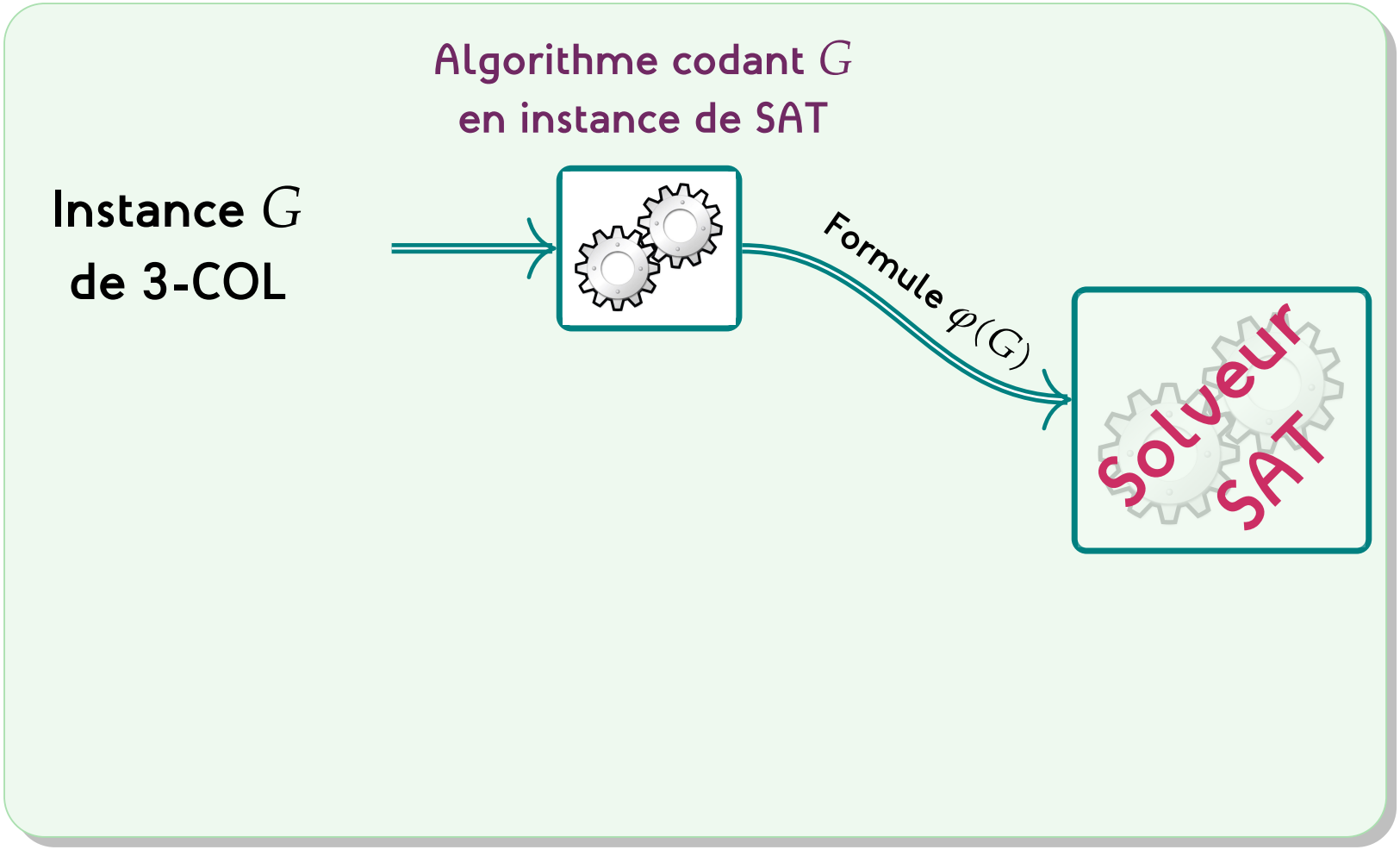
Algorithme codant G en instance de SAT

Instance G de 3-COL



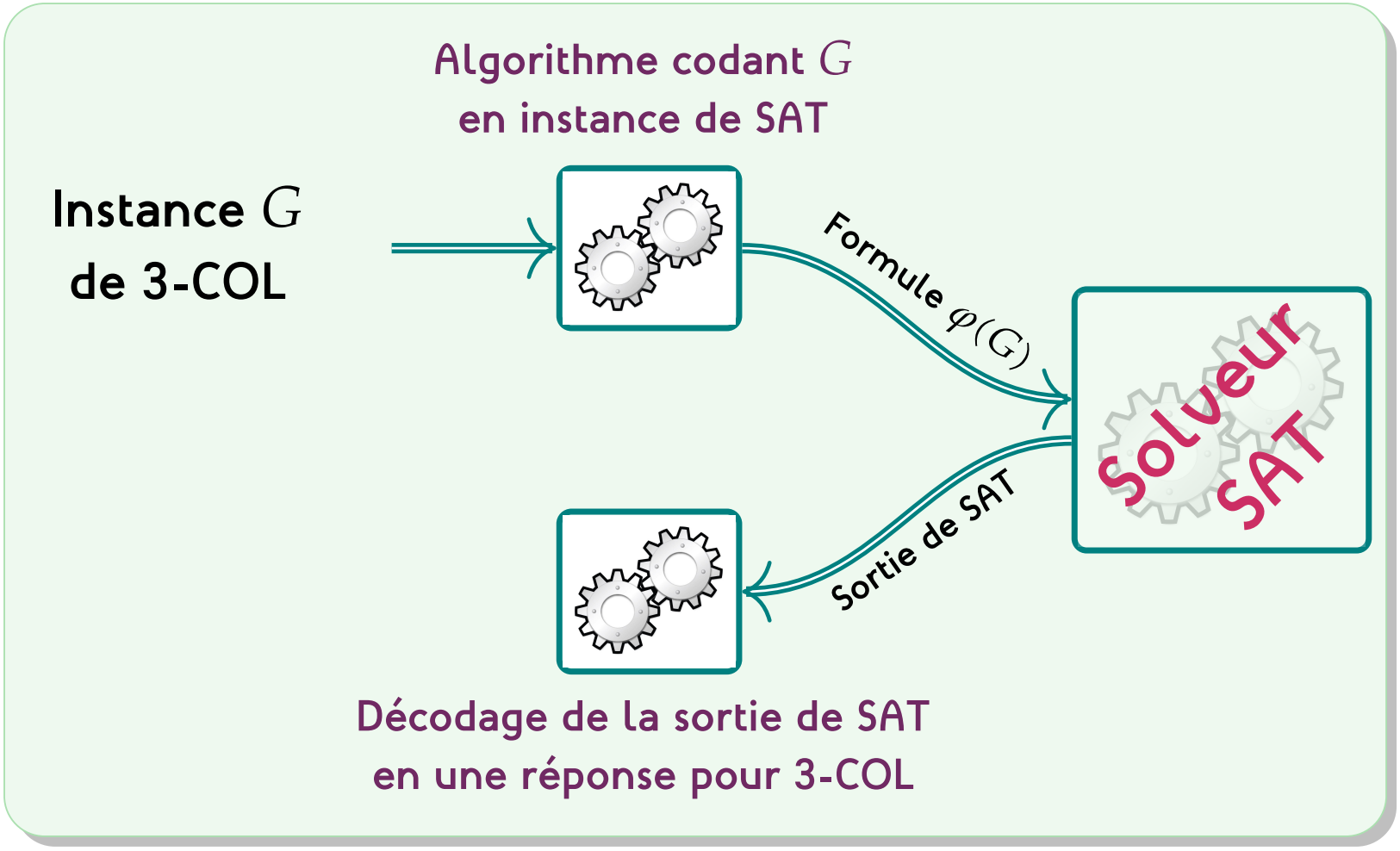
Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.



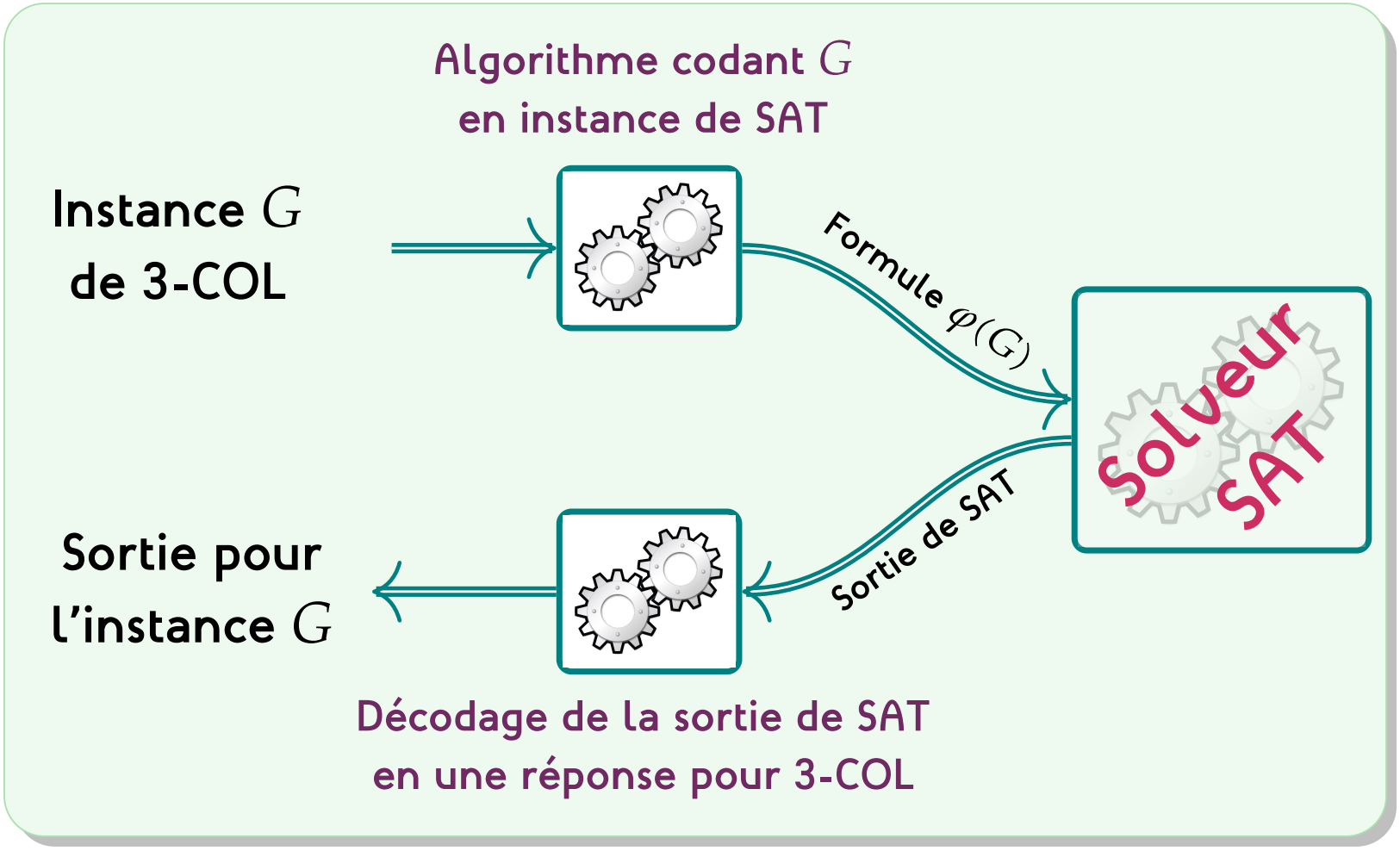
Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.



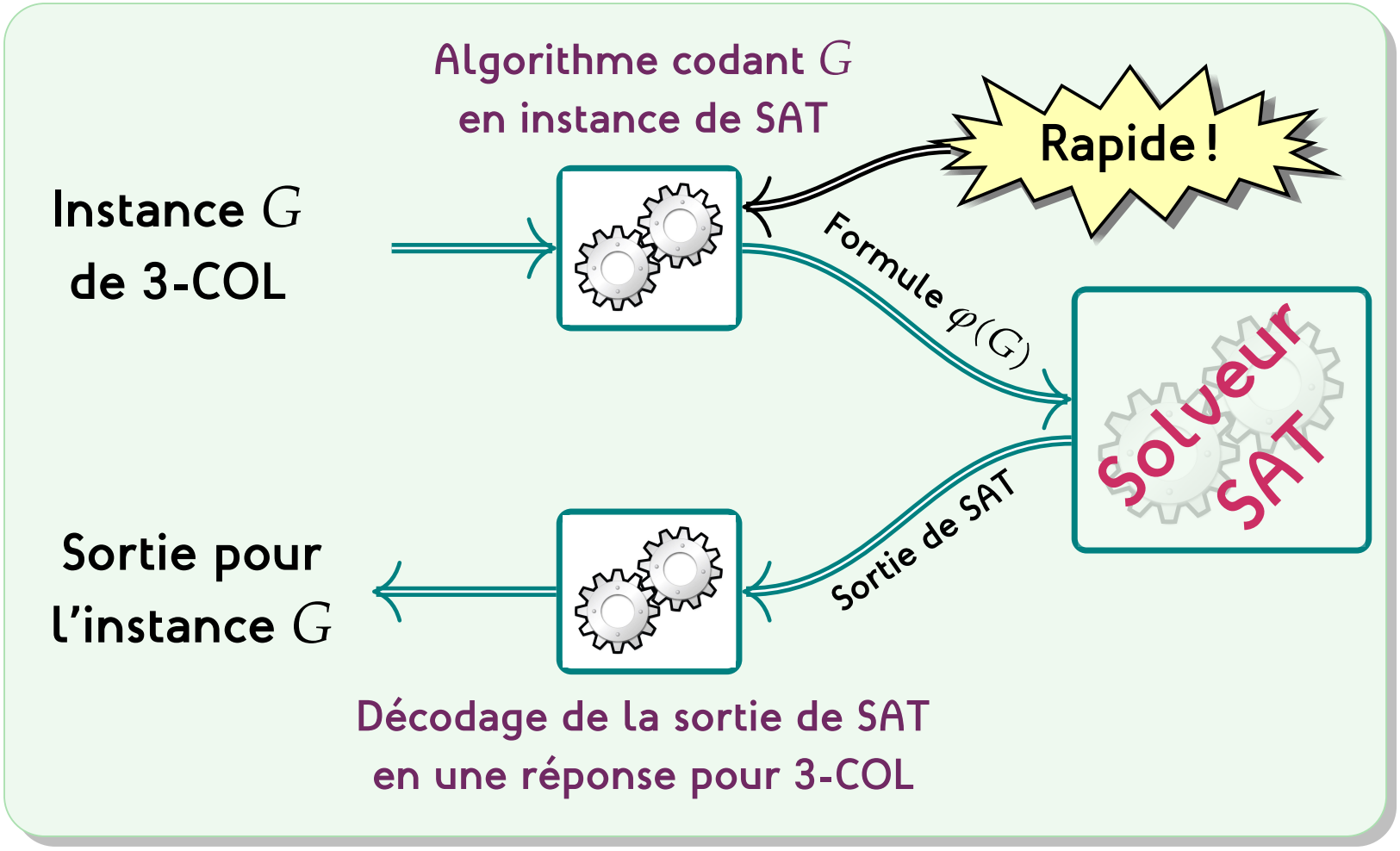
Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.



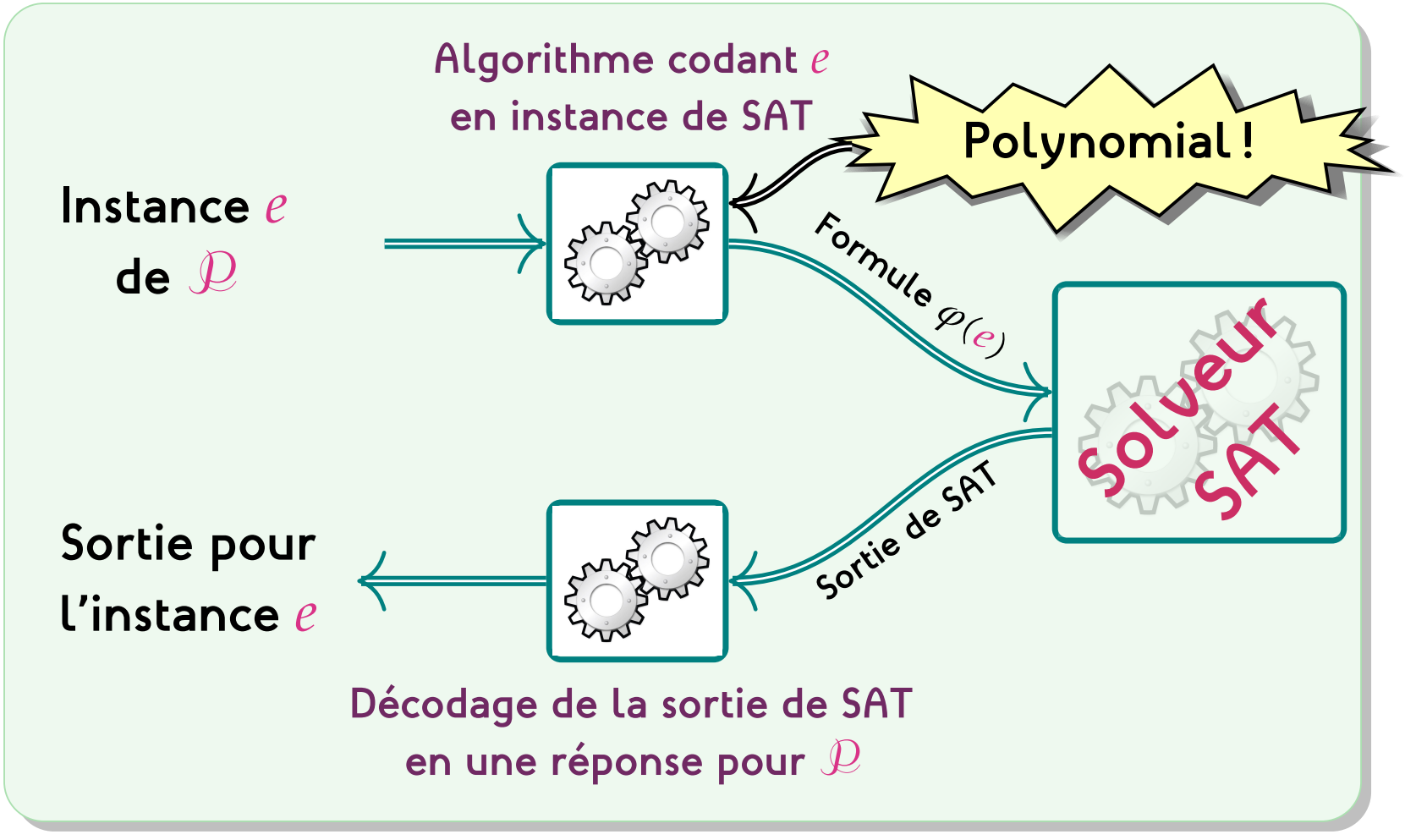
Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.



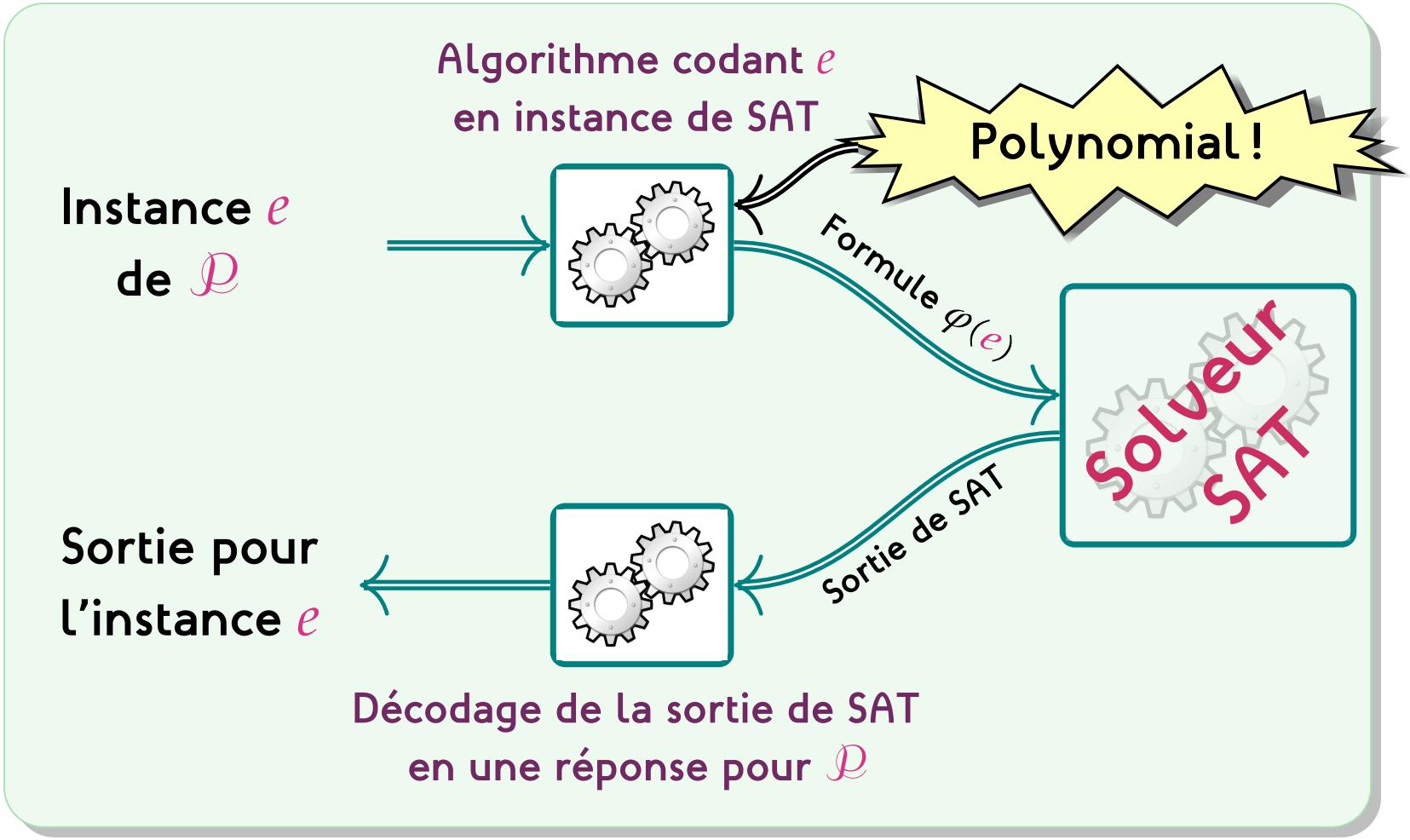
Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à SAT.



Pourquoi SAT est-il si important ?

- Premier problème montré NP-complet (Cook & Levin, 1971).
- Les problèmes de la classe NP se réduisent polynomialement à **SAT**.



- **Solveurs spécialisés SAT**, implémentant de nombreuses heuristiques.

Solveurs SAT

- Espace de recherche : $2^{|\#\text{Vars}|}$, rapidement gigantesque.
- Cependant, résolvent des instances à plusieurs millions de variables.

Citation de Moshe Vardi (lien)

When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.

Indeed, there are SAT instances with a few hundred variables that cannot be solved by any existant SAT solver.

But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!

Solveurs SAT : exemples

Solveurs SAT librement disponibles : Minisat, Glucose, Cryptominisat, etc.

Ex. $\varphi_1 = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c)$

Fichier source (format)

```
$ cat phi1.cnf
c 3 variables, 4 clauses
p cnf 3 4
c Clause = liste des variables
c          terminée par 0
c a <-> 1, b <-> 2, c <-> 3
c not(x_i) <-> -i
-1 2 0
-1 -2 0
1 -2 0
1 -3 0
```

Sortie solveur SAT

```
$ cryptominisat5 --verb=0 phi1.cnf
s SATISFIABLE
v -1 -2 -3 0
```

Solveurs SAT : exemples

Solveurs SAT librement disponibles : Minisat, Glucose, Cryptominisat, etc.

Ex. $\varphi_2 = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (a \vee \neg c) \wedge (a \vee b \vee c)$

Fichier source (format)

```
$ cat phi2.cnf
c 3 variables, 5 clauses
p cnf 3 5
c Clause = liste des variables
c          terminée par 0
c a <-> 1, b <-> 2, c <-> 3
c not(x_i) <-> -i
-1 2 0
-1 -2 0
1 -2 0
1 -3 0
1 2 3 0
```

Sortie solveur SAT

```
$ glucose -verb=0 phi2.cnf
c
c This is glucose-syrup 4.0 ([...])
--  based on MiniSAT ([...])
c
s UNSATISFIABLE
```

Exemples de problèmes de décision (2)

Problème 3 (Sudoku)
- I. Une grille de Sudoku $n^2 \times n^2$.
- Q. La grille a-t-elle une solution ?

Problème 4 (Eternity)
- I. Un puzzle Eternity $n \times n$.
- Q. Le puzzle a-t-il une solution ?

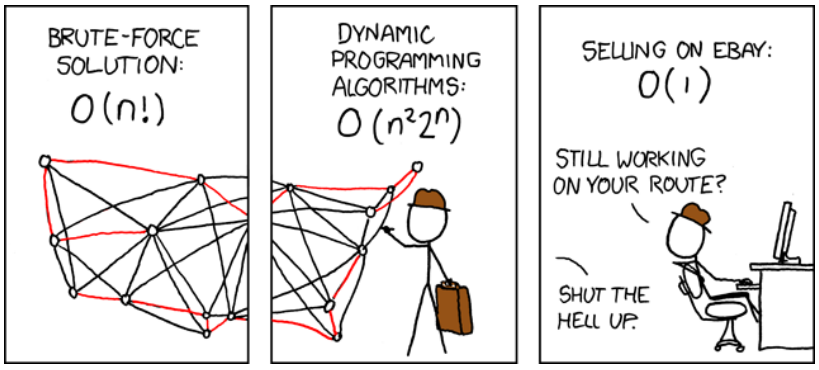
Problème 5 (3-COL)
- I. Un graphe.
- Q. Le graphe a-t-il une 3-coloration ?

Problème 6 (SAT)
- I. Une formule propositionnelle φ.
- Q. L'équation $\varphi = \texttt{True}$ a-t-elle une solution ?

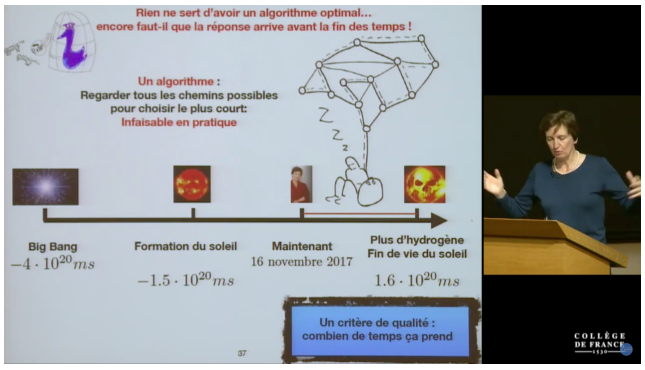
Problème 7 (TSP)
- I. Un entier k, des villes, les inter-distances.
- Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par toutes les villes ?

Problème 7 : TSP

« Problème fil rouge » UE TAP de L3, TSP.



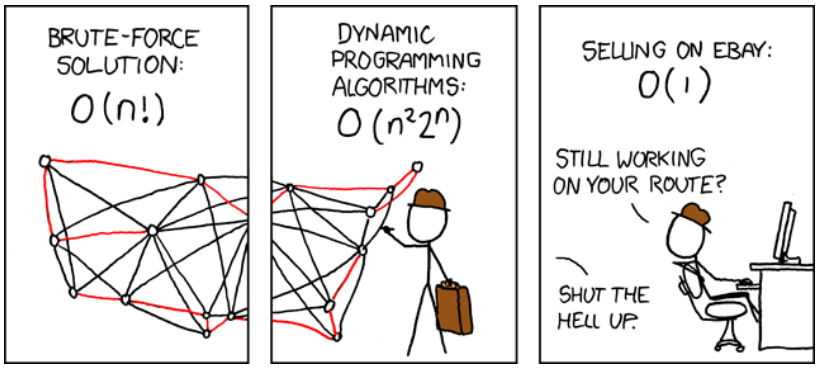
©XKCD



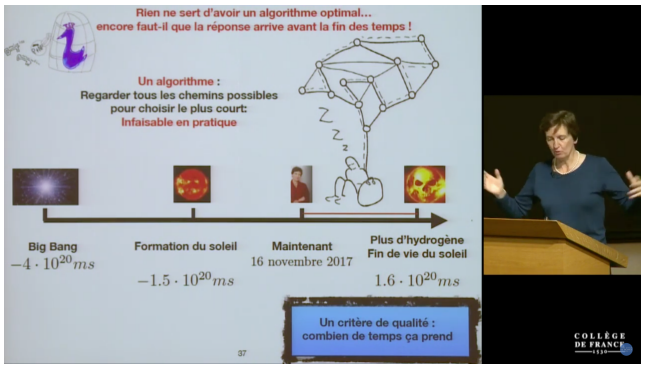
Claire Mathieu @ Collège de France

Problème 7 : TSP

« Problème fil rouge » UE TAP de L3, **TSP**.

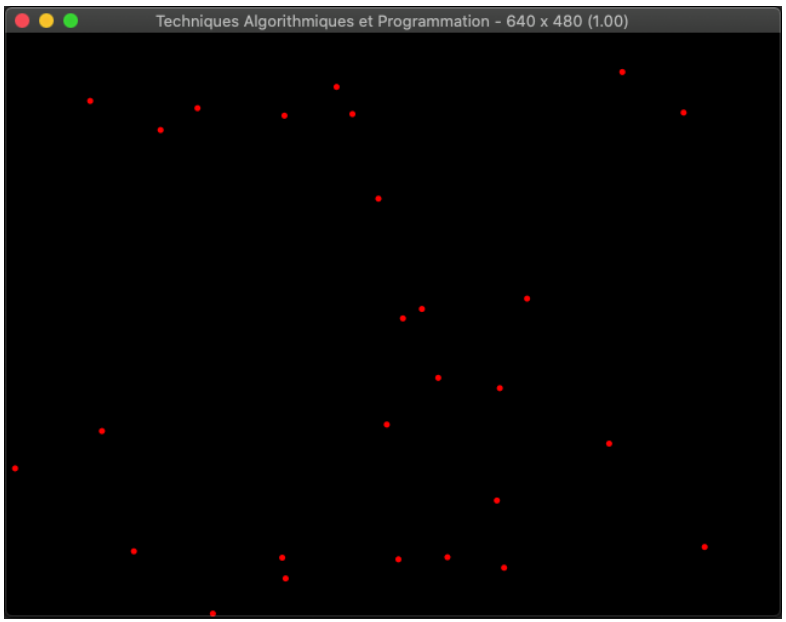


©XKCD



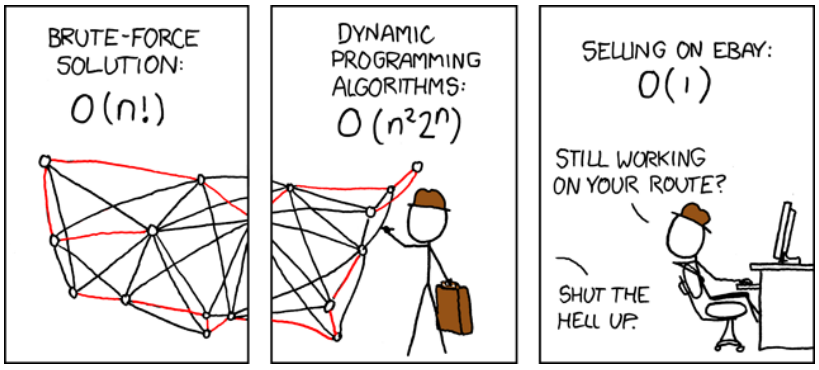
Claire Mathieu @ Collège de France

27 villes, soit **27 ! tournées** (3min 12sec, MB pro 2016, 16Gb RAM)

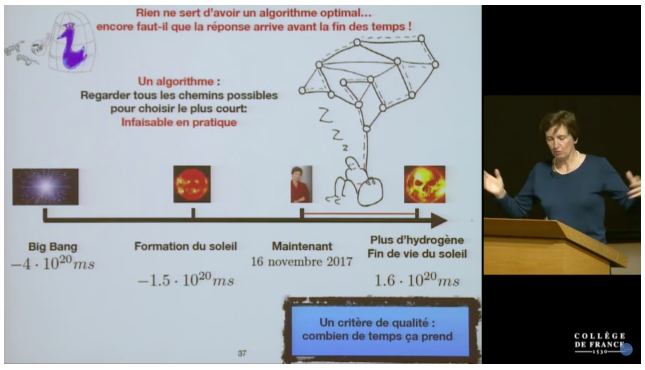


Problème 7 : TSP

« Problème fil rouge » UE TAP de L3, TSP.

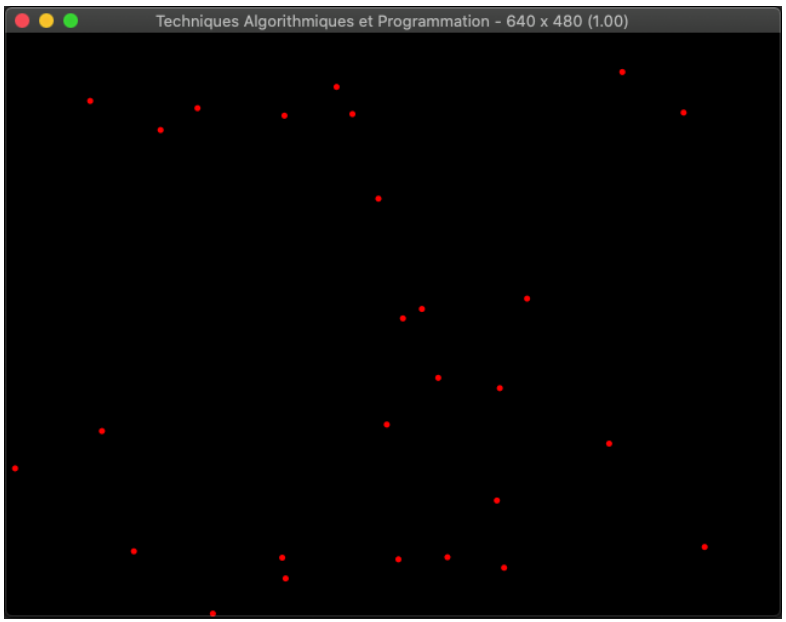


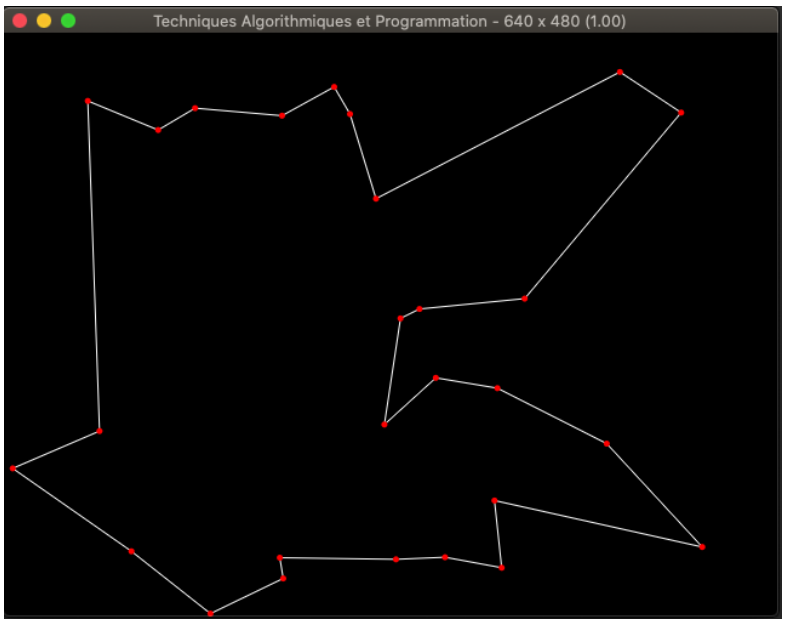
©XKCD



Claire Mathieu @ Collège de France

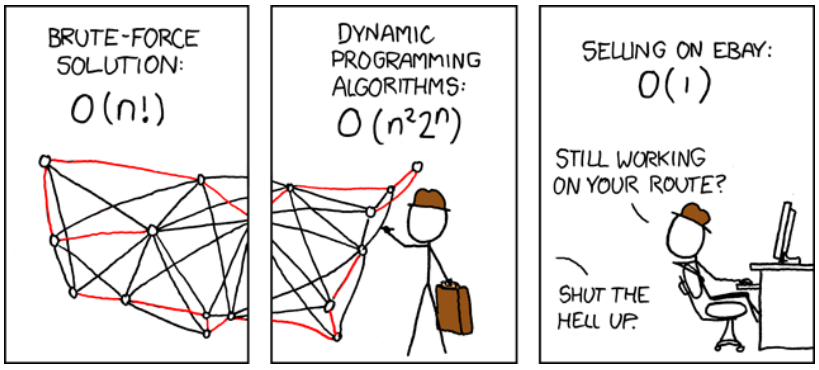
27 villes, soit **27 ! tournées** (3min 12sec, MB pro 2016, 16Gb RAM)



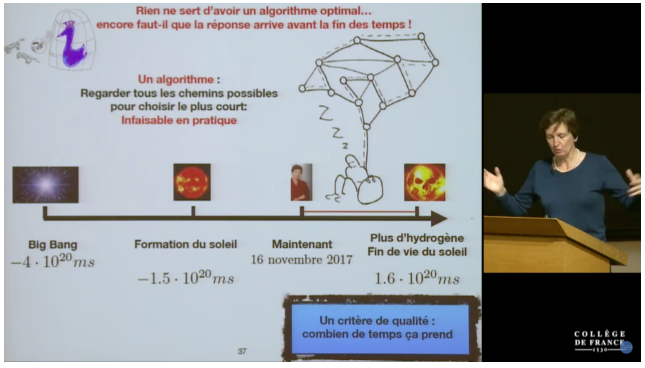


Problème 7 : TSP

« Problème fil rouge » UE TAP de L3, TSP.

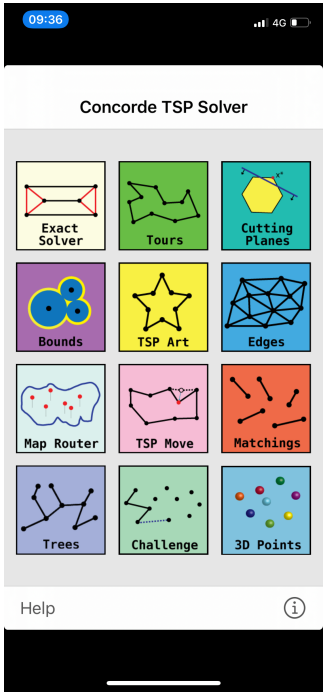


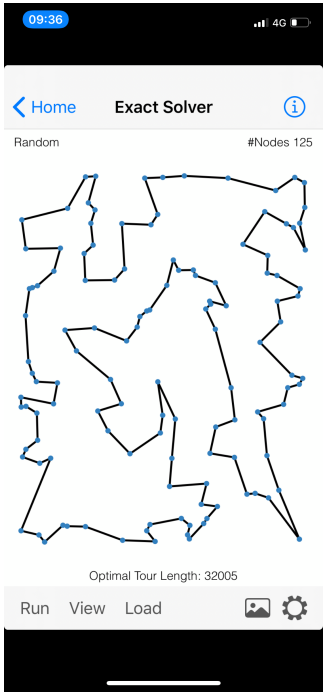
©XKCD



Claire Mathieu @ Collège de France

Jusqu'à 500 villes, résolution exacte, souvent rapide (mais pas toujours) :





Exemples de problèmes de décision (2)

Sudoku, Eternity, 3-Colorabilité, SAT et TSP : 

Exemples de problèmes de décision (2)

Sudoku, Eternity, 3-Colorabilité, SAT et TSP : 

Pour ces problèmes, on connaît des algorithmes ...

Exemples de problèmes de décision (2)

Sudoku, Eternity, 3-Colorabilité, SAT et TSP : 😕

Pour ces problèmes, on connaît des algorithmes … pouvant être très lents : dans le cas le pire, ils ne font pas bien mieux qu'une recherche exhaustive.

Exemples de problèmes de décision (2)

Sudoku, Eternity, 3-Colorabilité, SAT et TSP : 😕

Pour ces problèmes, on connaît des algorithmes ... pouvant être très lents : dans le cas le pire, ils ne font pas bien mieux qu'une recherche exhaustive.

Or, espace de solutions possibles ici **exponentiel** en la taille de l'entrée.

Question

- Nombre de 3-colorations pour un graphe à 60 sommets ?
- Nombre de trajets TSP distincts passant par 30 villes ?

À comparer avec le nombre de nanosecondes depuis le big-bang.

Exemples de problèmes de décision (3)

Problème 8 (Tschisla)

Instance Des entiers $n, k > 0$ et un chiffre $c \neq 0$.

Question Existe-t-il une expression,

- sur les opérateurs $+$, $-$, $\times$, $/$, $\sqrt{}$, $!$,
- utilisant au plus k fois le chiffre c,
- et dont la valeur est n ?

Exemples de problèmes de décision (3)

Problème 8 (Tschisla)

Instance Des entiers $n, k > 0$ et un chiffre $c \neq 0$.

Question Existe-t-il une expression,

- sur les opérateurs $+$, $-$, $\times$, $/$, $\sqrt{}$, $!$,
- utilisant au plus k fois le chiffre c,
- et dont la valeur est n ?

Problème 9 (Skolem)

I. Une suite récurrente à coefficients entiers, donnée par des entiers a_i, b_i $(0 \leqslant i < k)$ et

$$\begin{cases} u_0 = a_0, \ \ldots, \ u_{k-1} = a_{k-1} \\ u_{n+k} = b_{k-1} u_{n+k-1} + \cdots + b_0 u_n \end{cases}$$

Q. Existe-t-il n tel que $u_n = 0$?

Exemples de problèmes de décision (3)

Problème 8 (Tschisla)

Instance : Des entiers $n, k > 0$ et un chiffre $c \neq 0$.

Question : Existe-t-il une expression,

- sur les opérateurs $+$, $-$, $\times$, $/$, $\sqrt{}$, $!$,
- utilisant au plus k fois le chiffre c,
- et dont la valeur est n ?

Problème 9 (Skolem)

I. Une suite récurrente à coefficients entiers, donnée par des entiers a_i, b_i $(0 \leqslant i < k)$ et

$$\begin{cases} u_0 = a_0, \ \ldots, \ u_{k-1} = a_{k-1} \\ u_{n+k} = b_{k-1} u_{n+k-1} + \cdots + b_0 u_n \end{cases}$$

Q. Existe-t-il n tel que $u_n = 0$?

Pour ces deux problèmes, **on ne connaît pas d'algorithme**.

Exemples de problèmes de décision (4)

Problème 10

Instance Une équation Diophantienne à **9 variables**.

Question L'équation a-t-elle une solution dans $\mathbb{Z}$?

Exemples de problèmes de décision (4)

Problème 10

Instance : Une équation Diophantienne à **9 variables**.

Question : L'équation a-t-elle une solution dans $\mathbb{Z}$?

Problèmes 11, 12

I. Un nombre fini de **types** de pièces de Tetris/Wang.

Q. Peut-on paver le plan avec ces types de pièces ?

Exemples de problèmes de décision (4)

Problème 10

Instance : Une équation Diophantienne à **9 variables**.

Question : L'équation a-t-elle une solution dans $\mathbb{Z}$?

Problèmes 11, 12

I. Un nombre fini de **types** de pièces de Tetris/Wang.

Q. Peut-on paver le plan avec ces types de pièces ?

Problème 13 (GCP)

I. Un entier $m > 0$, des rationnels $a_0, \ldots, a_{m-1}$, $b_0, \ldots, b_{m-1}$, et une fonction $f : \mathbb{N} \to \mathbb{N}$ définie par cas : $f(x) = a_k x + b_k$ si $x \equiv k \pmod{m}$.

Q. Pour tout n, existe-t-il k tel que $f^k(n) = 1$?

Exemples de problèmes de décision (4)

Problème 10
- Instance : Une équation Diophantienne à **9 variables**.
- Question : L'équation a-t-elle une solution dans $\mathbb{Z}$?

Problèmes 11, 12
- I. Un nombre fini de **types** de pièces de Tetris/Wang.
- Q. Peut-on paver le plan avec ces types de pièces ?

Problème 13 (GCP)
- I. Un entier $m > 0$, des rationnels $a_0, \ldots, a_{m-1}$, $b_0, \ldots, b_{m-1}$, et une fonction $f : \mathbb{N} \to \mathbb{N}$ définie par cas : $f(x) = a_k x + b_k$ si $x \equiv k \pmod{m}$.
- Q. Pour tout n, existe-t-il k tel que $f^k(n) = 1$?

Problème 14
- I. Un **programme** P et une entrée x.
- Q. Le programme P s'arrête-t-il sur l'entrée x ?

Exemples de problèmes de décision (4)

Problème 10
- Instance : Une équation Diophantienne à **9 variables**.
- Question : L'équation a-t-elle une solution dans $\mathbb{Z}$?

Problèmes 11, 12
- I. Un nombre fini de **types** de pièces de Tetris/Wang.
- Q. Peut-on paver le plan avec ces types de pièces ?

Problème 13 (GCP)
- I. Un entier $m > 0$, des rationnels $a_0, \ldots, a_{m-1}$, $b_0, \ldots, b_{m-1}$, et une fonction $f : \mathbb{N} \to \mathbb{N}$ définie par cas : $f(x) = a_k x + b_k$ si $x \equiv k \pmod{m}$.
- Q. Pour tout n, existe-t-il k tel que $f^k(n) = 1$?

Problème 14
- I. Un **programme** P et une entrée x.
- Q. Le programme P s'arrête-t-il sur l'entrée x ?

Bad news...

Pour ces problèmes, on ne connaît pas d'algorithme ...

Exemples de problèmes de décision (4)

Problème 10
I. Une équation Diophantienne à **9 variables**.
Q. L'équation a-t-elle une solution dans $\mathbb{Z}$?

Problèmes 11, 12
I. Un nombre fini de **types** de pièces de Tetris/Wang.
Q. Peut-on paver le plan avec ces types de pièces ?

Problème 13 (GCP)
I. Un entier $m > 0$, des rationnels $a_0, \ldots, a_{m-1}$, $b_0, \ldots, b_{m-1}$, et une fonction $f : \mathbb{N} \to \mathbb{N}$ définie par cas : $f(x) = a_k x + b_k$ si $x \equiv k \pmod{m}$.
Q. Pour tout n, existe-t-il k tel que $f^k(n) = 1$?

Problème 14
I. Un **programme** P et une entrée x.
Q. Le programme P s'arrête-t-il sur l'entrée x ?

Bad news...

Pour ces problèmes, on ne connaît pas d'algorithme ...

Pire : on sait qu'**il n'en existe pas !**

Statut des problèmes d'informatique

- On connaît un algorithme qui résout le problème de façon efficace (en temps polynomial par rapport à la **taille de l'entrée**).
  Ce n'est pas du tout évident pour le problème de primalité.

Statut des problèmes d'informatique

- On connaît un algorithme qui résout le problème de façon efficace (en temps polynomial par rapport à la **taille de l'entrée**).
  Ce n'est pas du tout évident pour le problème de primalité.

- On connaît un algorithme, mais le meilleur connu fait un nombre exponentiel d'opérations par rapport à la **taille de l'entrée**.

Statut des problèmes d'informatique

- On connaît un algorithme qui résout le problème de façon efficace (en temps polynomial par rapport à la **taille de l'entrée**).
  Ce n'est pas du tout évident pour le problème de primalité.

- On connaît un algorithme, mais le meilleur connu fait un nombre exponentiel d'opérations par rapport à la **taille de l'entrée**.

- ? On ne connaît pas d'algorithme qui résout le problème, et on ne sait pas s'il en existe.

Statut des problèmes d'informatique

- On connaît un algorithme qui résout le problème de façon efficace (en temps polynomial par rapport à la **taille de l'entrée**).
  Ce n'est pas du tout évident pour le problème de primalité.

- On connaît un algorithme, mais le meilleur connu fait un nombre exponentiel d'opérations par rapport à la **taille de l'entrée**.

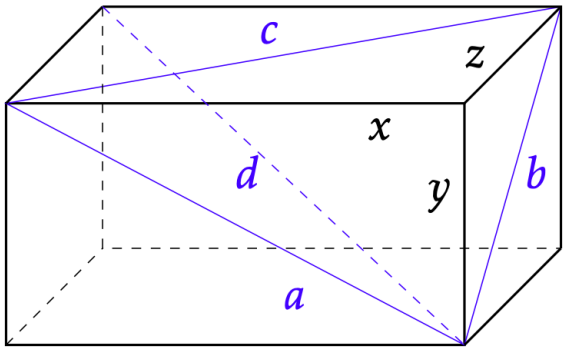
- ? On ne connaît pas d'algorithme qui résout le problème, et on ne sait pas s'il en existe.

- On **sait** qu'il **n'existe pas** d'algorithme pour ce problème (avec notre compréhension actuelle de ce qu'est un algorithme).

Problème 10 : équations Diophantiennes

Exemple, brique parfaite d'Euler (dessin issu du polycopié de Cyril Gavoille).

Existe-t-il un parallélépipède aux dimensions indiquées **entières** ?

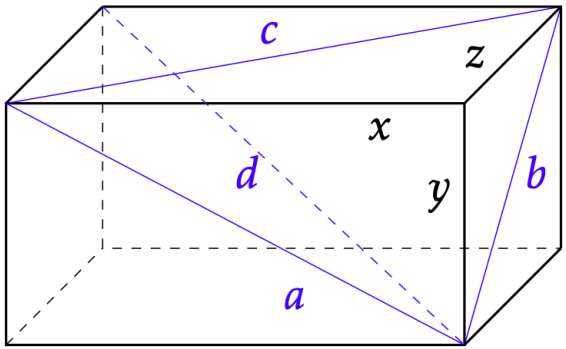


$$\begin{cases} a^2 = x^2 + y^2 \\ b^2 = y^2 + z^2 \\ c^2 = z^2 + x^2 \\ d^2 = x^2 + y^2 + z^2 \end{cases}$$

Problème 10 : équations Diophantiennes

Exemple, brique parfaite d'Euler (dessin issu du polycopié de Cyril Gavoille).

Existe-t-il un parallélépipède aux dimensions indiquées **entières** ?



$$\begin{cases} a^2 = x^2 + y^2 \\ b^2 = y^2 + z^2 \\ c^2 = z^2 + x^2 \\ d^2 = x^2 + y^2 + z^2 \end{cases}$$

Autrement dit, l'équation Diophantienne

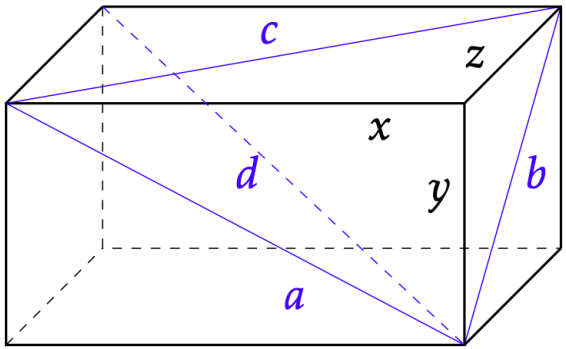
$$(a^2-x^2-y^2)^2+(b^2-y^2-z^2)^2+(c^2-z^2-x^2)^2+(d^2-x^2-y^2-z^2)^2 = 0$$

a-t-elle une solution pour $a, b, c, d, x, y, z \in \mathbb{N}^*$?

Problème 10 : équations Diophantiennes

Exemple, brique parfaite d'Euler (dessin issu du polycopié de Cyril Gavoille).

Existe-t-il un parallélépipède aux dimensions indiquées **entières** ?



$$\begin{cases} a^2 = x^2 + y^2 \\ b^2 = y^2 + z^2 \\ c^2 = z^2 + x^2 \\ d^2 = x^2 + y^2 + z^2 \end{cases}$$

Autrement dit, l'équation Diophantienne

$$(a^2-x^2-y^2)^2+(b^2-y^2-z^2)^2+(c^2-z^2-x^2)^2+(d^2-x^2-y^2-z^2)^2 = 0$$

a-t-elle une solution pour $a, b, c, d, x, y, z \in \mathbb{N}^*$?

C'est une question ouverte !

Problème 10 : équations Diophantiennes

Autres exemples

- $x^3 + y^3 + z^3 = 43$?

Problème 10 : équations Diophantiennes

Autres exemples

- $x^3 + y^3 + z^3 = 43$?

Facile : $x = y = 2, z = 3$.

Problème 10 : équations Diophantiennes

Autres exemples

- $x^3 + y^3 + z^3 = 43$?
- $x^3 + y^3 + z^3 = 42$?

Facile : $x = y = 2, z = 3$.

Problème 10 : équations Diophantiennes

Autres exemples

- $x^3 + y^3 + z^3 = 43$?
- $x^3 + y^3 + z^3 = 42$?

Facile : $x = y = 2, z = 3$.

```
>>> x = -8053873881207597 4
>>> y = 8043575814581751 5
>>> z = 1260212329733563 1
>>> print(x**3 + y**3 + z**3)
42
```

Problème 10 : équations Diophantiennes

Autres exemples

- $x^3 + y^3 + z^3 = 43$? Facile : $x = y = 2, z = 3$.
- $x^3 + y^3 + z^3 = 42$? Booker, Sutherland, 9/2019.

```
>>> x = -80538738812075974
>>> y = 80435758145817515
>>> z = 12602123297335631
>>> print(x**3 + y**3 + z**3)
42
```

- Calcul fait (après analyse) sur Charity Engine.
- Plus de un million d'heures de temps CPU (soit $\approx$ **114 années** !).

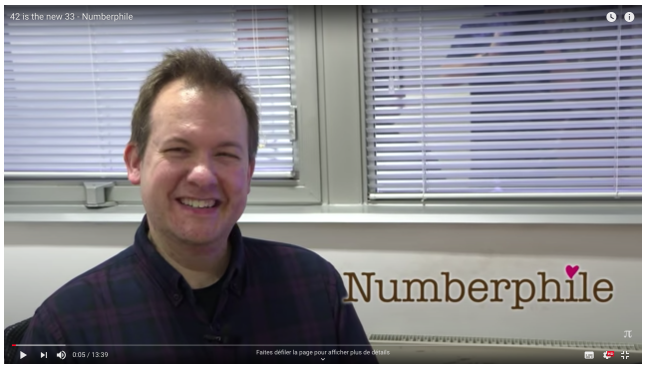
Problème 10 : équations Diophantiennes

Autres exemples

- $x^3 + y^3 + z^3 = 43$? Facile : $x = y = 2, z = 3$.
- $x^3 + y^3 + z^3 = 42$? Booker, Sutherland, 9/2019.

```
>>> x = -80538738812075974
>>> y = 80435758145817515
>>> z = 12602123297335631
>>> print(x**3 + y**3 + z**3)
42
```

- Calcul fait (après analyse) sur Charity Engine.
- Plus de un million d'heures de temps CPU (soit $\approx$ **114 années** !).
- Ce n'est **pas** une simple triple boucle ! Entier précédent, 33, mars 2019.



A. R. Booker, 33

A. R. Booker, 42

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

$$x^7 = 16x^4 - 4x^3 + 5x + 2023$$

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

$$x^7 = 16x^4 - 4x^3 + 5x + 2023$$

 Solutions divisent $2023 \Rightarrow$ espace de recherche fini

↪ Résolution automatique d'équations Diophantiennes à 1 variable.

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

$$x^7y^2 + 16xyz^4 = 4xz^5 + 5xyz + 2023 \qquad ????$$

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

$$x^7y^2 + 16xyz^4 = 4xz^5 + 5xyz + 2023 \qquad ????$$

Hilbert demande pas de résoudre une équation particulière, mais de trouver un **algorithme** qui les résoudrait toutes.

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

1928 **Hilbert** ***Entscheidungsproblem***. Peut-on mécaniser les maths (1er ordre) ?

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

1928 **Hilbert** ***Entscheidungsproblem***. Peut-on mécaniser les maths (1er ordre) ?

1931 **Gödel** publie ses théorèmes d'incomplétude.

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

1928 **Hilbert** ***Entscheidungsproblem***. Peut-on mécaniser les maths (1er ordre) ?

1931 **Gödel** publie ses théorèmes d'incomplétude.

1935 **Turing** formalise une définition de machine et de calcul.

1936 **Turing** prouve que le problème de l'arrêt est indécidable.

1936 **Church** exhibe un problème non résoluble par machine.

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

1928 **Hilbert** ***Entscheidungsproblem***. Peut-on mécaniser les maths (1er ordre) ?

1931 **Gödel** publie ses théorèmes d'incomplétude.

1935 **Turing** formalise une définition de machine et de calcul.

1936 **Turing** prouve que le problème de l'arrêt est indécidable.

1936 **Church** exhibe un problème non résoluble par machine.

1938 **Kleene** équivalence machines de Turing, λ-calcul, fonctions récursives

Historique rapide

1900 **Hilbert**, 10e problème : peut-on **décider**, de façon **mécanique**, si une équation Diophantienne a une solution en nombres entiers ?

1928 **Hilbert** ***Entscheidungsproblem***. Peut-on mécaniser les maths (1er ordre) ?

1931 **Gödel** publie ses théorèmes d'incomplétude.

1935 **Turing** formalise une définition de machine et de calcul.

1936 **Turing** prouve que le problème de l'arrêt est indécidable.

1936 **Church** exhibe un problème non résoluble par machine.

1938 **Kleene** équivalence machines de Turing, λ-calcul, fonctions récursives

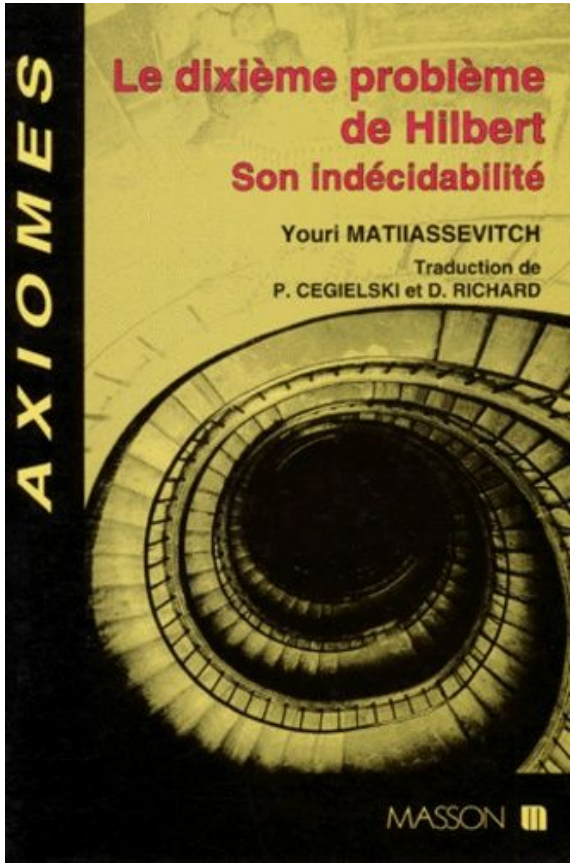
1947 **Post & Markov** prouvent qu'un problème posé par **Thue** en 1914 n'est pas résoluble mécaniquement.

1970 **Matiyasevich** répond **négativement** au 10e problème de Hilbert.

1971 **Cook & Levin** formalisent la notion de problème **NP-complet**.

Le 10e problème de Hilbert

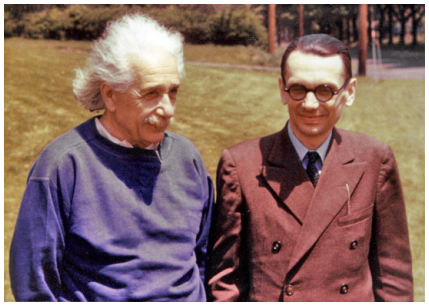
1900 : Hilbert – 1970 : Matiyasevich

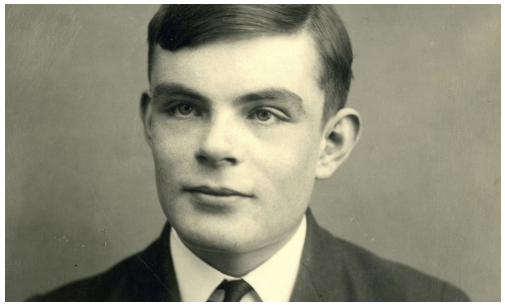


Problème indécidable = **sans algorithme pour le résoudre**

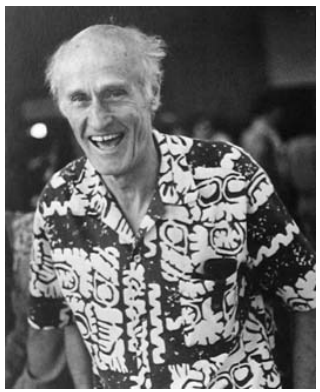
Portrait de famille (cliquez pour des détails)


Hilbert


Einstein, Gödel


Turing

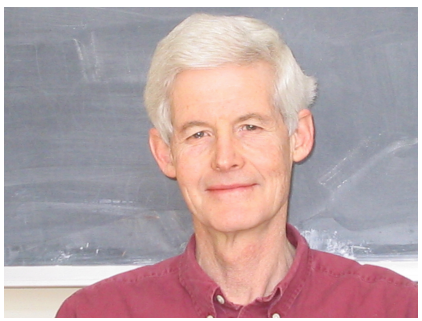

Church


Kleene


Post


Markov

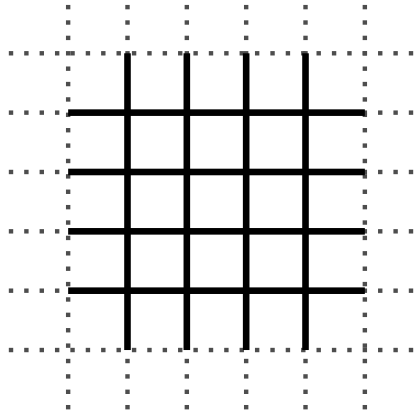

Matiyasevich


Cook


Levin

Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.



Problème du pavage par pièces de Tetris

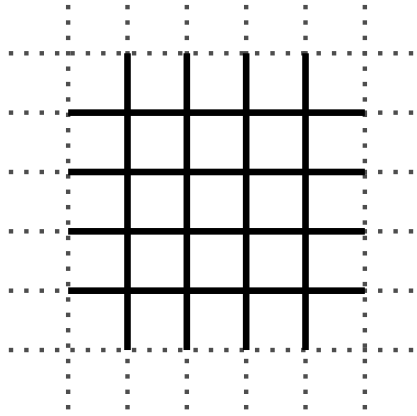
Instance Nombre fini de types de pièces, accolements de carrés 1×1.

Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(rotations permises, symétries axiales interdites)

Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.

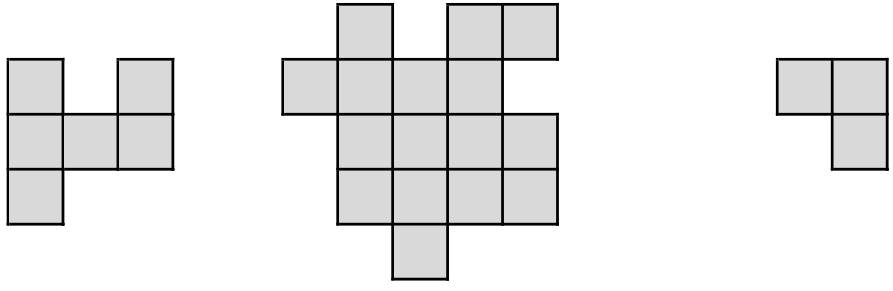


Problème du pavage par pièces de Tetris

Instance Nombre fini de types de pièces, accolements de carrés 1×1.

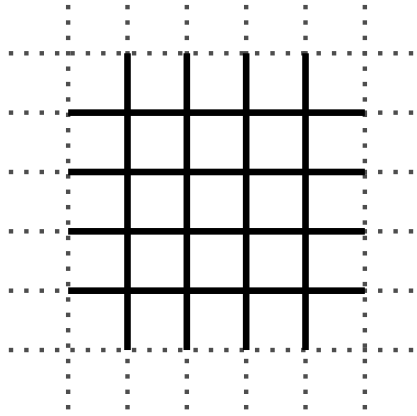
Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(rotations permises, symétries axiales interdites)



Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.

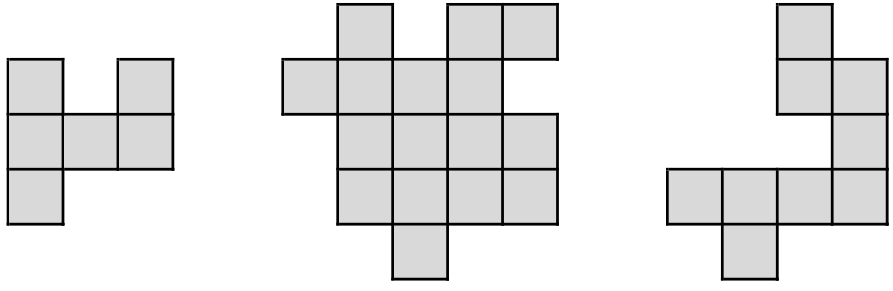


Problème du pavage par pièces de Tetris

Instance Nombre fini de types de pièces, accolements de carrés 1×1.

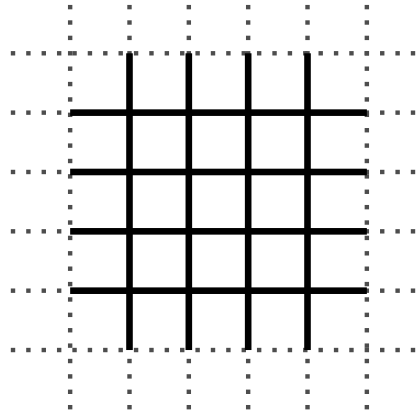
Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(rotations permises, symétries axiales interdites)



Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.

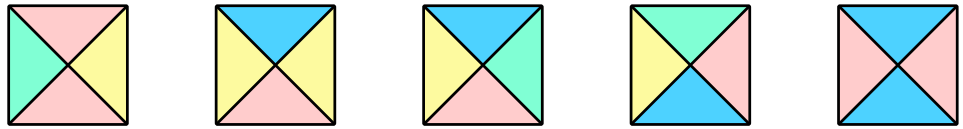


Problème du pavage par pièces de Wang

Instance Nombre fini de types de pièces 1×1 avec 4 couleurs.

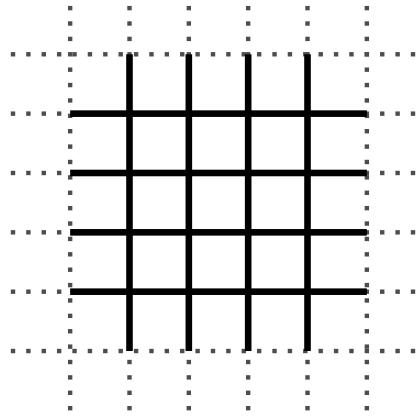
Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(rotations et symétries interdites)



Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.

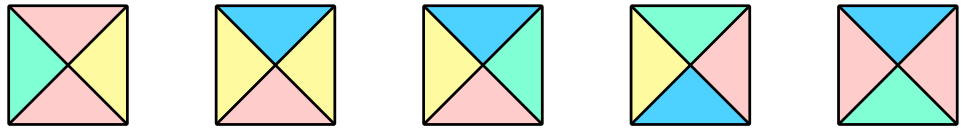


Problème du pavage par pièces de Wang

Instance Nombre fini de types de pièces 1×1 avec 4 couleurs.

Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(rotations et symétries interdites)



Problème 13 : Collatz généralisé (GCP)

Instance Un entier $m > 0$, des rationnels $a_0, \ldots, a_{m-1}, b_0, \ldots, b_{m-1}$ tels que la fonction définie par cas :

$$f(x) = a_k x + b_k \qquad \text{si } x \equiv k \pmod{m}$$

envoie tout entier sur un entier.

Question Pour tout $n \in \mathbb{N}$, existe-t-il k tel que $f^k(n) = 1$?

Problème 13 : Collatz généralisé (GCP)

Instance Un entier $m > 0$, des rationnels $a_0, \ldots, a_{m-1}, b_0, \ldots, b_{m-1}$ tels que la fonction définie par cas :

$$f(x) = a_k x + b_k \qquad \text{si } x \equiv k \pmod{m}$$

envoie tout entier sur un entier.

Question Pour tout $n \in \mathbb{N}$, existe-t-il k tel que $f^k(n) = 1$?

Exemple d'instance

Les valeurs $m = 2$, $a_0 = 1/2$, $b_0 = 0$, $a_1 = 3$, $b_1 = 1$ donnent la fonction

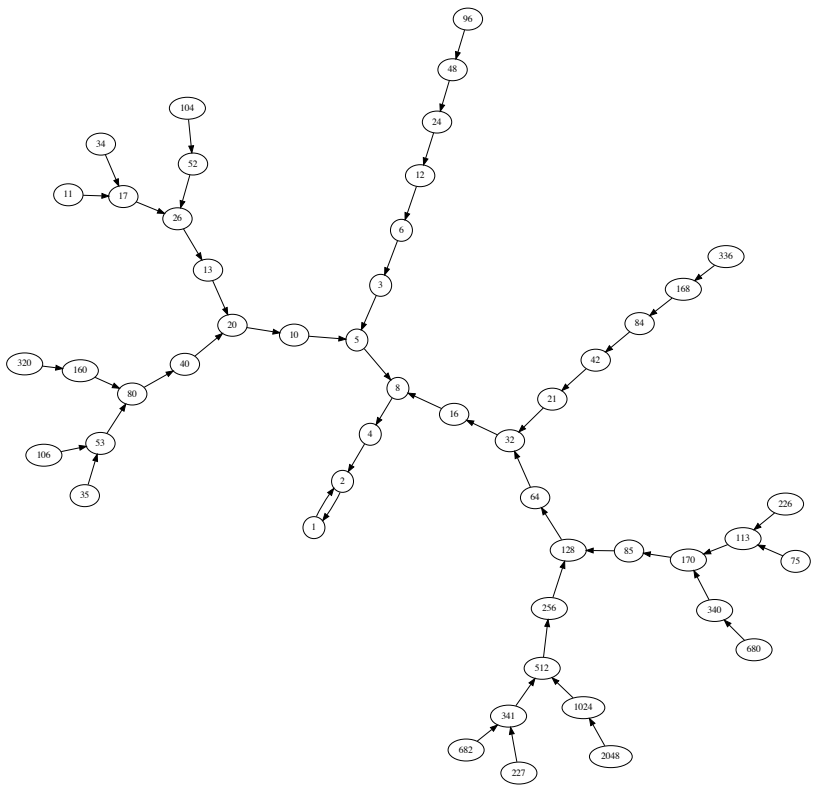
$$f(x) = \begin{cases} x/2 & \text{si } x \text{ est pair,} \\ 3x + 1 & \text{si } x \text{ est impair.} \end{cases}$$

Problème 13 : Collatz généralisé (GCP)

$$f(x) = \begin{cases} x/2 & \text{si } x \text{ est pair,} \\ 3x+1 & \text{si } x \text{ est impair.} \end{cases}$$

Visualisation partielle avec gengraph

```
$ gengraph syracuse 50 -label 1 -visu
$ open g.pdf
```



Problème 14 : problème de l'arrêt

Problème de l'arrêt HALT.

Instance Un **programme** P et une entrée x.

Question Le programme P s'arrête-t-il sur l'entrée x ?

Problème 14 : problème de l'arrêt

Problème de l'arrêt HALT.

Instance Un **programme** P et une entrée x.

Question Le programme P s'arrête-t-il sur l'entrée x ?

Un programme se représente simplement par une chaîne de caractères

$\Longrightarrow$ un problème peut avoir un programme comme instance.

Problème 14 : problème de l'arrêt

Problème de l'arrêt HALT.

Instance Un **programme** P et une entrée x.

Question Le programme P s'arrête-t-il sur l'entrée x ?

Un programme se représente simplement par une chaîne de caractères

$\Longrightarrow$ un problème peut avoir un programme comme instance.

C'est le **premier problème montré indécidable (par Turing)**.

Idée de preuve dans la prochaine partie.

Plan

1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

3. Réductions et NP-complétude

4. Complexité des problèmes – P $\stackrel{?}{=}$ NP

Dans cette partie

1. Indécidabilité du problème de l'arrêt, HALT.
2. Machines de Turing.

Définition clé de cette partie

Définition

Un problème est **indécidable** s'il n'y a **aucun algorithme** pour le résoudre.

Un problème fondamental : l'arrêt (HALT)

Remarque préliminaire

Si on exécute un programme sur une entrée, 2 comportements possibles :

- Le programme s'arrête au bout d'un temps fini, ou
- Le programme ne s'arrête jamais (on dit dans ce cas qu'il **boucle**).

Un problème fondamental : l'arrêt (HALT)

Remarque préliminaire

Si on exécute un programme sur une entrée, 2 comportements possibles :

- Le programme s'arrête au bout d'un temps fini, ou
- Le programme ne s'arrête jamais (on dit dans ce cas qu'il **boucle**).

Problème de l'arrêt HALT

Entrée Un programme P et une valeur d'entrée x de ce programme.

Question Le programme P s'arrête-t-il quand son entrée est x ?

Un problème fondamental : l'arrêt (HALT)

Remarque préliminaire

Si on exécute un programme sur une entrée, 2 comportements possibles :

- Le programme s'arrête au bout d'un temps fini, ou
- Le programme ne s'arrête jamais (on dit dans ce cas qu'il **boucle**).

Problème de l'arrêt HALT

Entrée Un programme P et une valeur d'entrée x de ce programme.

Question Le programme P s'arrête-t-il quand son entrée est x ?

Indécidabilité du problème de l'arrêt (Turing)

Aucun algorithme ne peut résoudre le problème HALT.

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Supposons qu'il existe un algorithme, `halt`, qui

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Supposons qu'il existe un algorithme, `halt`, qui

- prend en argument un programme P et un paramètre x de P,
- et teste l'arrêt de P lancé sur x.

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Supposons qu'il existe un algorithme, `halt`, qui

- prend en argument un programme `P` et un paramètre `x` de `P`,
- et teste l'arrêt de `P` lancé sur `x`.

```
def halt(P, x):
    # Fonction hypothétique supposée prendre en argument
    # le texte d'un programme, P (une chaîne de caractères)
    # et un argument x de P (une chaîne également).

    # Renvoie True si P s'arrête sur x
    #         False sinon
```

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Supposons qu'il existe un algorithme, `halt`, qui

- prend en argument un programme P et un paramètre x de P,
- et teste l'arrêt de P lancé sur x.

```
def halt(P, x):
    # Fonction hypothétique supposée prendre en argument
    # le texte d'un programme, P (une chaîne de caractères)
    # et un argument x de P (une chaîne également).

    # Renvoie True si P s'arrête sur x
    #         False sinon
```

On va montrer que c'est impossible en obtenant une contradiction logique.

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Hypothèse : il existe un algorithme `halt` testant si `P` s'arrête sur `x`.

```
def halt(P, x):
    # Renvoie True si P s'arrête sur x
    #         False sinon
```

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Hypothèse : il existe un algorithme `halt` testant si `P` s'arrête sur `x`.

```
def halt(P, x):
    # Renvoie True si P s'arrête sur x
    #         False sinon
```

Considérons le programme

```
def contradition(P):
    if halt(P, P):
        while(True):
            print("Je boucle")
    else:
        print("Je m'arrête")
```

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Hypothèse : il existe un algorithme `halt` testant si `P` s'arrête sur `x`.

```
def halt(P, x):
    # Renvoie True si P s'arrête sur x
    #         False sinon
```

Considérons le programme

```
def contradition(P):
    if halt(P, P):
        while(True):
            print("Je boucle")
    else:
        print("Je m'arrête")
```

Quand on lance **`contradiction(contradiction)`**, 2 cas possibles.

1er cas : Cet appel s'arrête.
Alors `halt(contradiction,contradiction)` renvoie `True`
Donc l'appel passe dans la boucle `while` et ne s'arrête pas.

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Hypothèse : il existe un algorithme `halt` testant si `P` s'arrête sur `x`.

```
def halt(P, x):
    # Renvoie True si P s'arrête sur x
    #         False sinon
```

Considérons le programme

```
def contradition(P):
    if halt(P, P):
        while(True):
            print("Je boucle")
    else:
        print("Je m'arrête")
```

Quand on lance **`contradiction(contradiction)`**, 2 cas possibles.

2e cas : Cet appel ne s'arrête pas.
Alors `halt(contradiction,contradiction)` renvoie `False`
Donc l'appel ne passe pas dans la boucle `while` et s'arrête.

Indécidabilité du problème de l'arrêt (HALT)

Démonstration par l'absurde

Hypothèse : il existe un algorithme `halt` testant si `P` s'arrête sur `x`.

```
def halt(P, x):
    # Renvoie True si P s'arrête sur x
    #         False sinon
```

Considérons le programme

```
def contradition(P):
    if halt(P, P):
        while(True):
            print("Je boucle")
    else:
        print("Je m'arrête")
```

Quand on lance **`contradiction(contradiction)`**, 2 cas possibles.

- Dans les 2 cas, on aboutit à une contradiction.
- Donc l'algorithme `halt` **ne peut pas exister**.

Recap : indécidabilité, problème de l'arrêt

Un problème est indécidable si aucun algorithme ne le résout.

Turing, 1935

Le problème de l'arrêt est indécidable.

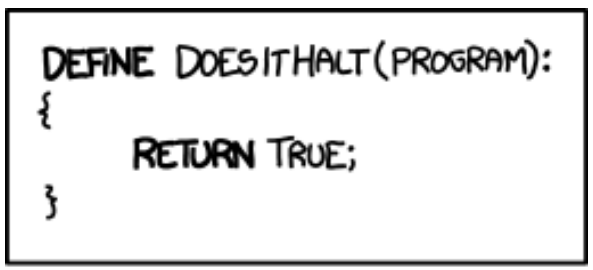
Recap : indécidabilité, problème de l'arrêt

Un problème est indécidable si aucun algorithme ne le résout.

Turing, 1935

Le problème de l'arrêt est indécidable.

Le point de vue d'XKCD :



THE BIG PICTURE SOLUTION
TO THE HALTING PROBLEM

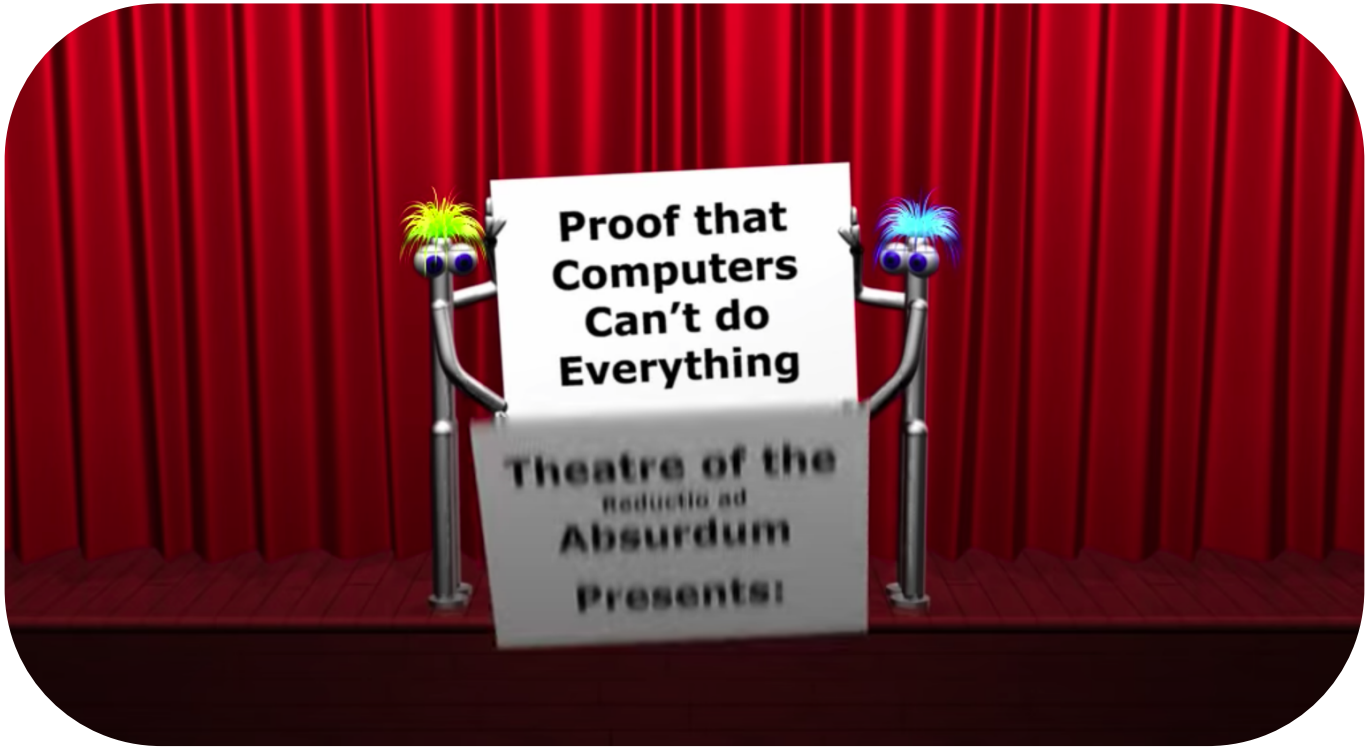
Recap : indécidabilité, problème de l'arrêt

Un problème est indécidable si aucun algorithme ne le résout.

Turing, 1935

Le problème de l'arrêt est indécidable.

La même preuve en vidéo (*click it!*) :



Deux questions “followup”

1. Ce problème est-il une exception, une anomalie ?

   i.e., il n’y a peut-être que **ce** problème spécifique qui est indécidable.

Deux questions “followup”

1. Ce problème est-il une exception, une anomalie ?

   i.e., il n’y a peut-être que **ce** problème spécifique qui est indécidable.

2. Le langage dans lequel on a fait la preuve compte-t-il ?

   i.e., pourrait-on programmer `halt` en C ? Java ? OCaml ?

Deux questions “followup”

1. Ce problème est-il une exception, une anomalie ?

   i.e., il n’y a peut-être que **ce** problème spécifique qui est indécidable.

2. Le langage dans lequel on a fait la preuve compte-t-il ?

   i.e., pourrait-on programmer `halt` en C ? Java ? OCaml ?

1re question : HALT est-il une anomalie ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?
- termine-t-il sur **toute** entrée ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?
- termine-t-il sur **toute** entrée ?
- atteint-il sa 42e instruction ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?
- termine-t-il sur **toute** entrée ?
- atteint-il sa 42^{e} instruction ?
- affecte-t-il 42 à la variable v ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?
- termine-t-il sur **toute** entrée ?
- atteint-il sa 42^{e} instruction ?
- affecte-t-il 42 à la variable v ?
- effectue-t-il une division par 0 ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?
- termine-t-il sur **toute** entrée ?
- atteint-il sa 42e instruction ?
- affecte-t-il 42 à la variable v ?
- effectue-t-il une division par 0 ?
- écrit-il « quarante-deux » ?

1re question : HALT est-il une anomalie ?

Non ! C'est même le contraire...

Les problèmes suivants sont tous indécidables

Étant donné un programme comme instance :

- termine-t-il sur **une entrée au moins** ?
- termine-t-il sur **toute** entrée ?
- atteint-il sa 42^{e} instruction ?
- affecte-t-il 42 à la variable v ?
- effectue-t-il une division par 0 ?
- écrit-il « quarante-deux » ?

Nombreux exemples hors « info-informatique ».

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

Pour le montrer, on utilise l'indécidabilité du problème de l'arrêt.

Problème de l'arrêt HALT (déjà connu indécidable)

Instance Un programme P et une valeur x du paramètre de P.

Question Le programme P s'arrête-t-il quand son paramètre vaut x ?

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

Pour le montrer, on utilise l'indécidabilité du problème de l'arrêt.

Problème de l'arrêt HALT (déjà connu indécidable)

Instance Un programme P et une valeur x du paramètre de P.

Question Le programme P s'arrête-t-il quand son paramètre vaut x ?

Réduction

Idée clé : un algorithme résolvant « Affectation à 42 » peut être utilisé pour construire un algorithme résolvant HALT.

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

À partir d'une instance P,x de **HALT**,

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

À partir d'une instance P,x de **HALT**,
on construit une instance Q,x,v de **Affectation à 42** telle que

P s'arrête sur x $\iff$ Q sur l'entrée x affecte 42 à v.

Exemple : problème de l'affectation à 42

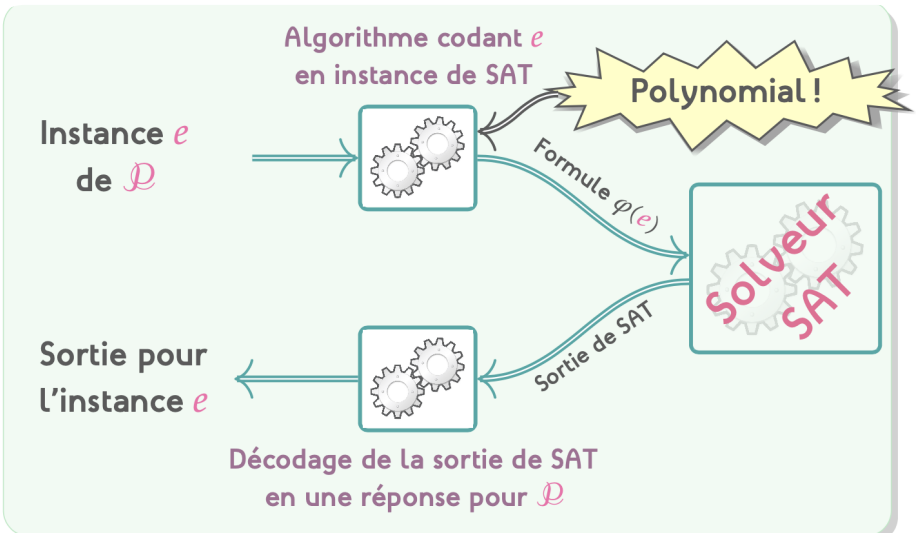
Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

À partir d'une instance P,x de **HALT**,
on construit une instance Q,x,v de **Affectation à 42** telle que

P s'arrête sur x $\iff$ Q sur l'entrée x affecte 42 à v.



Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

À partir d'une instance P,x de **HALT**,
on construit une instance Q,x,v de **Affectation à 42** telle que

P s'arrête sur x $\iff$ Q sur l'entrée x affecte 42 à v.

Construction : Q obtenu à partir de P en ajoutant une variable v et en précédant **chaque instruction** qui termine P par une affectation de v à 42.

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

À partir d'une instance P,x de **HALT**,
on construit une instance Q,x,v de **Affectation à 42** telle que

P s'arrête sur x $\iff$ Q sur l'entrée x affecte 42 à v.

Construction : Q obtenu à partir de P en ajoutant une variable v et en précédant **chaque instruction** qui termine P par une affectation de v à 42.

Dans le programme P

```
return 0
```

Exemple : problème de l'affectation à 42

Le problème « Affectation à 42 » est indécidable

Instance Un programme Q, une entrée x de Q et une variable v de Q.

Question Q lancé sur x affecte-t-il 42 à v ?

À partir d'une instance P,x de **HALT**,
on construit une instance Q,x,v de **Affectation à 42** telle que

P s'arrête sur x $\iff$ Q sur l'entrée x affecte 42 à v.

Construction : Q obtenu à partir de P en ajoutant une variable v et en précédant **chaque instruction** qui termine P par une affectation de v à 42.

Dans le programme P

```
return 0
```

À l'endroit correspondant de Q

```
v = 42
return 0
```

Les limites du calcul automatique

Théorème de Rice

Toute propriété de l'exécution des programmes est indécidable ou triviale.

Les limites du calcul automatique

Théorème de Rice

Toute propriété de l'exécution des programmes est indécidable ou triviale.

En clair : on ne peut pas programmer un « super compilateur » qui, à la super-compilation, prendrait en entrée un programme source et qui :

- Détecterait les instructions non atteintes,
- Détecterait les assertions fausses.
- ...

Les limites du calcul automatique

Théorème de Rice

Toute propriété de l'exécution des programmes est indécidable ou triviale.

En clair : on ne peut pas programmer un « super compilateur » qui, à la super-compilation, prendrait en entrée un programme source et qui :

- Détecterait les instructions non atteintes,
- Détecterait les assertions fausses.
- ...

Et avec une syntaxe simple ? sans types compliqués ? sans récursivité ?

Les limites du calcul automatique

Théorème de Rice

Toute propriété de l'exécution des programmes est indécidable ou triviale.

En clair : on ne peut pas programmer un « super compilateur » qui, à la super-compilation, prendrait en entrée un programme source et qui :

- Détecterait les instructions non atteintes,
- Détecterait les assertions fausses.
- ...

Et avec une syntaxe simple ? sans types compliqués ? sans récursivité ?

Le théorème de Rice est valide sur jeu d'instructions réduit

Même pour les programmes à 2 **variables entières**, et 2 **instructions** :

- `x += 1`
- `if (x == 0) goto p else x-- goto q`

Deux questions “followup”

1. Ce problème est-il une exception, une anomalie ?

   i.e., il n’y a peut-être que **ce** problème spécifique qui est indécidable.

NON

Deux questions “followup”

1. Ce problème est-il une exception, une anomalie ?

   i.e., il n’y a peut-être que **ce** problème spécifique qui est indécidable.

   NON

2. Le langage dans lequel on a fait la preuve compte-t-il ?

   i.e., pourrait-on programmer `halt` en C ? Java ? OCaml ?

Deux questions “followup”

1. Ce problème est-il une exception, une anomalie ?

   i.e., il n’y a peut-être que **ce** problème spécifique qui est indécidable.

   NON

2. Le langage dans lequel on a fait la preuve compte-t-il ?

   i.e., pourrait-on programmer `halt` en C ? Java ? OCaml ?

2e question : « puissance » des langages

2e question : « puissance » des langages

Pas de langage plus puissant qu'un autre

Toute fonction qui calcule, écrite dans n'importe quel langage actuel, se compile en un langage très simple : le langage des machines de Turing.

Machines de Turing

Abstractions mathématiques de programmes.

Machines de Turing

Abstractions mathématiques de programmes.

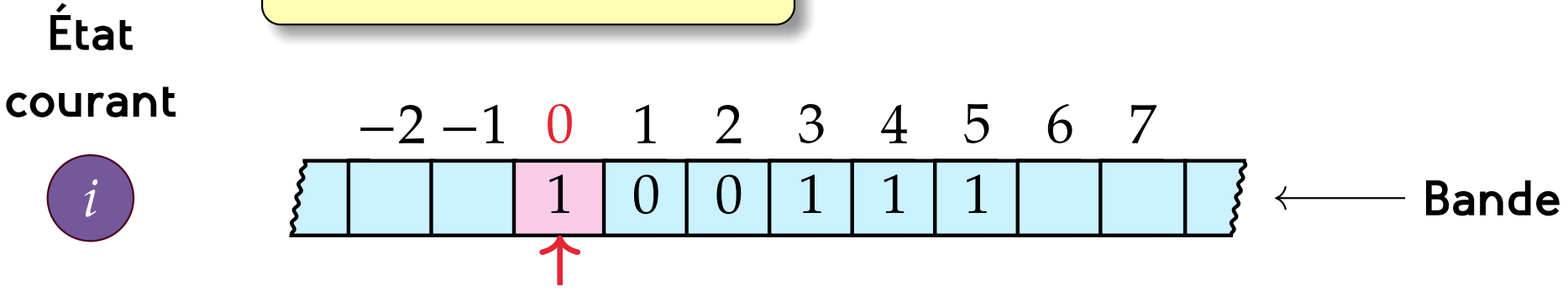
Composants d'une machine de Turing

- Une bande infinie à droite et à gauche faite de cases consécutives.
- Dans chaque case se trouve un symbole, éventuellement blanc $\square$.
- Une tête de lecture-écriture.
- Un nombre fini d'états.
- Un nombre fini de transitions, constituant le programme.

Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

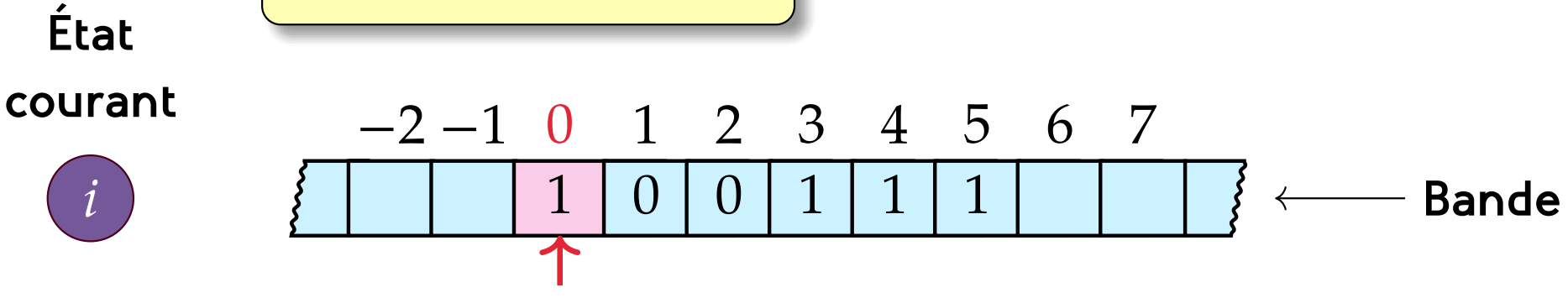
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

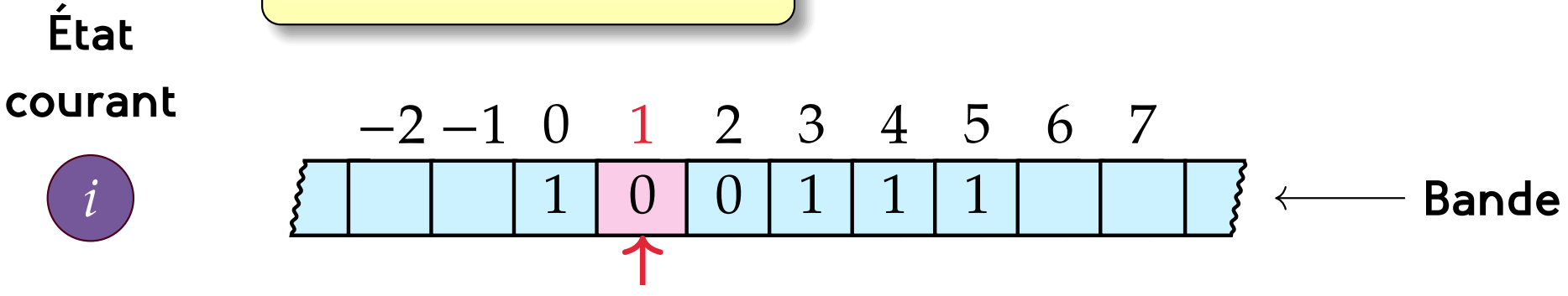
$$
\begin{aligned}
\delta(i,0) &= (i,0,\rightarrow)\\
\delta(i,1) &= (i,1,\rightarrow)\\
\delta(i,\square) &= (a,\square,\leftarrow)\\
\delta(a,0) &= (f,1,\leftarrow)\\
\delta(a,1) &= (a,0,\leftarrow)\\
\delta(a,\square) &= (f,1,\leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

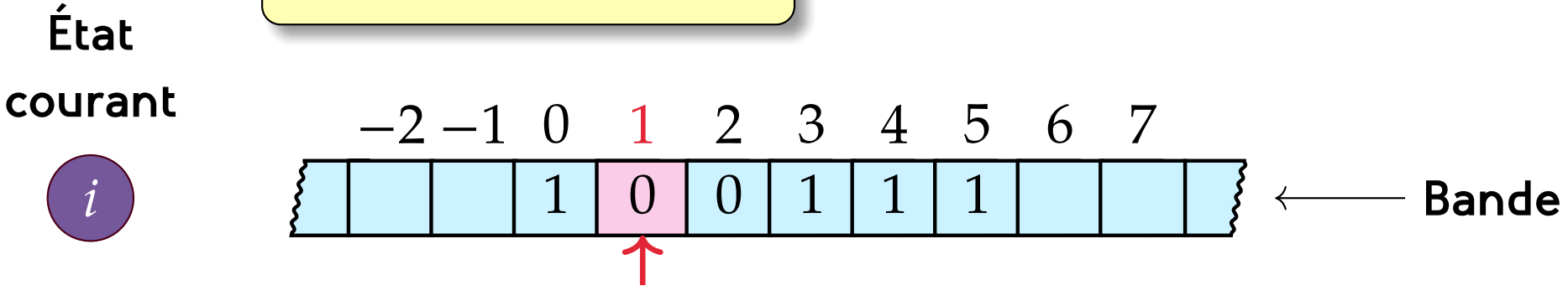
$$
\begin{aligned}
\delta(i,0) &= (i,0,\rightarrow)\\
\delta(i,1) &= (i,1,\rightarrow)\\
\delta(i,\square) &= (a,\square,\leftarrow)\\
\delta(a,0) &= (f,1,\leftarrow)\\
\delta(a,1) &= (a,0,\leftarrow)\\
\delta(a,\square) &= (f,1,\leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

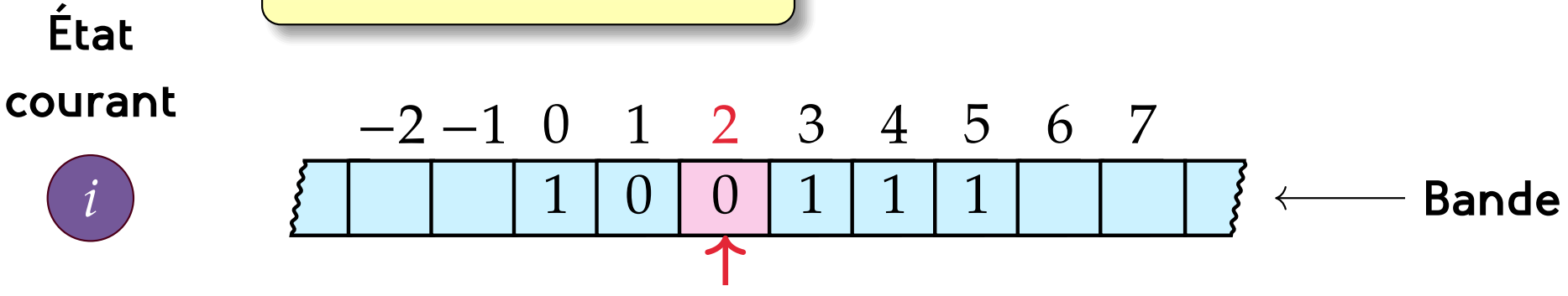
$$
\begin{aligned}
\delta(i,0) &= (i,0,\rightarrow)\\
\delta(i,1) &= (i,1,\rightarrow)\\
\delta(i,\square) &= (a,\square,\leftarrow)\\
\delta(a,0) &= (f,1,\leftarrow)\\
\delta(a,1) &= (a,0,\leftarrow)\\
\delta(a,\square) &= (f,1,\leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

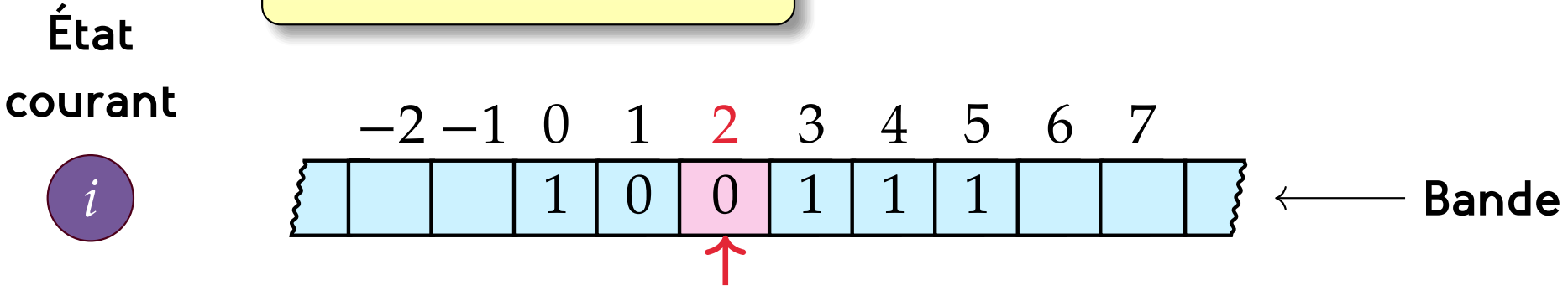
$$
\begin{aligned}
\delta(i,0) &= (i,0,\rightarrow)\\
\delta(i,1) &= (i,1,\rightarrow)\\
\delta(i,\square) &= (a,\square,\leftarrow)\\
\delta(a,0) &= (f,1,\leftarrow)\\
\delta(a,1) &= (a,0,\leftarrow)\\
\delta(a,\square) &= (f,1,\leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

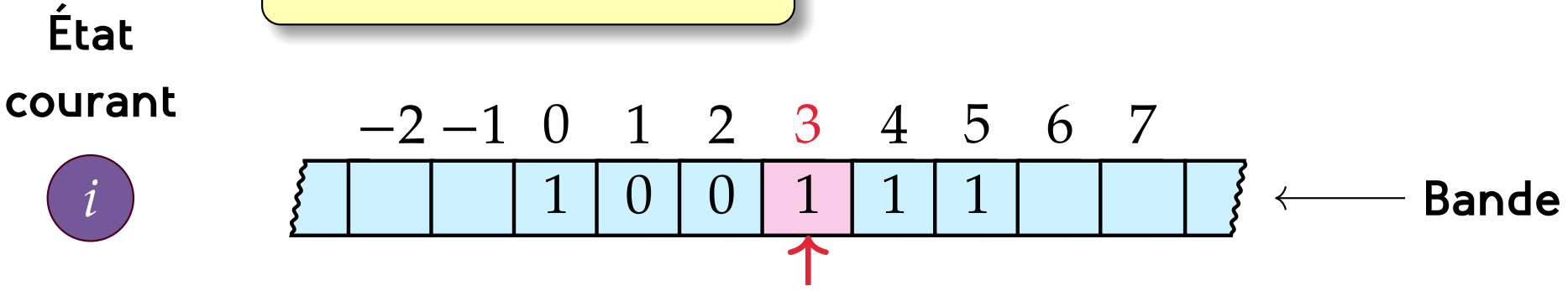
$$
\begin{aligned}
\delta(i,0) &= (i,0,\rightarrow)\\
\delta(i,1) &= (i,1,\rightarrow)\\
\delta(i,\square) &= (a,\square,\leftarrow)\\
\delta(a,0) &= (f,1,\leftarrow)\\
\delta(a,1) &= (a,0,\leftarrow)\\
\delta(a,\square) &= (f,1,\leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

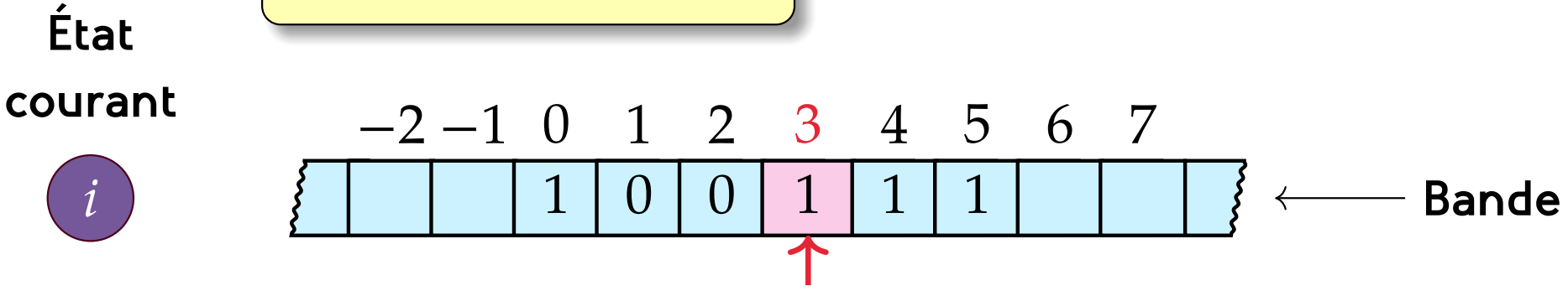
$$
\begin{aligned}
\delta(i,0) &= (i,0,\rightarrow)\\
\delta(i,1) &= (i,1,\rightarrow)\\
\delta(i,\square) &= (a,\square,\leftarrow)\\
\delta(a,0) &= (f,1,\leftarrow)\\
\delta(a,1) &= (a,0,\leftarrow)\\
\delta(a,\square) &= (f,1,\leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

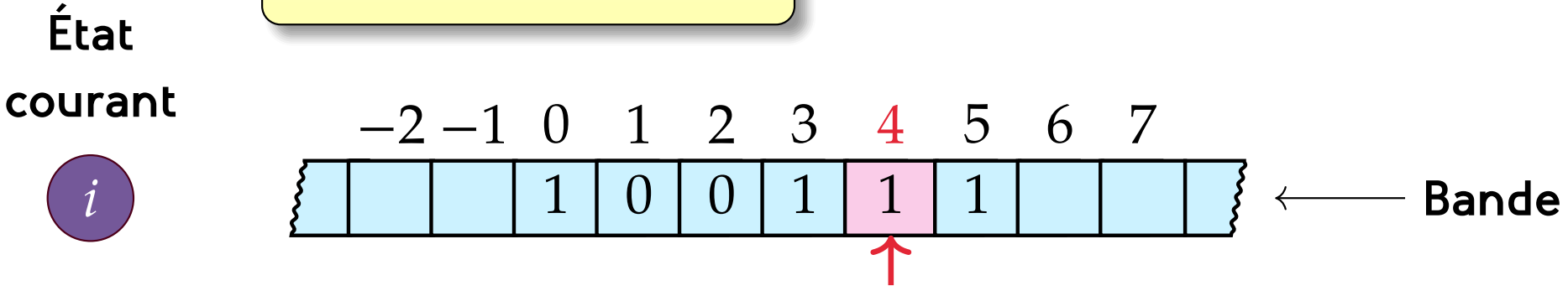
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

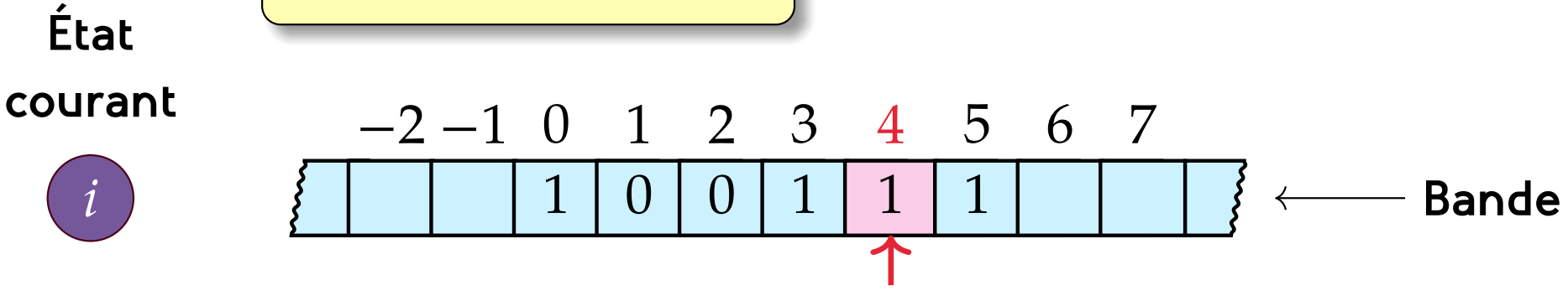
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

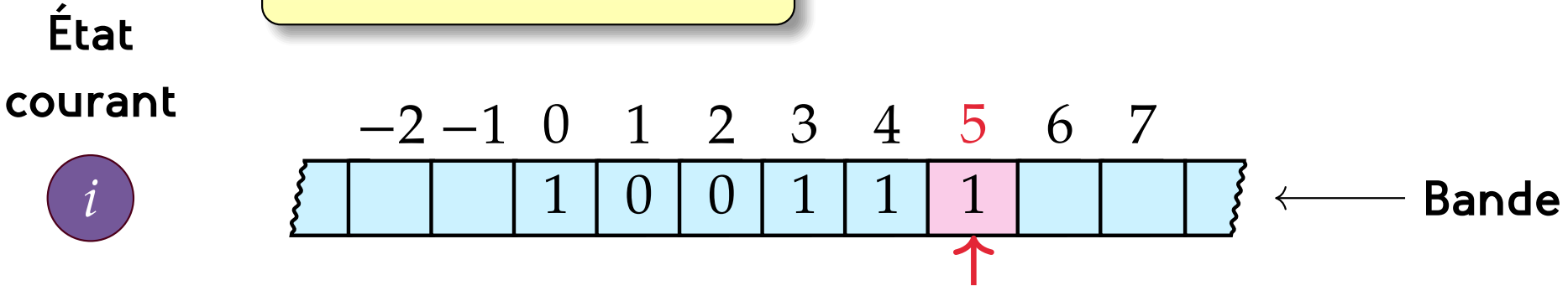
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

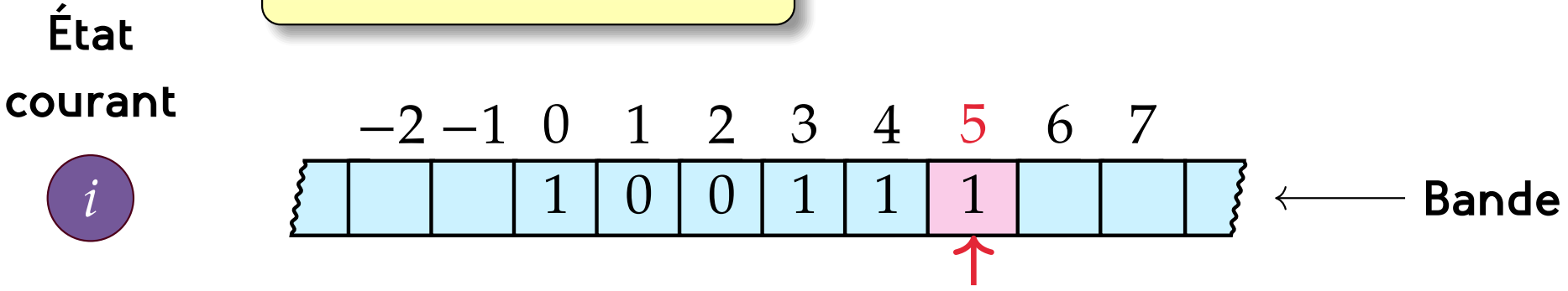
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

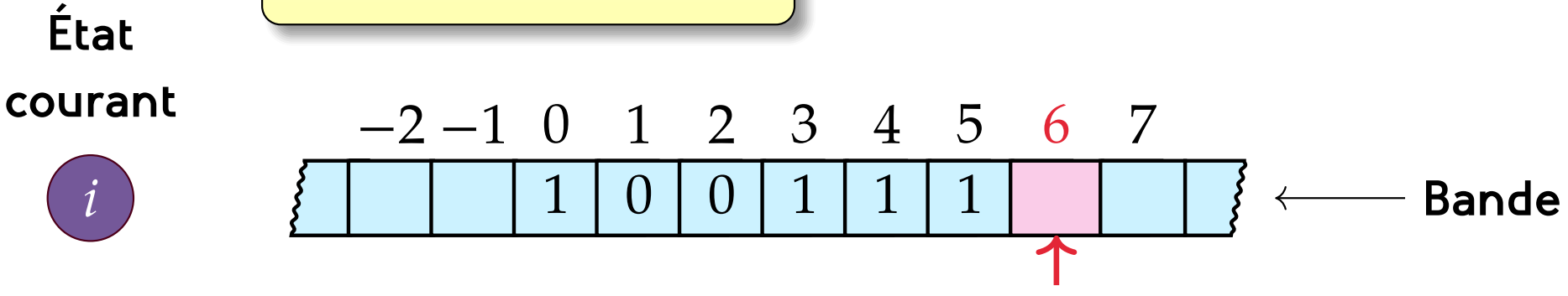
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

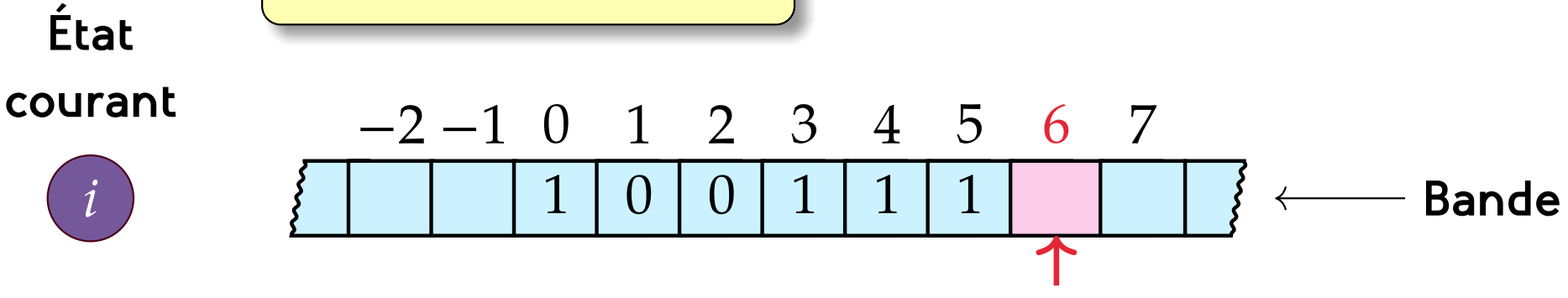
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

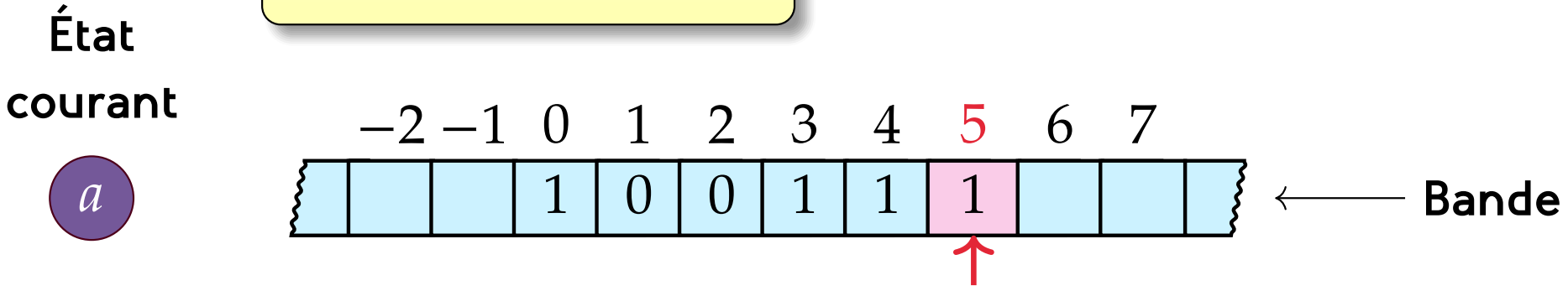
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

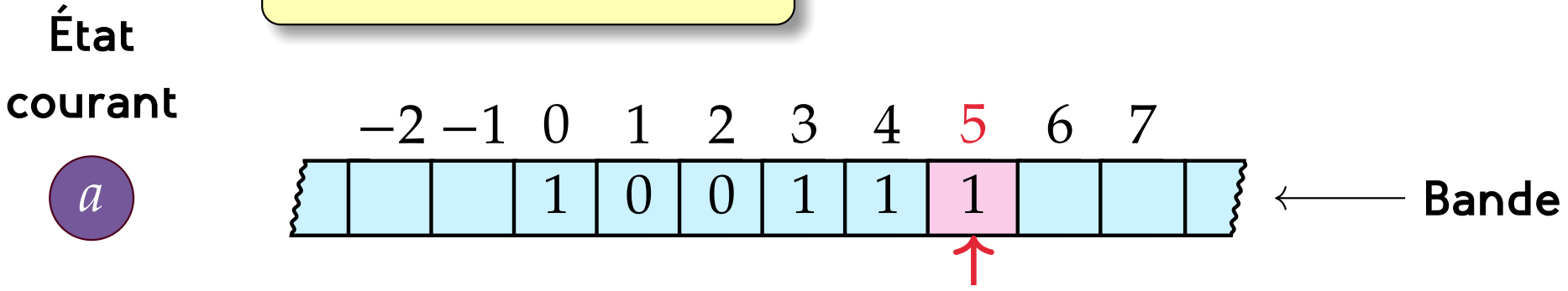
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

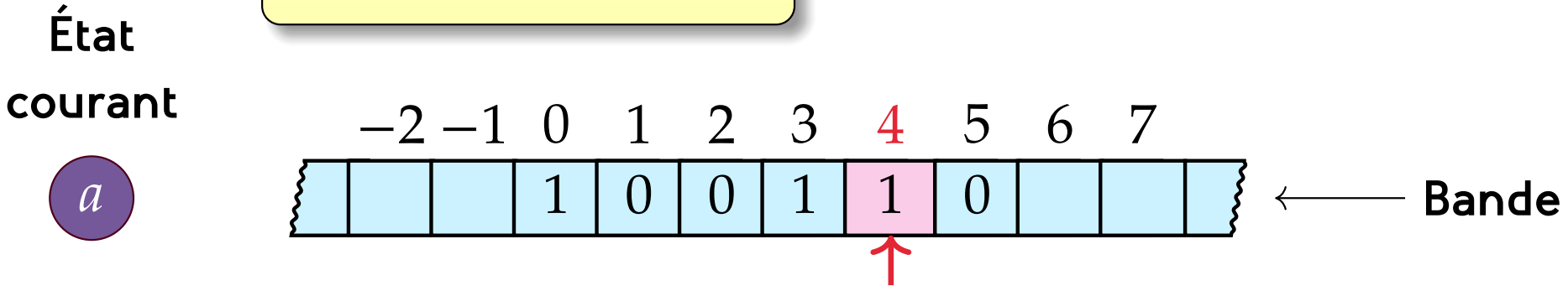
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

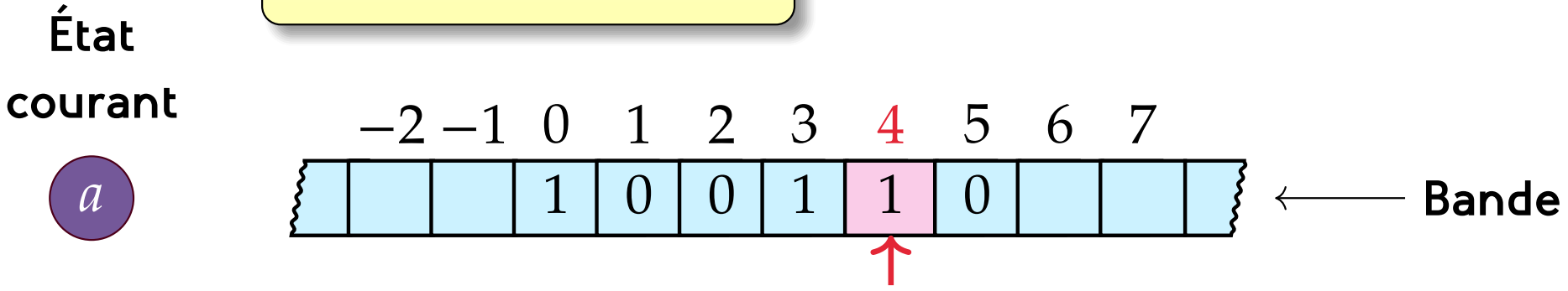
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

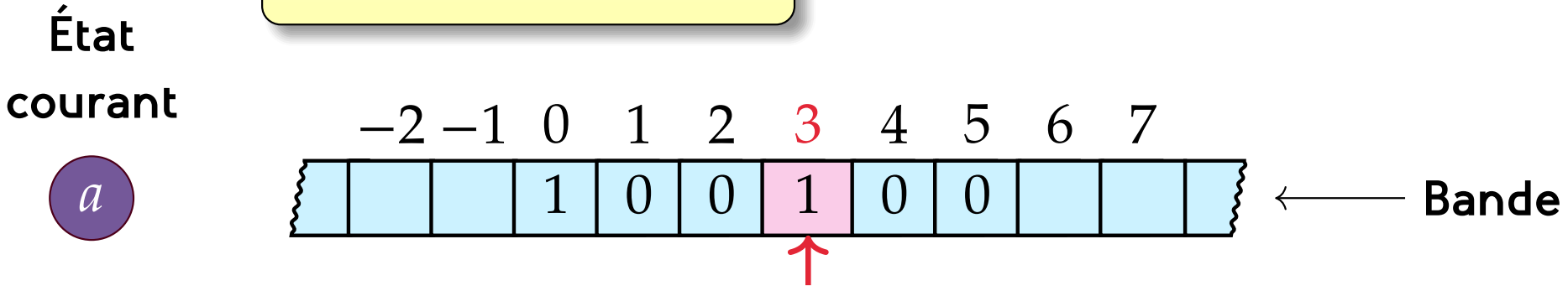
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

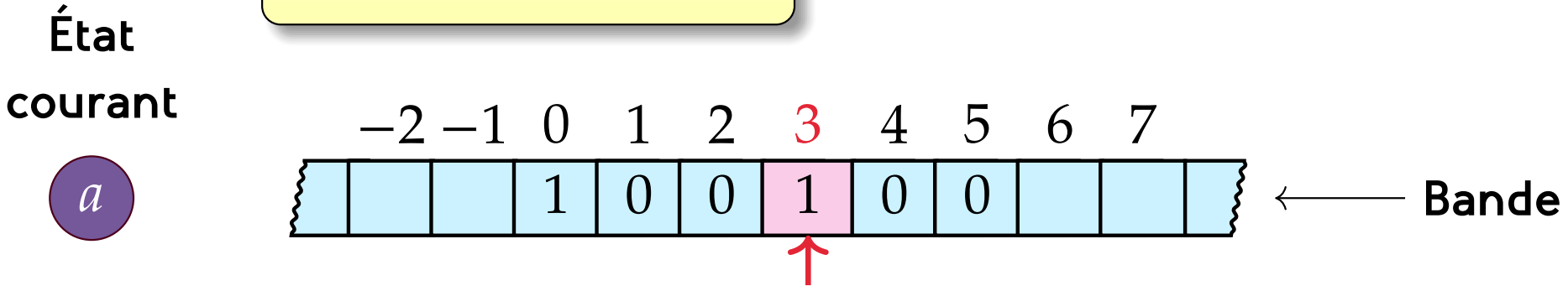
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

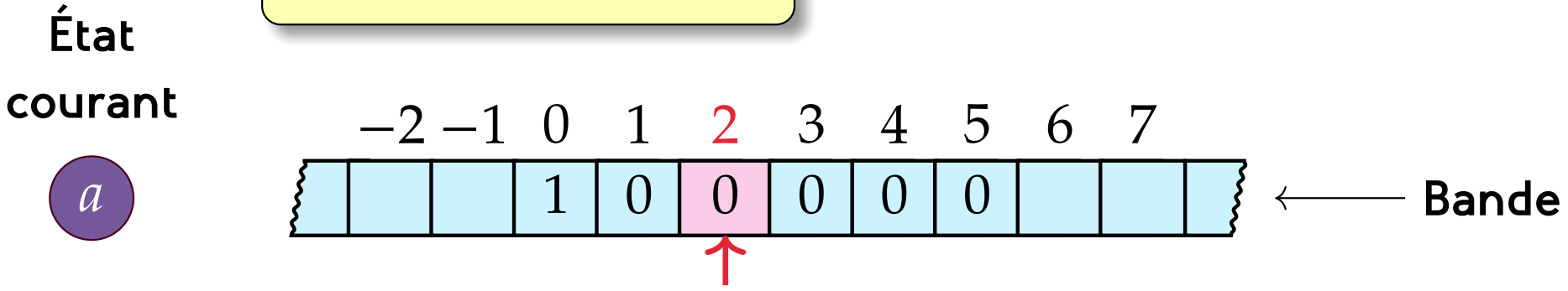
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

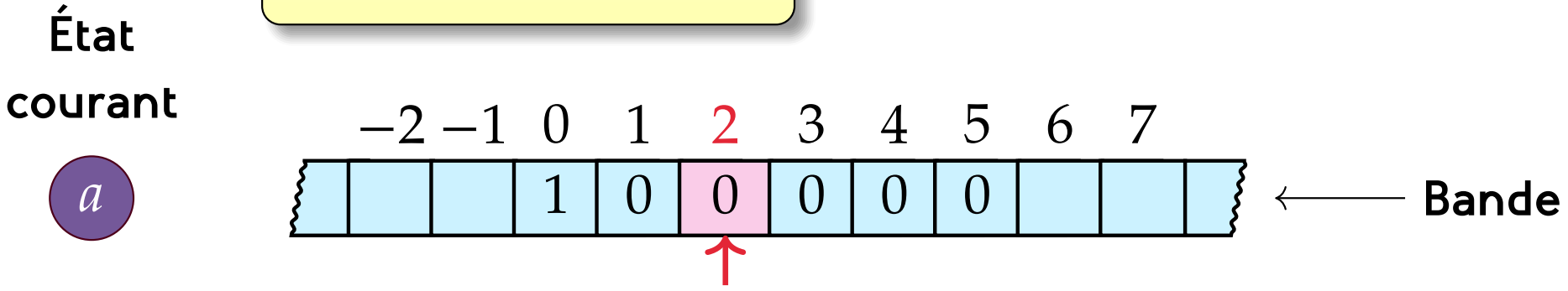
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

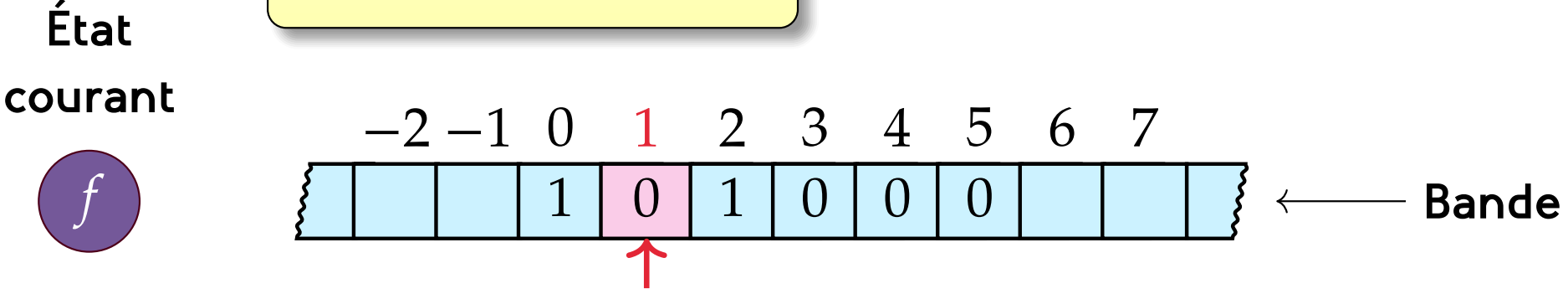
$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Exemple : machine d'incrément en base 2

- Ensemble fini d'états : ici i, a, f.
- Instructions comme $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait passer en état f, écrire 1 (qui écrase le 0), et déplacer la tête à gauche.

$$
\begin{aligned}
\delta(i, 0) &= (i, 0, \rightarrow) \\
\delta(i, 1) &= (i, 1, \rightarrow) \\
\delta(i, \square) &= (a, \square, \leftarrow) \\
\delta(a, 0) &= (f, 1, \leftarrow) \\
\delta(a, 1) &= (a, 0, \leftarrow) \\
\delta(a, \square) &= (f, 1, \leftarrow)
\end{aligned}
$$



Machines de Turing : intuition

- Le nombre de transitions d'une machine de Turing est fini.

  Un programme a un nombre fini d'instructions.

- La bande infinie représente la mémoire de la machine.

  Ajout de périphériques mémoire (disques...) de façon quasi-illimitée.

- La tête se déplace à droite ou à gauche d'une case à chaque étape.

  L'accès à la mémoire est séquentiel.

Exécution d'une machine de Turing

Une machine = **un** programme.

- On part d'une configuration initiale (mot d'entrée sur la bande).
- À chaque étape, **au plus une** transition applicable (déterminisme).
- Tant qu'une transition peut s'appliquer, elle a lieu.
- La machine peut
  - ne jamais s'arrêter, (on dit qu'elle ... ?).
  - s'arrêter,

    (l'état atteint peut être utilisé pour déterminer l'acceptation).

La notion de machine non-déterministe, existe aussi.

Ce modèle mathématique ne représente pas les programmes usuels.

Exécution d'une machine de Turing

Une machine = **un** programme.

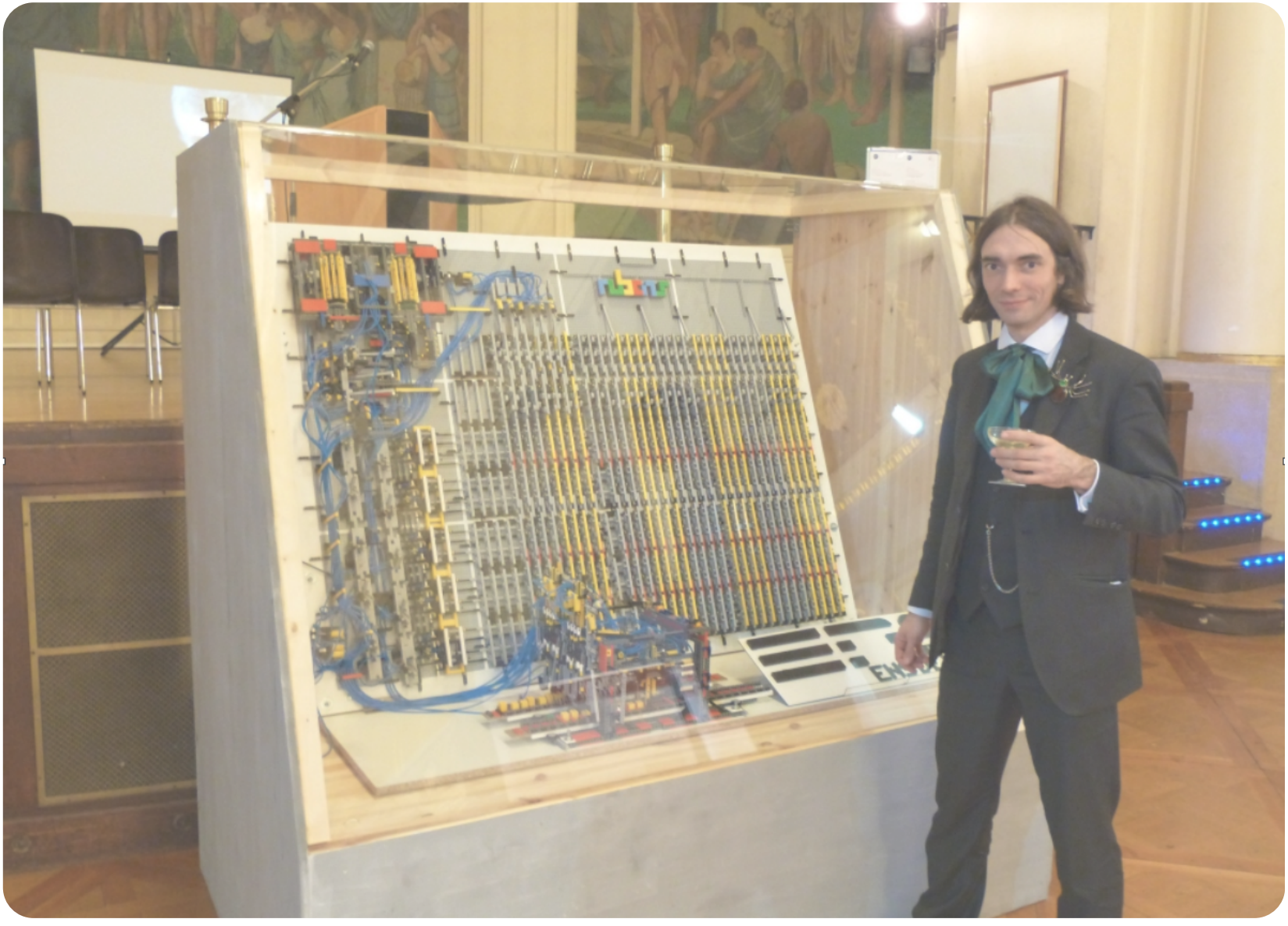
- On part d'une configuration initiale (mot d'entrée sur la bande).
- À chaque étape, **au plus une** transition applicable (déterminisme).
- Tant qu'une transition peut s'appliquer, elle a lieu.
- La machine peut
  - ne jamais s'arrêter, (on dit qu'elle boucle).
  - s'arrêter,

    (l'état atteint peut être utilisé pour déterminer l'acceptation).

La notion de machine non-déterministe, existe aussi.

Ce modèle mathématique ne représente pas les programmes usuels.

Machine de Turing physique



https://www.dailymotion.com/video/xrn0yi

Machines de Turing et thèse de Church

Aussi faible que le langage des machines de Turing puisse paraître :

- On peut coder toute entrée d'un problème informatique sur la bande.
- Tout programme RAM se « **compile** » en Machine de Turing.

Machines de Turing et thèse de Church

Aussi faible que le langage des machines de Turing puisse paraître :

- On peut coder toute entrée d'un problème informatique sur la bande.
- Tout programme RAM se « **compile** » en Machine de Turing.

Thèse de Church

Toute fonction calculable se calcule par machine de Turing.

Conséquences de la thèse de Church

Thèse de Church

Toute fonction calculable se calcule par machine de Turing.

- Il n'y a pas de langage de programmation plus puissant qu'un autre.

  La notion de fonction calculable **ne dépend pas** du langage.

Conséquences de la thèse de Church

Thèse de Church

Toute fonction calculable se calcule par machine de Turing.

- Il n'y a pas de langage de programmation plus puissant qu'un autre.
  La notion de fonction calculable **ne dépend pas** du langage.
- L'indécidabilité de l'arrêt est indépendante du langage utilisé.

Conséquences de la thèse de Church

Thèse de Church

Toute fonction calculable se calcule par machine de Turing.

- Il n'y a pas de langage de programmation plus puissant qu'un autre.
  La notion de fonction calculable **ne dépend pas** du langage.
- L'indécidabilité de l'arrêt est indépendante du langage utilisé.
- Anecdotique : tout langage « Turing-complet » admet un *quine*.
  Lien Wikipedia.

Conséquences de la thèse de Church

Thèse de Church

Toute fonction calculable se calcule par machine de Turing.

- Il n'y a pas de langage de programmation plus puissant qu'un autre.
  La notion de fonction calculable **ne dépend pas** du langage.
- L'indécidabilité de l'arrêt est indépendante du langage utilisé.
- Anecdotique : tout langage « Turing-complet » admet un *quine*.
  Lien Wikipedia.

Prochaine partie : les **réductions**.

Plan

1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

3. Réductions et NP-complétude

4. Complexité des problèmes – P $\stackrel{?}{=}$ NP

Idée générale

Principe des réductions

Pour résoudre un problème 1, utiliser un algorithme pour un problème 2.

Idée générale

Principe des réductions

Pour résoudre un problème 1, utiliser un algorithme pour un problème 2.

3 étapes (certaines peuvent être triviales) :

1. Transformer l'entrée e du problème 1 en entrée t(e) du problème 2.
2. Faire tourner l'algorithme du problème 2 sur t(e).
3. Interpréter le résultat pour répondre au problème 1 sur l'instance e.

Idée générale

Principe des réductions

Pour résoudre un problème 1, utiliser un algorithme pour un problème 2.

3 étapes (certaines peuvent être triviales) :

1. Transformer l'entrée e du problème 1 en entrée t(e) du problème 2.
2. Faire tourner l'algorithme du problème 2 sur t(e).
3. Interpréter le résultat pour répondre au problème 1 sur l'instance e.

Intérêt double

- Obtenir un algorithme (et un coût maximal) pour le problème 1.
- Si on sait le problème 1 difficile, il en va de même du problème 2.

Exemples avec le problème du tri

Problème du tri

Instance Un tableau de n entiers.

Question Déterminer un tri de ce tableau en comparant les éléments.

Remarque

Avec un modèle de coût raisonnable, on sait (lien) que ce problème

- admet un algorithme de coût $O(n \log(n))$,
- qu'on ne peut pas faire mieux : borne inférieure $\Omega(n \log(n))$.

Exemple 1

Problème de la médiane

Instance Un tableau de n entiers.

Question Déterminer la médiane des éléments du tableau.

Exemple 1

Problème de la médiane

Instance Un tableau de n entiers.

Question Déterminer la médiane des éléments du tableau.

On utilise un algorithme de tri (*i.e.*, Tri joue le rôle du problème 2).

2. trier le tableau, $O(n \log(n))$
3. retourner l'élément en position `n // 2`. $O(1)$

Exemple 1

Problème de la médiane

Instance Un tableau de n entiers.

Question Déterminer la médiane des éléments du tableau.

On utilise un algorithme de tri (*i.e.*, Tri joue le rôle du problème 2).

2. trier le tableau, $O(n \log(n))$
3. retourner l'élément en position `n // 2`. $O(1)$

Le problème de la médiane **se réduit** au problème du tri avec surcoût $O(1)$

« *Médiane* est plus facile que *tri* (avec surcoût $O(1)$) »

On obtient ainsi un algorithme en $O(n \log(n))$.

Exemple 1

Problème de la médiane

Instance Un tableau de n entiers.

Question Déterminer la médiane des éléments du tableau.

On utilise un algorithme de tri (*i.e.*, Tri joue le rôle du problème 2).

2. trier le tableau, $O(n \log(n))$
3. retourner l'élément en position `n // 2`. $O(1)$

Le problème de la médiane **se réduit** au problème du tri avec surcoût $O(1)$

« *Médiane* est plus facile que *tri* (avec surcoût $O(1)$) »

On obtient ainsi un algorithme en $O(n \log(n))$.

Remarque

On peut faire mieux : calcul de la médiane en temps $O(n)$.

Exemple 2

Problème Distinctness

Instance Un tableau de n entiers.

Question Toutes les valeurs du tableau sont-elles distinctes ?

Exemple 2

Problème Distinctness

Instance Un tableau de n entiers.

Question Toutes les valeurs du tableau sont-elles distinctes ?

On utilise un algorithme de tri qui joue le rôle du problème 2.

2. trier le tableau, $O(n \log(n))$
3. rechercher si deux éléments **consécutifs** sont égaux. $O(n)$

Exemple 2

Problème Distinctness

Instance Un tableau de n entiers.

Question Toutes les valeurs du tableau sont-elles distinctes ?

On utilise un algorithme de tri qui joue le rôle du problème 2.

2. trier le tableau, $O(n \log(n))$
3. rechercher si deux éléments **consécutifs** sont égaux. $O(n)$

Le problème distinctness **se réduit** au problème du tri avec surcoût $O(n)$

« *Distinctness* est plus facile que *tri* (avec surcoût $O(n)$) »

On obtient ainsi un algorithme en $O(n \log(n))$.

Exemple 2

Problème Distinctness

Instance Un tableau de n entiers.

Question Toutes les valeurs du tableau sont-elles distinctes ?

On utilise un algorithme de tri qui joue le rôle du problème 2.

2. trier le tableau, $O(n \log(n))$
3. rechercher si deux éléments **consécutifs** sont égaux. $O(n)$

Le problème distinctness **se réduit** au problème du tri avec surcoût $O(n)$

« *Distinctness* est plus facile que *tri* (avec surcoût $O(n)$) »

On obtient ainsi un algorithme en $O(n \log(n))$.

Remarque

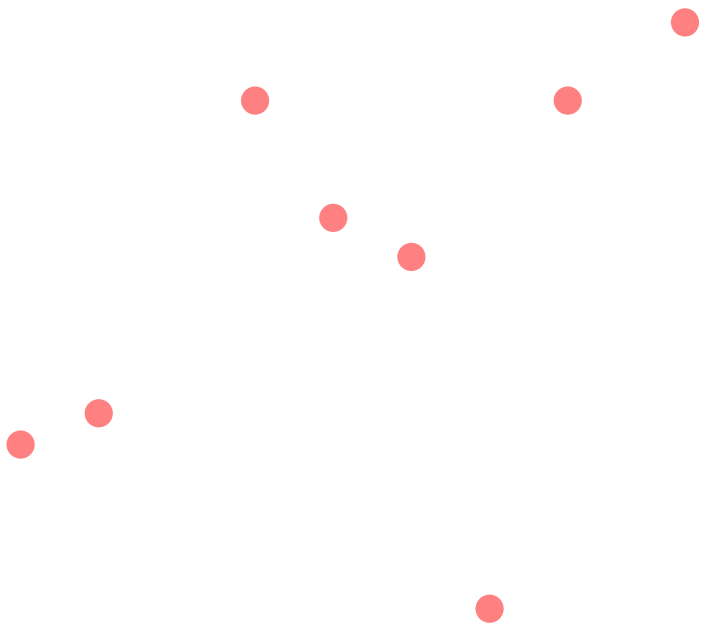
On ne peut pas faire mieux que $n \log(n)$ *(avec modèle de coût raisonnable)*.

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

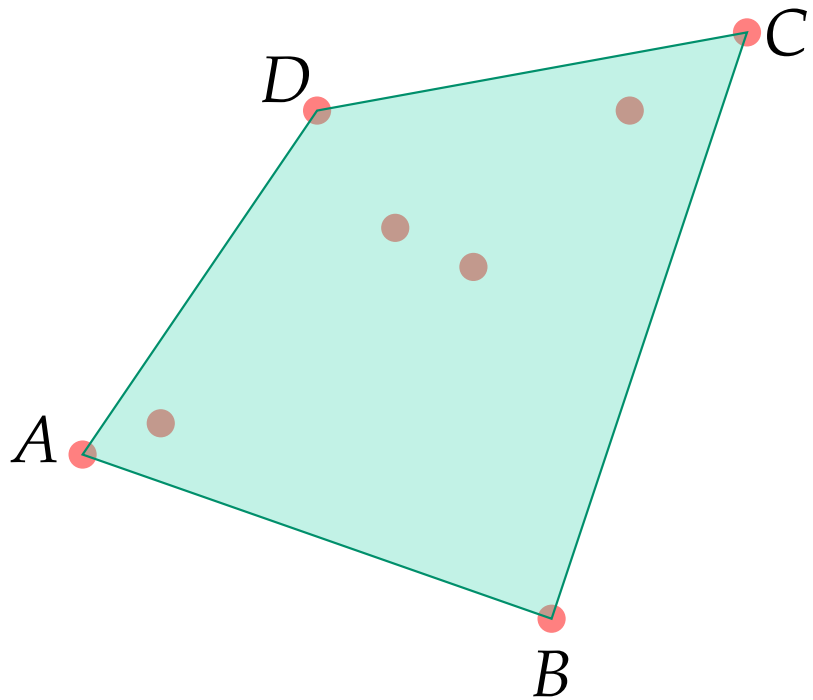


Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.



Enveloppe convexe : A, B, C, D

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$

Exemple 3

Problème du calcul d'enveloppe convexe

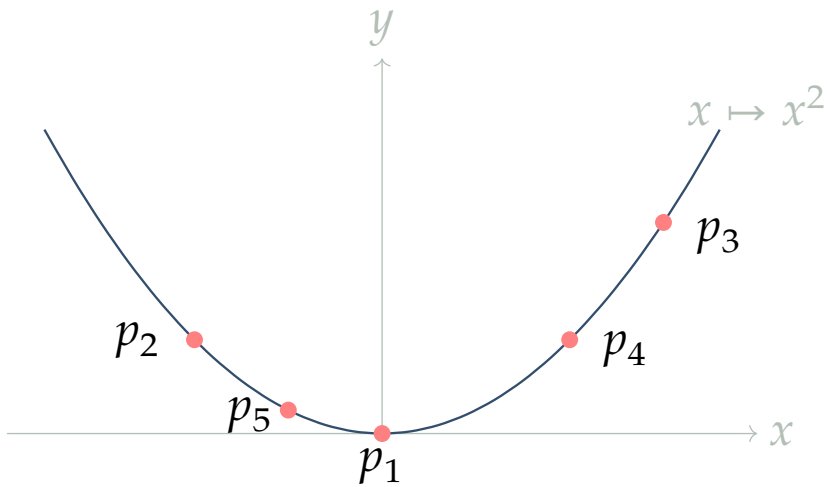
Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$



Exemple 3

Problème du calcul d'enveloppe convexe

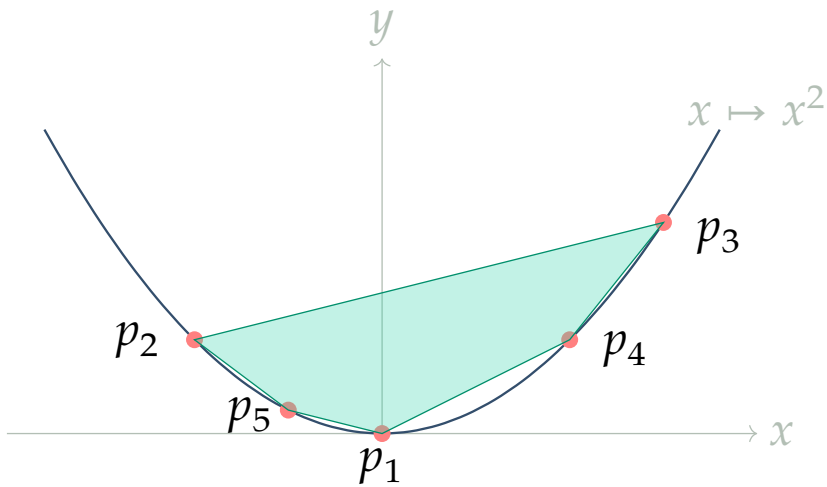
Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$



Exemple 3

Problème du calcul d'enveloppe convexe

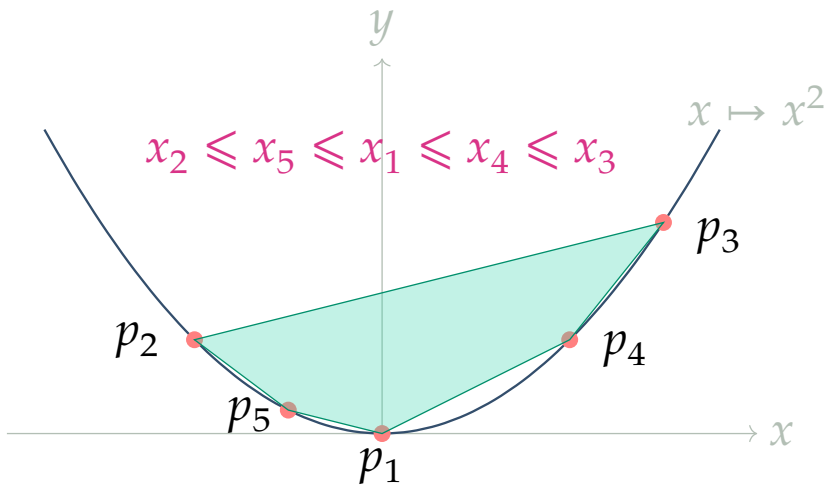
Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$



Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$

À quoi cela sert-il ? Je sais **déjà** résoudre le problème Tri !

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$

À quoi cela sert-il ? Je sais **déjà** résoudre le problème Tri !

- On a résolu Tri en $O(n) + f(n)$.
- On sait qu'on ne peut pas faire mieux que $\Omega(n \log(n))$.
- Donc $f(n) = \Omega(n \log(n))$.

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$

À quoi cela sert-il ? Je sais **déjà** résoudre le problème Tri !

- On a résolu Tri en $O(n) + f(n)$.
- On sait qu'on ne peut pas faire mieux que $\Omega(n \log(n))$.
- Donc $f(n) = \Omega(n \log(n))$.

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$

À quoi cela sert-il ? Je sais **déjà** résoudre le problème Tri !

- On a résolu Tri en $O(n) + f(n)$.
- On sait qu'on ne peut pas faire mieux que $\Omega(n \log(n))$.
- Donc $f(n) = \Omega(n \log(n))$.

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$

À quoi cela sert-il ? Je sais **déjà** résoudre le problème Tri !

Conséquence : Tout algorithme pour Enveloppe convexe est $\Omega(n \log(n))$.

Exemple 3

Problème du calcul d'enveloppe convexe

Instance Un tableau de n points du plan.

Question Déterminer son enveloppe convexe.

On utilise Enveloppe convexe, supposé de pire coût $f(n)$, pour résoudre Tri.

Pour trier $x_1, \ldots, x_n$:

1. calculer les points $p_i = (x_i, x_i^2)$. $O(n)$
2. appliquer l'algorithme d'Enveloppe convexe sur les p_i. $f(n)$
3. lire les abscisses des points retournés depuis la plus petite. $O(n)$

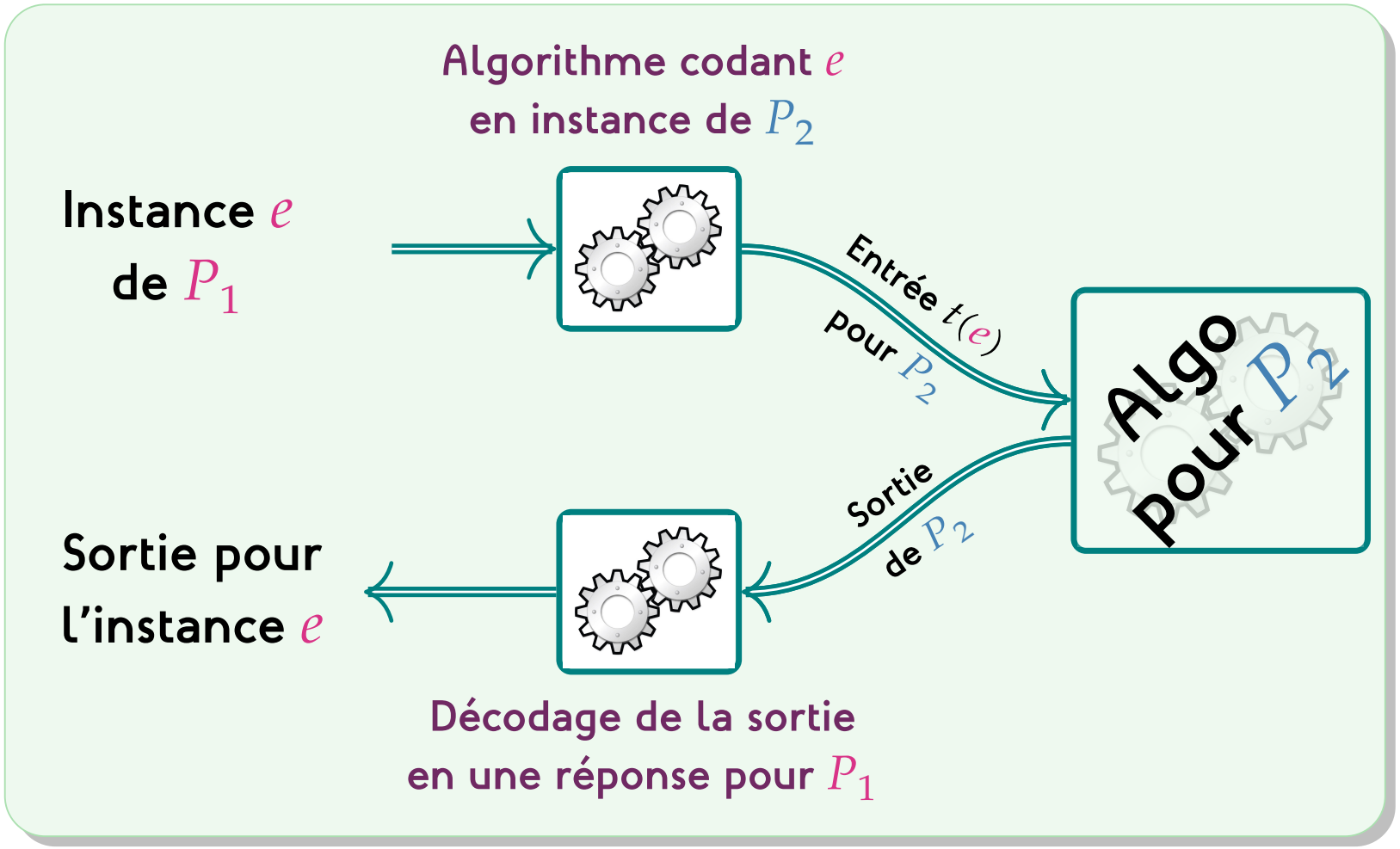
À quoi cela sert-il ? Je sais **déjà** résoudre le problème Tri !

Conséquence : Tout algorithme pour Enveloppe convexe est $\Omega(n \log(n))$.

Remarque

En fait, on peut calculer l'enveloppe convexe en $O(n \log(n))$.

Schéma général déjà rencontré !



Récapitulatif

Réduire P_1 à P_2 = ramener la résolution de P_1 à celle de P_2.

Algorithme pour P_2 $\Rightarrow$ algorithme pour P_1.

Récapitulatif

Réduire P_1 à P_2 = ramener la résolution de P_1 à celle de P_2.

Algorithme pour P_2 $\Rightarrow$ algorithme pour P_1.

Les propriétés **positives** de P_2 se transmettent à P_1.
Les propriétés **négatives** de P_1 se transmettent à P_2.

Récapitulatif

Réduire P_1 à P_2 = ramener la résolution de P_1 à celle de P_2.

Algorithme pour P_2 $\Rightarrow$ algorithme pour P_1.

Les propriétés **positives** de P_2 se transmettent à P_1.
Les propriétés **négatives** de P_1 se transmettent à P_2.

Conséquence 1 : « héritage » des propriétés positives

On peut utiliser un programme efficace pour résoudre P_1.

Cas pratique : P_2 = SAT. Intéressant car :

- solveurs SAT efficaces,
- codage vers SAT facile.

Récapitulatif

Réduire P_1 à P_2 = ramener la résolution de P_1 à celle de P_2.

Algorithme pour P_2 $\Rightarrow$ algorithme pour P_1.

Les propriétés **positives** de P_2 se transmettent à P_1.
Les propriétés **négatives** de P_1 se transmettent à P_2.

Conséquence 1 : « héritage » des propriétés positives

On peut utiliser un programme efficace pour résoudre P_1.

Cas pratique : P_2 = SAT. Intéressant car :

- solveurs SAT efficaces,
- codage vers SAT facile.

Conséquence 2 : « héritage » des propriétés négatives

1. Si P_1 est connu indécidable, alors P_2 l'est aussi.
2. Si P_1 a une complexité au moins exponentielle et que la réduction est polynomiale, P_2 a une complexité au moins exponentielle.

Réduction pour prouver l'indécidabilité

On a déjà réduit le problème de l'arrêt à celui de l'affectation de 42.

Réduction pour prouver l'indécidabilité

On a déjà réduit le problème de l'arrêt à celui de l'affectation de 42.

Indécidabilité de pavage par Tetris, admettant celle de pavage par Wang

À partir d'un jeu de tuiles de Wang,

- on interprète les couleurs comme des entiers strictement positifs,
- on crée une tuile Tetris par tuile de Wang,
- les encoches/saillies ont la profondeur des entiers correspondants.

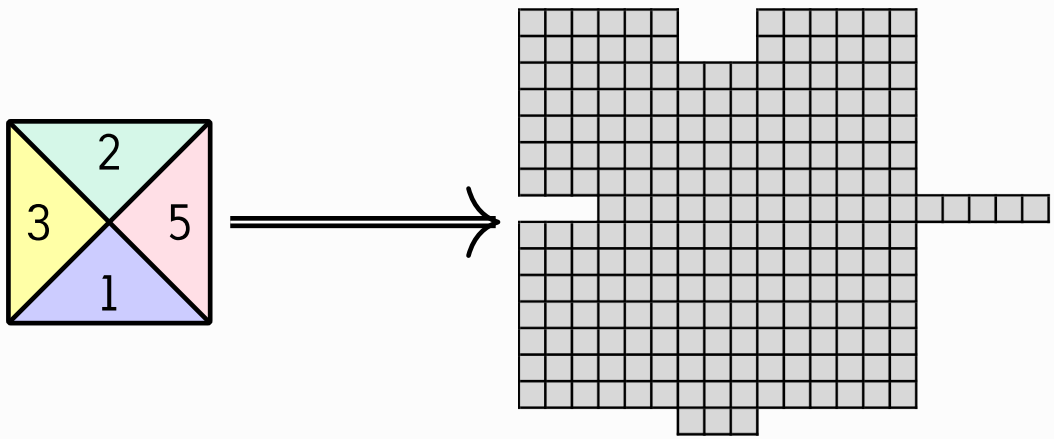
Réduction pour prouver l'indécidabilité

On a déjà réduit le problème de l'arrêt à celui de l'affectation de 42.

Indécidabilité de pavage par Tetris, admettant celle de pavage par Wang

À partir d'un jeu de tuiles de Wang,

- on interprète les couleurs comme des entiers strictement positifs,
- on crée une tuile Tetris par tuile de Wang,
- les encoches/saillies ont la profondeur des entiers correspondants.



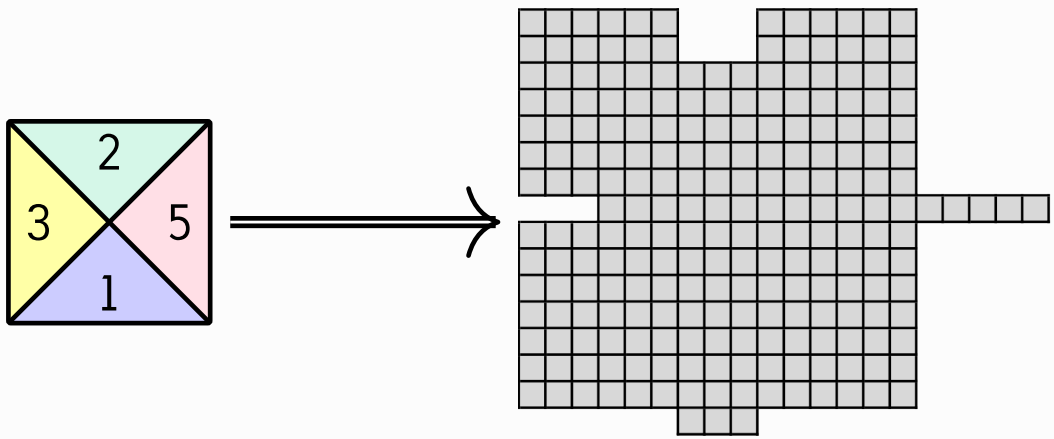
Réduction pour prouver l'indécidabilité

On a déjà réduit le problème de l'arrêt à celui de l'affectation de 42.

Indécidabilité de pavage par Tetris, admettant celle de pavage par Wang

À partir d'un jeu de tuiles de Wang,

- on interprète les couleurs comme des entiers strictement positifs,
- on crée une tuile Tetris par tuile de Wang,
- les encoches/saillies ont la profondeur des entiers correspondants.



Pavage avec jeu de Wang original $\iff$ pavage avec jeu de Tetris construit.

Tetris est donc aussi **indécidable**.

Solveurs SAT et réductions : démo

Tests de coloration (graphes générés par gengraph, solveur Glucose)

```
$ ./checkproperty kcolor 3  petersen
$ ./checkproperty kcolor 3  grotzsch
$ ./checkproperty kcolor 3  outerplanar 1000
$ ./checkproperty kcolor 3  gabriel 1000
$ ./checkproperty kcolor 4  gabriel 1000
$ ./checkproperty kcolor 2  hypercube 5
$ ./checkproperty kcolor 2  cubic 100
$ ./checkproperty kcolor 6  ktree 5000 6
$ ./checkproperty kcolor 7  ktree 5000 6
$ ./checkproperty kcolor 10 ktree 50 10
$ ./checkproperty kcolor 3  fdrg 60 3 80 4 10 8 20 10 .
$ ./checkproperty kcolor 2  syracuse 18
$ ./checkproperty kcolor 9  clique 10
$ ./checkproperty kcolor 11 clique 12
```

Plan

1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

3. Réductions et NP-complétude

4. Complexité des problèmes – $P \stackrel{?}{=} NP$

Objectif

Classer les problèmes de décision selon leur difficulté.

Première classification

- Problèmes décidables.
- Problèmes indécidables.

Objectif

Classer les problèmes de décision selon leur difficulté.

Première classification

- Problèmes décidables.
- Problèmes indécidables.

Attention : on parle bien de classer les problèmes de décision.

Objectif

Classer les problèmes de décision selon leur difficulté.

Première classification

- Problèmes décidables.
- Problèmes indécidables.

Attention : on parle bien de classer les problèmes de décision.

Classification naturelle des problèmes décidables

Selon du coût du **meilleur algorithme** pour les résoudre.

Objectif

Classer les problèmes de décision selon leur difficulté.

Première classification

- Problèmes décidables.
- Problèmes indécidables.

Attention : on parle bien de classer les problèmes de décision.

Classification naturelle des problèmes décidables

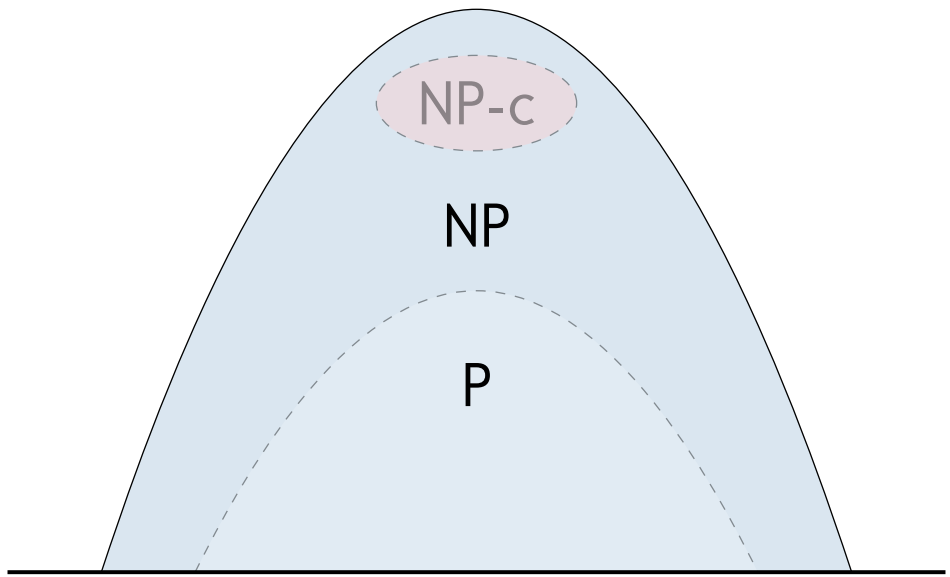
Selon du coût du **meilleur algorithme** pour les résoudre.

- Considéré comme bon : polynomial.
- À opposer, typiquement, à un coût exponentiel.

Situer les classes qu'on va définir

A priori, il y a 3 classes de **problèmes de décision**, et non 2 :

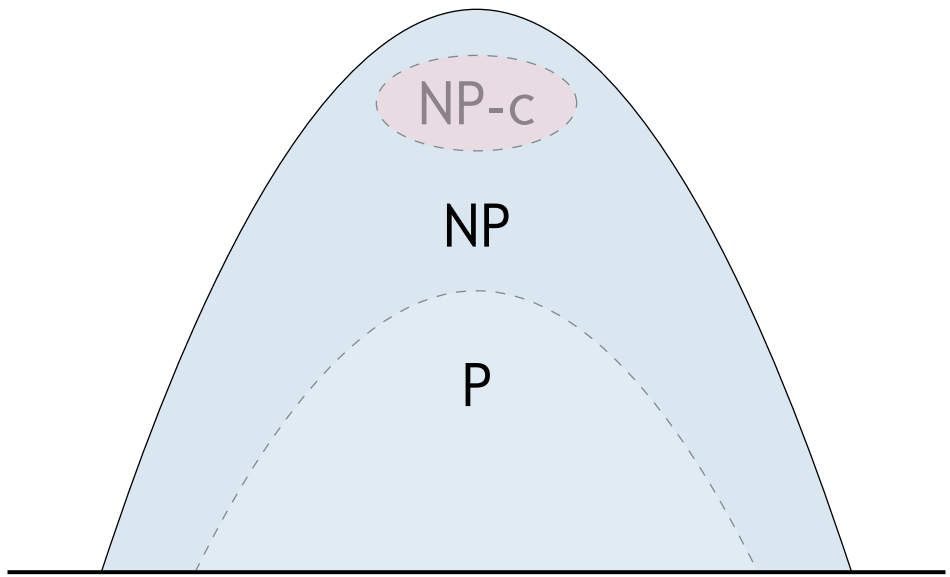
- La classe P.
- La classe NP.
- La classe des problèmes NP-complets.



Situer les classes qu'on va définir

A priori, il y a 3 classes de **problèmes de décision**, et non 2 :

- La classe P.
- La classe NP.
- La classe des problèmes NP-complets.



- « *A priori* », parce que si on avait P = NP, ces classes coïncideraient.
- Il s'agit de classes **de problèmes** : SAT est un « point » de cette figure.

La classe P

La classe P

Constituée des problèmes qui peuvent se résoudre en temps polynomial.

La classe P

La classe P

Constituée des problèmes qui peuvent se résoudre en temps polynomial.

Attention : ce temps est évalué en fonction de la taille de l'entrée.

La classe P

La classe P

Constituée des problèmes qui peuvent se résoudre en temps polynomial.

Attention : ce temps est évalué en fonction de la taille de l'entrée.

Exemple (non trivial)

Test de primalité d'un entier.

Où placer SAT ?

Après P (temps polynomial), on peut définir EXP (temps exponentiel).

Où placer SAT ?

Après P (temps polynomial), on peut définir EXP (temps exponentiel).

La difficulté posée par le problème SAT

- On ne sait pas montrer qu'il est (ou n'est pas) dans P.
- On croit qu'il n'est pas un des problèmes « les plus difficiles » de EXP.

Où placer SAT ?

Après P (temps polynomial), on peut définir EXP (temps exponentiel).

La difficulté posée par le problème SAT

- On ne sait pas montrer qu'il est (ou n'est pas) dans P.
- On croit qu'il n'est pas un des problèmes « les plus difficiles » de EXP.

Pour le situer, on définit une classe intermédiaire, NP, entre P et EXP.

La classe NP

La classe NP

Constituée des problèmes qui peuvent se résoudre en temps polynomial par machine de Turing non-déterministe.

Un modèle non réaliste : machines non déterministes

C'est une machine de Turing qui peut, en plus, « deviner » de l'information.
La machine a une capacité « magique » : deviner.

La classe NP

La classe NP

Constituée des problèmes qui peuvent se résoudre en temps polynomial par machine de Turing non-déterministe.

Un modèle non réaliste : machines non déterministes

C'est une machine de Turing qui peut, en plus, « deviner » de l'information. La machine a une capacité « magique » : deviner.

Un problème de décision est dans NP s'il est résolu par une machine qui :

- **en phase 1**, devine une « solution » en temps polynomial,
- **en phase 2**, vérifie la solution en temps polynomial déterministe.

La classe NP

La classe NP

Constituée des problèmes qui peuvent se résoudre en temps polynomial par machine de Turing non-déterministe.

Un modèle non réaliste : machines non déterministes

C'est une machine de Turing qui peut, en plus, « deviner » de l'information.
La machine a une capacité « magique » : deviner.

Un problème de décision est dans NP s'il est résolu par une machine qui :

- **en phase 1**, devine une « solution » en temps polynomial,
- **en phase 2**, vérifie la solution en temps polynomial déterministe.

Une telle machine est un **outil mathématique** pour situer SAT.

La classe NP

La classe NP

Constituée des problèmes qui peuvent se résoudre en temps polynomial par machine de Turing non-déterministe.

Un modèle non réaliste : machines non déterministes

C'est une machine de Turing qui peut, en plus, « deviner » de l'information.
La machine a une capacité « magique » : deviner.

Un problème de décision est dans NP s'il est résolu par une machine qui :

- **en phase 1**, devine une « solution » en temps polynomial,
- **en phase 2**, vérifie la solution en temps polynomial déterministe.

Une telle machine est un **outil mathématique** pour situer SAT.
Se passer de l'instruction « deviner » coûte-t-il le caractère polynomial ?

La classe NP : exemples

Attention au nom de la classe

NP signifie « non déterministe polynomial », et pas « non polynomial ».

La classe NP : exemples

Attention au nom de la classe

NP signifie « non déterministe polynomial », et pas « non polynomial ».

SAT est dans NP

Phase 1 : On devine une affectation V/F des variables (temps linéaire).

Phase 2 : On vérifie que cette affectation est correcte (temps linéaire).

La classe NP : exemples

Attention au nom de la classe

NP signifie « non déterministe polynomial », et pas « non polynomial ».

SAT est dans NP

Phase 1 : On devine une affectation V/F des variables (temps linéaire).

Phase 2 : On vérifie que cette affectation est correcte (temps linéaire).

3-COL est dans NP

Phase 1 : On devine un 3-coloration (temps linéaire).

Phase 2 : On vérifie que cette coloration est correcte (temps linéaire).

La classe NP : exemples

Attention au nom de la classe

NP signifie « non déterministe polynomial », et pas « non polynomial ».

SAT est dans NP

Phase 1 : On devine une affectation V/F des variables (temps linéaire).

Phase 2 : On vérifie que cette affectation est correcte (temps linéaire).

3-COL est dans NP

Phase 1 : On devine un 3-coloration (temps linéaire).

Phase 2 : On vérifie que cette coloration est correcte (temps linéaire).

Tout problème de P est dans NP

On n'a même pas besoin de deviner en phase 1 : on fait un calcul exact.

$$P \subseteq NP$$

SAT dans la classe NP

On peut montrer :

$$P \subseteq NP \subseteq EXP$$

SAT dans la classe NP

On peut montrer :

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$$

On voudrait que la classe NP « **sépare** » SAT des problèmes de P.

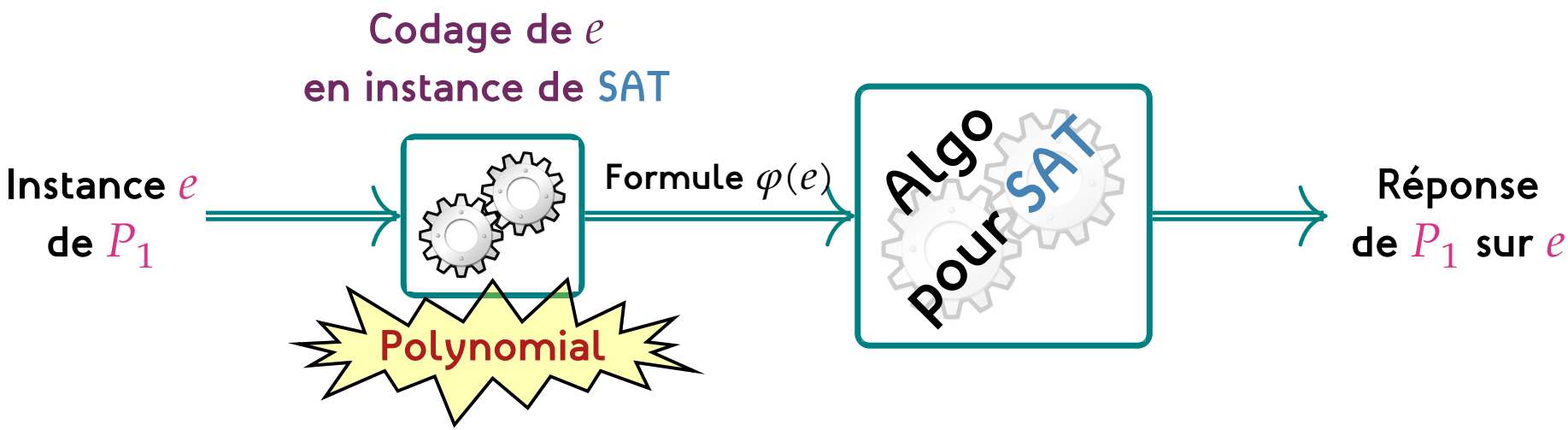
SAT dans la classe NP

On peut montrer :

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$$

On voudrait que la classe NP « **sépare** » SAT des problèmes de P.

Retour sur la notion de réduction (sans décodage)



« P_1 **se réduit** à SAT en temps polynomial ».

« P_1 **est plus facile que** SAT, modulo surcoût polynomial ».

Théorème de Cook-Levin

Définition : problème NP-complet

Un problème Q est **NP-complet** si

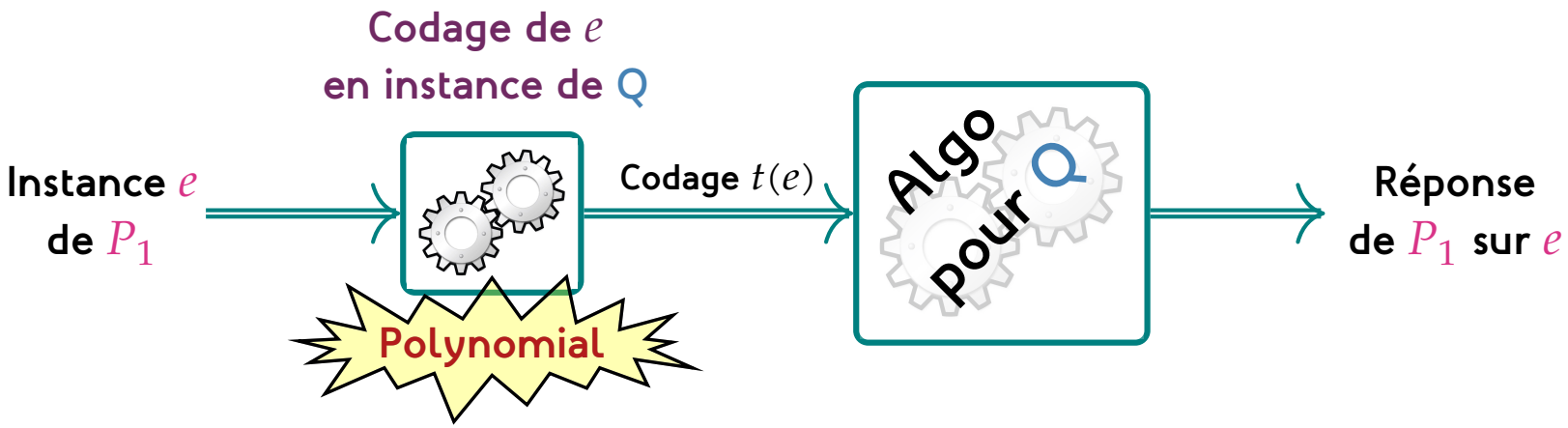
1. Q est dans NP.
2. Tout problème de la classe NP se réduit à Q en temps polynomial.

Théorème de Cook-Levin

Définition : problème NP-complet

Un problème Q est **NP-complet** si

1. Q est dans NP.
2. Tout problème de la classe NP se réduit à Q en temps polynomial.

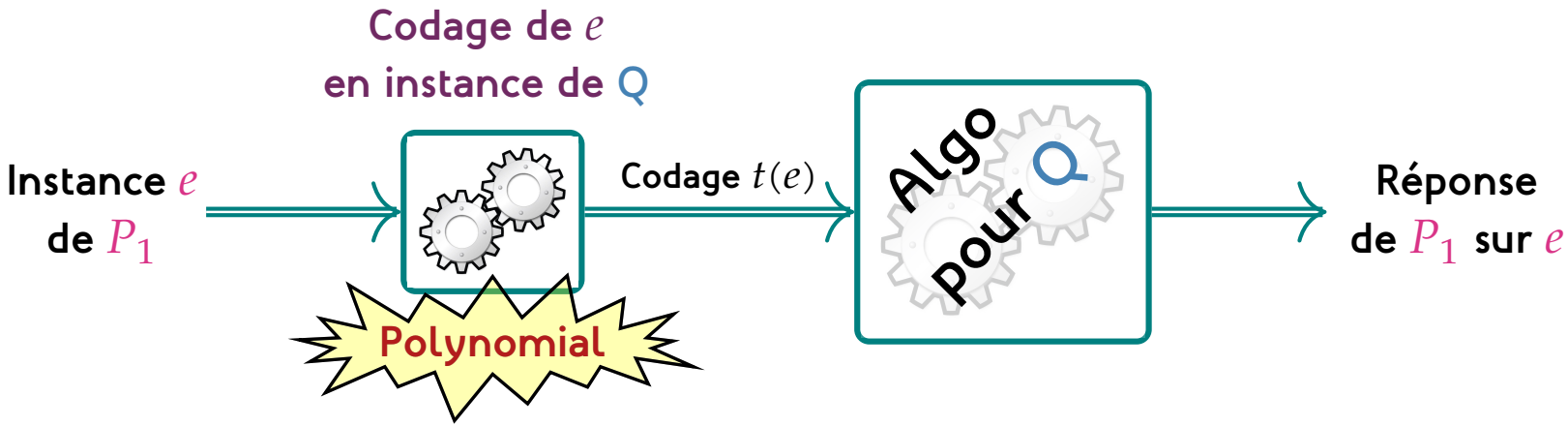


Théorème de Cook-Levin

Définition : problème NP-complet

Un problème Q est **NP-complet** si

1. Q est dans NP.
2. Tout problème de la classe NP se réduit à Q en temps polynomial.



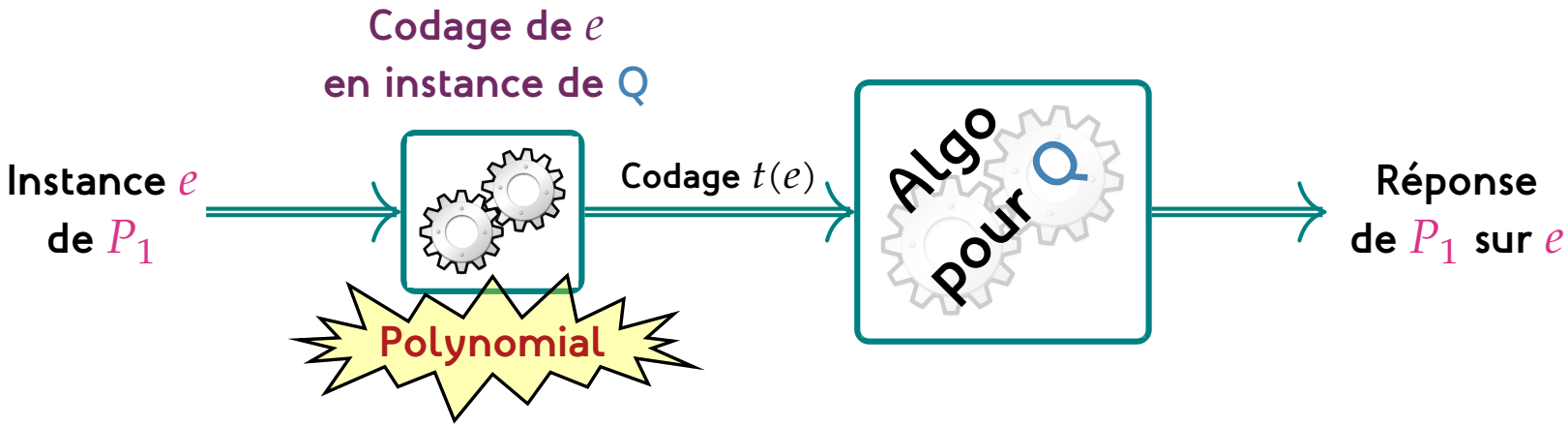
En particulier, si un tel problème Q est dans la classe **P**, alors **P** = **NP**.

Théorème de Cook-Levin

Définition : problème NP-complet

Un problème Q est **NP-complet** si

1. Q est dans NP.
2. Tout problème de la classe NP se réduit à Q en temps polynomial.



En particulier, si un tel problème Q est dans la classe **P**, alors **P** = **NP**.

Théorème de Cook-Levin (1971)

SAT est NP-complet.

Théorème de Cook-Levin

Théorème de Cook-Levin (1971)

SAT est NP-complet.

On peut réduire tout problème de **NP** en SAT, modulo surcoût polynomial.

Théorème de Cook-Levin

Théorème de Cook-Levin (1971)

SAT est NP-complet.

On peut réduire tout problème de **NP** en SAT, modulo surcoût polynomial.

Corollaire important !

Si SAT est dans **P**, alors **P** $=$ **NP**.

Une réduction polynomiale de 3-COL à SAT

Problème 3-COL

Instance Un graphe.

Question Le graphe a-t-il une 3-coloration ?

Une réduction polynomiale de 3-COL à SAT

Problème 3-COL

Instance Un graphe.

Question Le graphe a-t-il une 3-coloration ?

Objectif : réduction vers SAT, i.e., algorithme polynomial...

... prenant en entrée un graphe G, construisant une formule $\varphi(G)$, où

$$\varphi(G) \textbf{ satisfaisable} \iff G \textbf{ est 3-colorable}.$$

Une réduction polynomiale de 3-COL à SAT

Problème 3-COL

Instance Un graphe.

Question Le graphe a-t-il une 3-coloration ?

Objectif : réduction vers SAT, i.e., algorithme polynomial...

... prenant en entrée un graphe G, construisant une formule $\varphi(G)$, où

$$\varphi(G) \textbf{ satisfaisable} \iff G \textbf{ est 3-colorable}.$$

- pour chaque sommet s, on crée 3 variables r_s, g_s, b_s et une formule $\alpha_s = (r_s \vee g_s \vee b_s) \wedge \neg(r_s \wedge g_s) \wedge \neg(r_s \wedge b_s) \wedge \neg(g_s \wedge b_s)$

Une réduction polynomiale de 3-COL à SAT

Problème 3-COL

Instance Un graphe.

Question Le graphe a-t-il une 3-coloration ?

Objectif : réduction vers SAT, i.e., algorithme polynomial...

... prenant en entrée un graphe G, construisant une formule $\varphi(G)$, où

$$\varphi(G) \textbf{ satisfaisable} \iff G \textbf{ est 3-colorable}.$$

- pour chaque sommet s, on crée 3 variables r_s, g_s, b_s et une formule $\alpha_s = (r_s \vee g_s \vee b_s) \wedge \neg(r_s \wedge g_s) \wedge \neg(r_s \wedge b_s) \wedge \neg(g_s \wedge b_s)$

- Pour chaque arête $s - t$, on crée la formule $\beta_{s,t} = \neg(r_s \wedge r_t) \wedge \neg(g_s \wedge g_t) \wedge \neg(b_s \wedge b_t)$.

Une réduction polynomiale de 3-COL à SAT

Problème 3-COL

Instance Un graphe.

Question Le graphe a-t-il une 3-coloration ?

Objectif : réduction vers SAT, i.e., algorithme polynomial...

... prenant en entrée un graphe G, construisant une formule $\varphi(G)$, où

$$\varphi(G) \textbf{ satisfaisable} \iff G \textbf{ est 3-colorable}.$$

- pour chaque sommet s, on crée 3 variables r_s, g_s, b_s et une formule $\alpha_s = (r_s \vee g_s \vee b_s) \wedge \neg(r_s \wedge g_s) \wedge \neg(r_s \wedge b_s) \wedge \neg(g_s \wedge b_s)$
- Pour chaque arête $s - t$, on crée la formule $\beta_{s,t} = \neg(r_s \wedge r_t) \wedge \neg(g_s \wedge g_t) \wedge \neg(b_s \wedge b_t)$.
- La formule $\bigwedge_s \alpha_s \ \wedge \ \bigwedge_{s,t} \beta_{s,t}$ répond à notre objectif.

Une réduction polynomiale de 3-COL à SAT

Problème 3-COL

Instance Un graphe.

Question Le graphe a-t-il une 3-coloration ?

Objectif : réduction vers SAT, i.e., algorithme polynomial...

... prenant en entrée un graphe G, construisant une formule $\varphi(G)$, où

$$\varphi(G) \textbf{ satisfaisable} \iff G \textbf{ est 3-colorable}.$$

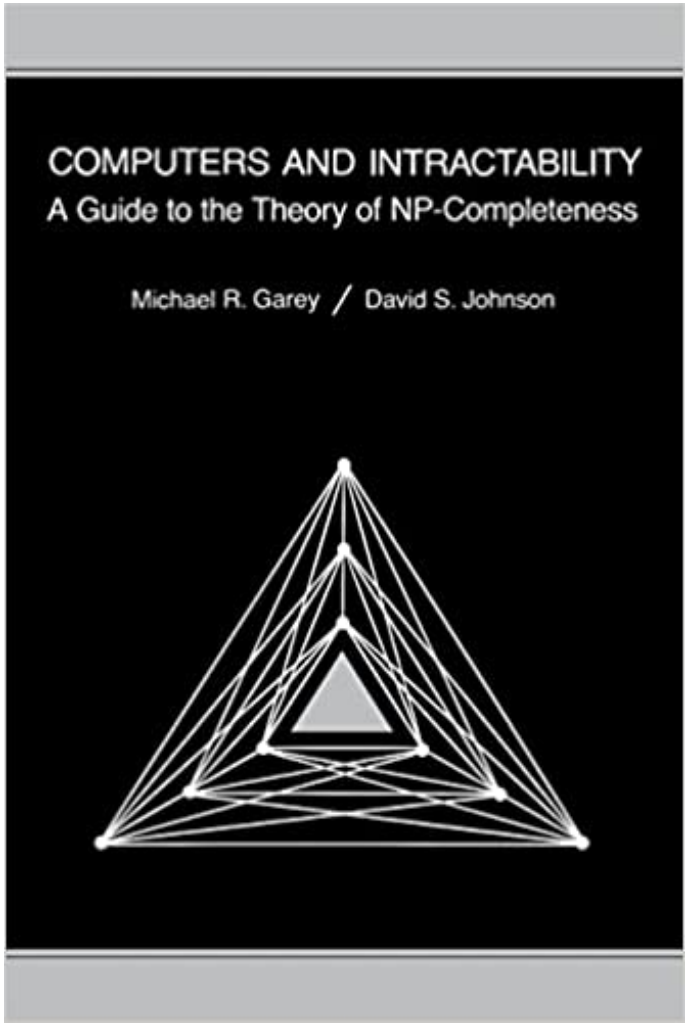
- pour chaque sommet s, on crée 3 variables r_s, g_s, b_s et une formule $\alpha_s = (r_s \vee g_s \vee b_s) \wedge \neg(r_s \wedge g_s) \wedge \neg(r_s \wedge b_s) \wedge \neg(g_s \wedge b_s)$
- Pour chaque arête $s - t$, on crée la formule $\beta_{s,t} = \neg(r_s \wedge r_t) \wedge \neg(g_s \wedge g_t) \wedge \neg(b_s \wedge b_t)$.
- La formule $\bigwedge_s \alpha_s \ \wedge \ \bigwedge_{s,t} \beta_{s,t}$ répond à notre objectif.

Cette réduction a été implémentée dans le programme `checkproperty`

Des centaines de problèmes NP-complets !

Propager la NP-completude

Si X est NP-complet, s'il se réduit à Y, et si $Y \in$ NP, alors Y est NP-complet.



Garey / Johnson, 1979, © Bell Labs

CLIQUE
INSTANCE: A graph $G = (V,E)$ and a positive integer $J \leqslant |V|$.
QUESTION: Does G contain a *clique* of size J or more, that is, a subset $V' \subseteq V$ such that $|V'| \geqslant J$ and every two vertices in V' are joined by an edge in E?

HAMILTONIAN CIRCUIT (HC)
INSTANCE: A graph $G = (V,E)$.
QUESTION: Does G contain a Hamiltonian circuit, that is, an ordering $\langle v_1, v_2, \ldots, v_n \rangle$ of the vertices of G, where $n = |V|$, such that $\{v_n, v_1\} \in E$ and $\{v_i, v_{i+1}\} \in E$ for all i, $1 \leqslant i < n$?

PARTITION
INSTANCE: A finite set A and a "size" $s(a) \in Z^+$ for each $a \in A$.
QUESTION: Is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a) \; ?$$

One reason for the popularity of these six problems is that they all appeared in the original list of 21 NP-complete problems presented in [Karp, 1972]. We shall begin our illustration of the techniques for proving NP-completeness by proving that each of these six problems is NP-complete, noting, whenever appropriate, variants of these problems whose NP-completeness follows more or less directly from that of the basic problems.

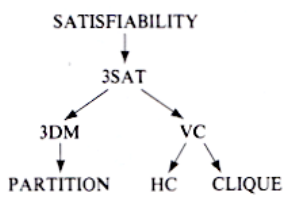


Figure 3.1 Diagram of the sequence of transformations used to prove that the six basic problems are NP-complete.

Our initial transformation will be from SATISFIABILITY, since it is the only "known" NP-complete problem we have so far. However, as we proceed through these six proofs, we will be enlarging our collection of known NP-complete problems, and all problems proved NP-complete before a problem Π will be available for use in proving that Π is NP-complete.

Problèmes NP-complets via réduction

En plus de Sudoku, Eternity, SAT, TSP...

Cycle Hamiltonien

I. Un graphe G.
Q. G a-t-il un cycle visitant chaque sommet une fois ?

Somme d'entiers

I. Des entiers $x_1, \dots, x_n$ et un entier S.
Q. Existe-t-il $I \subseteq [1; n]$ tel que $\sum_{i \in I} x_i = S$?

Partition

I. Des entiers $x_1, \dots, x_n$.
Q. Existe-t-il $I \subseteq [1; n]$ tel que $\sum_{i \in I} x_i = \sum_{i \notin I} x_i$?

3-SAT

I. Une formule propositionnelle φ en forme 3-CNF.
Q. φ est-elle satisfaisable ?

k-COL (où $k \geqslant 3$)

I. Un graphe G.
Q. G est-il k-colorable ?

Exercice 1

On admet que Cycle Hamiltonien NP-complet.

1. Trouver une réduction polynomiale de Cycle Hamiltonien à TSP.
2. Montrer que TSP est dans NP.
3. En déduire que le problème TSP est NP-complet.

Exercice 2

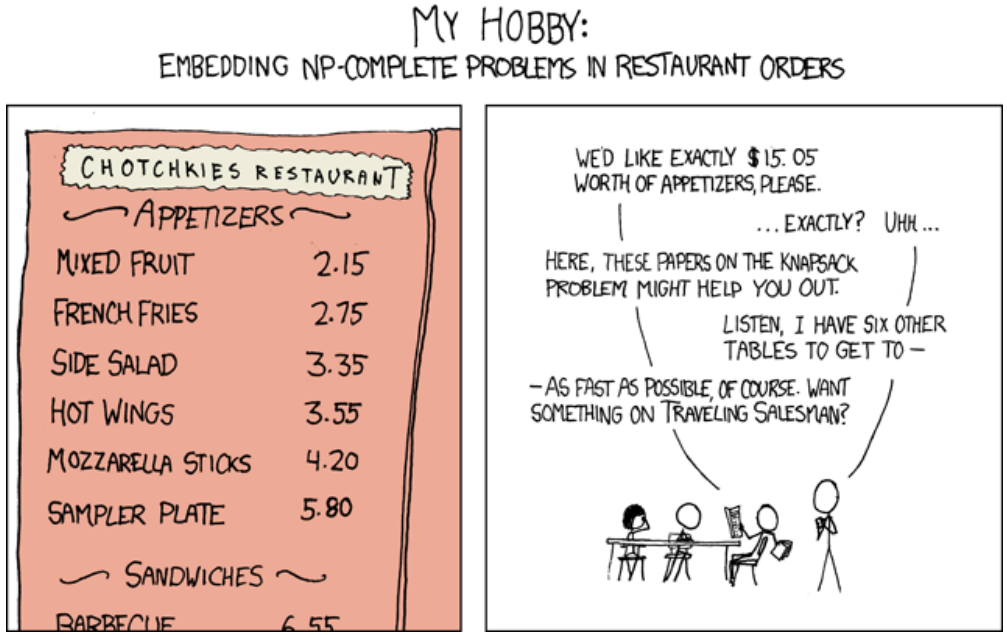
On admet que 3-COL est NP-complet.

1. Trouver une réduction polynomiale de 3-COL à 4-COL.
2. Montrer que 4-COL est dans NP.
3. En déduire que le problème 4-COL est NP-complet.
4. Généraliser au problème k-COL, pour $k \geqslant 3$ fixé.
5. Montrer que le problème 2-COL est dans **P**.

Exercice 3

On admet que Somme d'entiers est NP-complet.

1. Somme d'entiers se résout en temps $O(nS)$ (prog. dynamique). Est-ce contradictoire avec le fait qu'il est NP-complet ?
2. Trouver une réduction polynomiale de Somme d'entiers à Partition.
3. Montrer que Partition est dans NP.
4. En déduire que le problème Partition est NP-complet.



Que faire avec un problème NP-complet ?

- Réduire vers SAT et utilisation d'un solveur.

Citation de Moshe Vardi, bis (lien)

> *When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.*
>
> *[...]*
>
> *But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!*

Que faire avec un problème NP-complet ?

- Réduire vers SAT et utilisation d'un solveur.

Citation de Moshe Vardi, bis (lien)

> *When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.*
>
> *[...]*
>
> *But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!*

- Approximer $\rightsquigarrow$ algorithme polynomial avec garantie de qualité.

Que faire avec un problème NP-complet ?

- Réduire vers SAT et utilisation d'un solveur.

Citation de Moshe Vardi, bis (lien)

> *When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.*
>
> *[...]*
>
> *But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!*

- Approximer $\rightsquigarrow$ algorithme polynomial avec garantie de qualité.
- Utiliser des algorithmes randomisés.

Que faire avec un problème NP-complet ?

- Réduire vers SAT et utilisation d'un solveur.

Citation de Moshe Vardi, bis (lien)

> *When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.*
>
> *[...]*
>
> *But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!*

- Approximer $\rightsquigarrow$ algorithme polynomial avec garantie de qualité.
- Utiliser des algorithmes randomisés.
- Restreindre le problème à une sous-classe d'entrées.

Que faire avec un problème NP-complet ?

- Réduire vers SAT et utilisation d'un solveur.

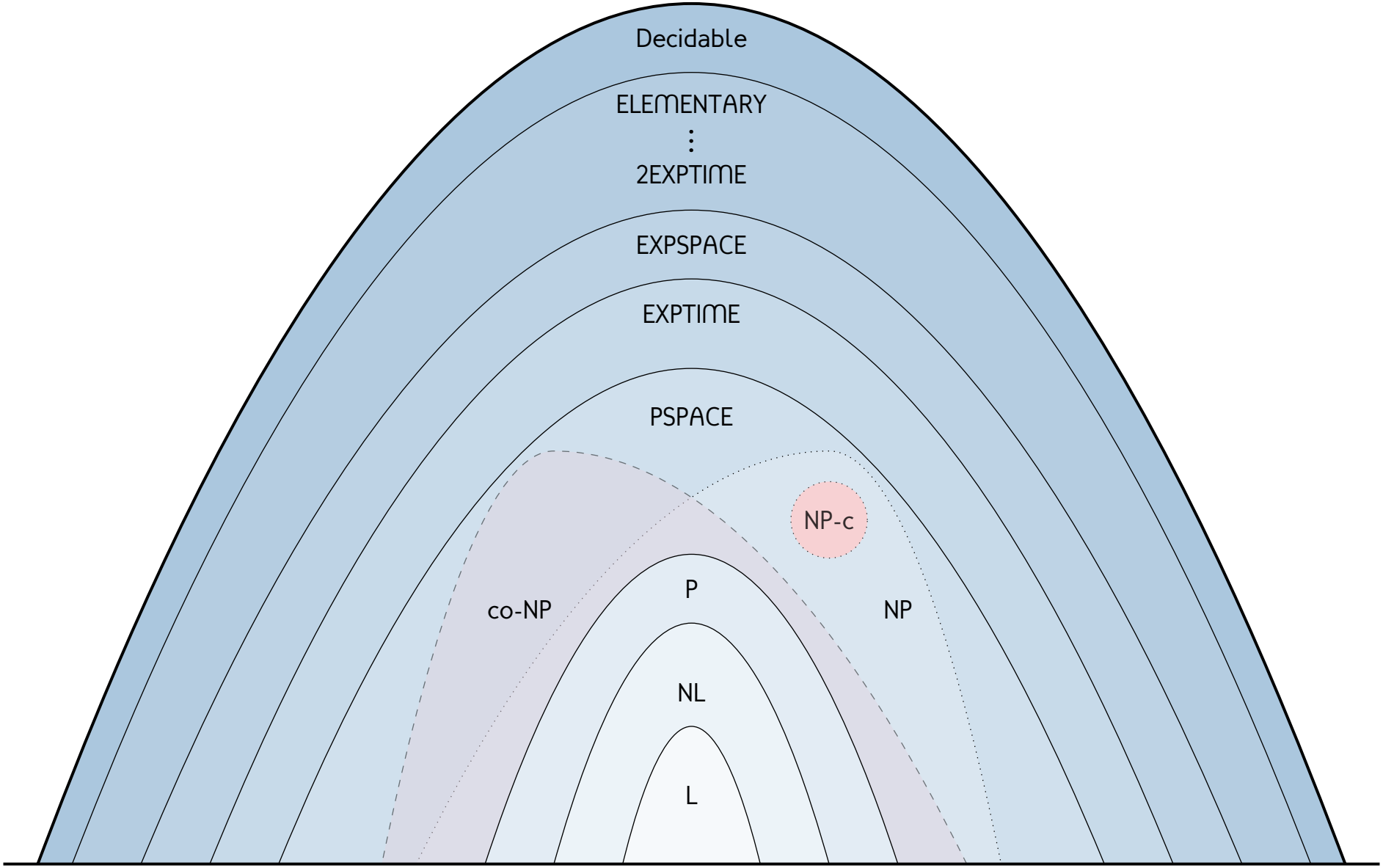
Citation de Moshe Vardi, bis (lien)

> *When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.*
>
> *[...]*
>
> *But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!*

- Approximer $\rightsquigarrow$ algorithme polynomial avec garantie de qualité.
- Utiliser des algorithmes randomisés.
- Restreindre le problème à une sous-classe d'entrées.
- Utiliser des heuristiques (moins satisfaisant).

De nombreuses classes de complexité !

Quelques unes des classes les plus courantes.



Des problèmes très difficiles

Problème d'accessibilité par addition de vecteurs

Instance
- Un entier d,
- Un ensemble fini de vecteurs $\mathsf{A} = \{\vec{a}_1, \dots, \vec{a}_m\} \subset \mathbb{Z}^d$
- Deux vecteurs $\vec{y}, \vec{z} \in \mathbb{N}^d$.

Question Existe-t-il une suite finie $\vec{x}_1, \dots, \vec{x}_k$ de vecteurs de A telle que :
- $\vec{z} = \vec{y} + \sum_{i=1}^{k} \vec{x}_i$, et
- toutes les sommes partielles $\vec{y} + \sum_{i=1}^{\ell} \vec{x}_i$ sont dans $\mathbb{N}^d$?

Des problèmes très difficiles

Problème d'accessibilité par addition de vecteurs

Instance
- Un entier d,
- Un ensemble fini de vecteurs $\mathsf{A} = \{\vec{a}_1, \dots, \vec{a}_m\} \subset \mathbb{Z}^d$
- Deux vecteurs $\vec{y}, \vec{z} \in \mathbb{N}^d$.

Question Existe-t-il une suite finie $\vec{x}_1, \dots, \vec{x}_k$ de vecteurs de A telle que :
- $\vec{z} = \vec{y} + \sum_{i=1}^{k} \vec{x}_i$, et
- toutes les sommes partielles $\vec{y} + \sum_{i=1}^{\ell} \vec{x}_i$ sont dans $\mathbb{N}^d$?

Théorème [Mayr, 1981]

Ce problème est décidable.

Des problèmes très difficiles

Problème d'accessibilité par addition de vecteurs

Instance
- Un entier d,
- Un ensemble fini de vecteurs $\mathsf{A} = \{\vec{a}_1, \dots, \vec{a}_m\} \subset \mathbb{Z}^d$
- Deux vecteurs $\vec{y}, \vec{z} \in \mathbb{N}^d$.

Question Existe-t-il une suite finie $\vec{x}_1, \dots, \vec{x}_k$ de vecteurs de A telle que :
- $\vec{z} = \vec{y} + \sum_{i=1}^{k} \vec{x}_i$, et
- toutes les sommes partielles $\vec{y} + \sum_{i=1}^{\ell} \vec{x}_i$ sont dans $\mathbb{N}^d$?

Théorème [Mayr, 1981]

Ce problème est décidable.

Théorème [Czerwinski, Lasota, Lazic, Leroux, Mazowiecki, 2019]

Tout algorithme résolvant ce problème coûte dans le cas le pire, environ

$$\left.2^{2^{2^{\cdots^{2}}}}\right\} \text{hauteur } n \qquad (n = \text{ taille de l'entrée})$$

Bonne rentrée !