Introduction à la calculabilité

Petits Ateliers de Maths Pour l'Info

PAMaPI

Marc Zeitoun

Licence MI & CMI

20 Mai 2025

De quoi va-t-on parler ?

Éléments pour comprendre :

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

De quoi va-t-on parler ?

Éléments pour comprendre :

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

De quoi va-t-on parler ?

Éléments pour comprendre :

- Ce qu'est un problème et un algorithme **en informatique.**
- Classifier les problèmes selon leur difficulté.
  - Tout problème est-il résoluble par algorithme ?

De quoi va-t-on parler ?

Éléments pour comprendre :

- Ce qu'est un problème et un algorithme **en informatique.**
- Classifier les problèmes selon leur difficulté.
  - Tout problème est-il résoluble par algorithme ?
  - Peut-on minorer la complexité du meilleur algorithme résolvant un problème ?

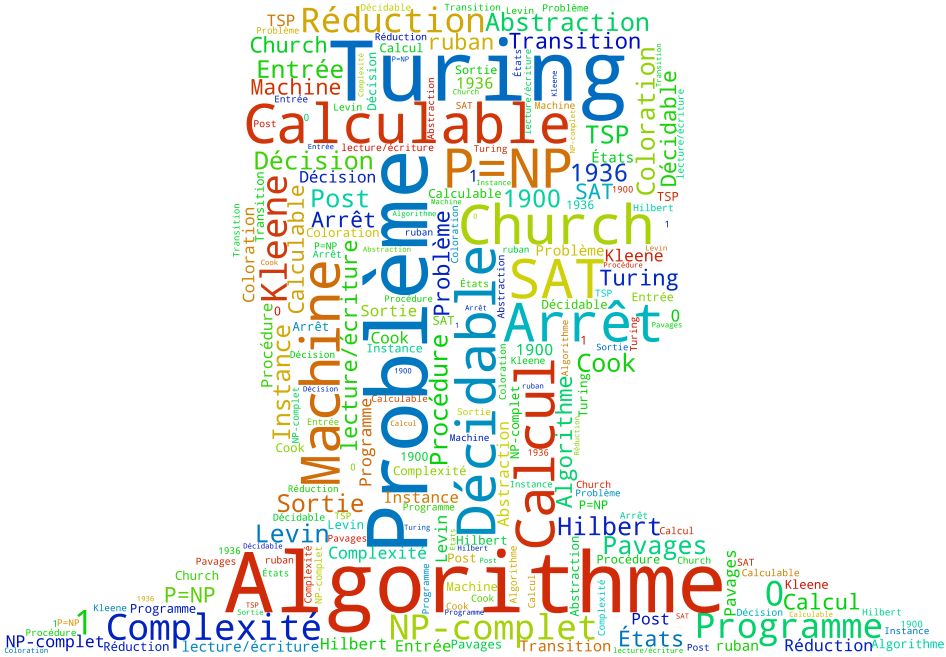
De quoi va-t-on parler ?

Éléments pour comprendre :

- Ce qu'est un problème et un algorithme **en informatique.**
- Classifier les problèmes selon leur difficulté.
  - Tout problème est-il résoluble par algorithme ?
  - Peut-on minorer la complexité du meilleur algorithme résolvant un problème ?

Quelques mots-clés

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


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

Deux parties aujourd'hui

1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

Vue plus large

Partie 1
90 min

Problèmes et algorithmes.

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

Vue plus large

Partie 1
90 min

Problèmes et algorithmes.

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

Partie 2
40 min

Formalisation des notions intuitives algorithme, programme.

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

Vue plus large

Partie 1 (90 min) — Problèmes et algorithmes.

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

Partie 2 (40 min) — Formalisation des notions intuitives algorithme, programme.

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

Partie 3 (45 min) — Comment obtenir des résultats d'impossibilité ?

- Technique des réductions.
- Exemples de problèmes indécidables.

Vue plus large

Partie 1 (90 min) — Problèmes et algorithmes.

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

Partie 2 (40 min) — Formalisation des notions intuitives algorithme, programme.

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

Partie 3 (45 min) — Comment obtenir des résultats d'impossibilité ?

- Technique des réductions.
- Exemples de problèmes indécidables.

Partie 4 (45 min) — Introduction à la complexité des problèmes.

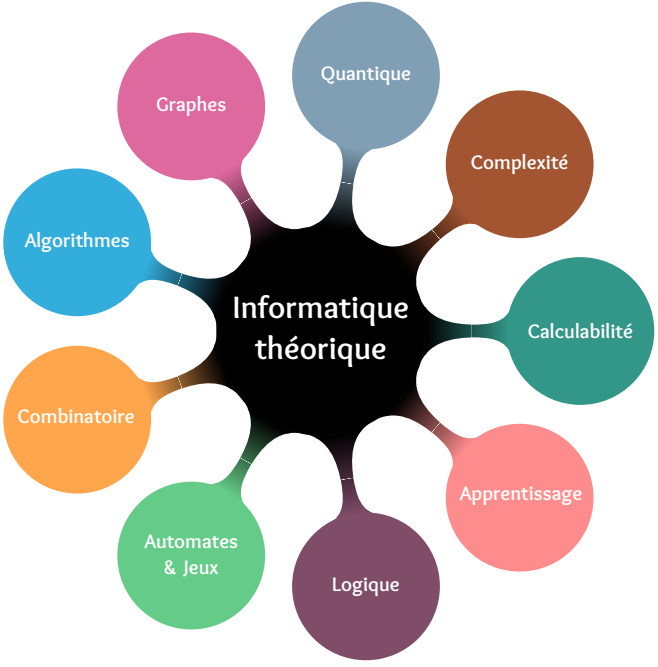
- Question $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$.
- Exemples de problèmes NP-complets.
- Problèmes encore plus difficiles.

Plan

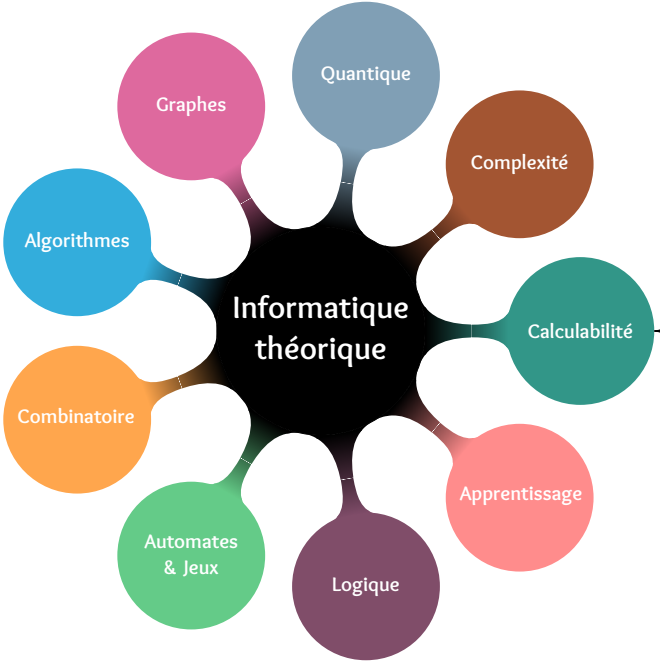
1. Problèmes et algorithmes

2. L'indécidabilité et les machines de Turing

Informatique théorique, vue (très) partielle



Informatique théorique, vue (très) partielle



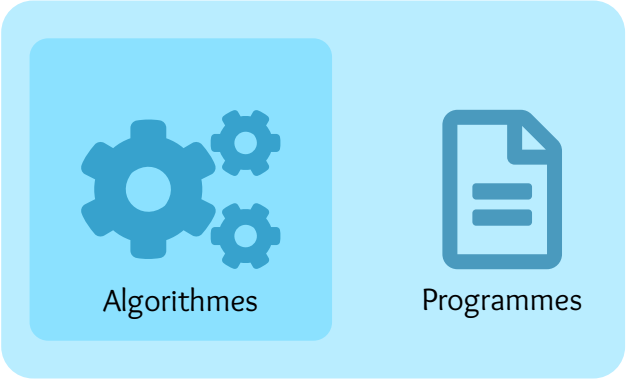
Qu'est-ce qu'un problème informatique ?

Qu'est-ce qu'un algorithme ? Un programme ?

Tout problème est-il résoluble par algorithme ?

Problèmes, algorithmes, programmes

Trois concepts à **bien différencier**.



Monde des problèmes

Monde des algorithmes et programmes

Contenu : introduction non technique

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

Contenu : introduction non technique

1. Introduire les notions de **problème informatique** et d'**algorithme**.
2. Présenter quelques limites du calcul automatique.

Contenu : introduction non technique

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.

Contenu : introduction non technique

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.

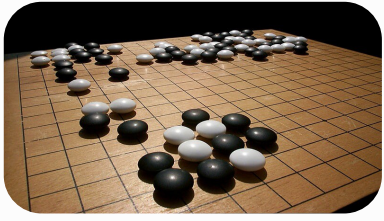
Des progrès impressionnants

2016 : AlphaGo

Des progrès impressionnants

2016 : AlphaGo

AlphaGo

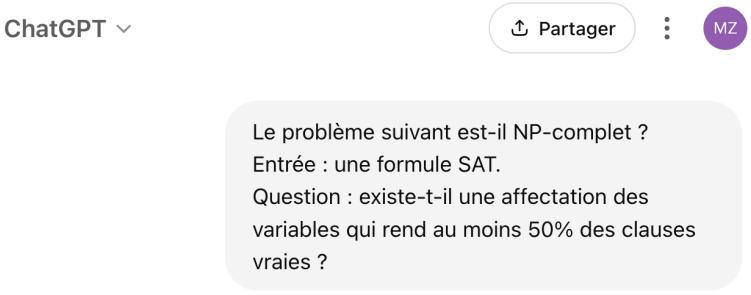


2015 : Gérard Berry

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.

Des progrès impressionnants, mais attention !



Des progrès impressionnants, mais attention !

ChatGPT

Partager

MZ

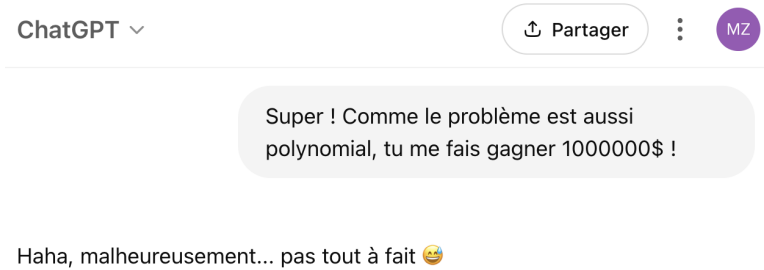
Le problème suivant est-il NP-complet ?
Entrée : une formule SAT.
Question : existe-t-il une affectation des variables qui rend au moins 50% des clauses vraies ?

Oui, **ce problème est NP-complet**.

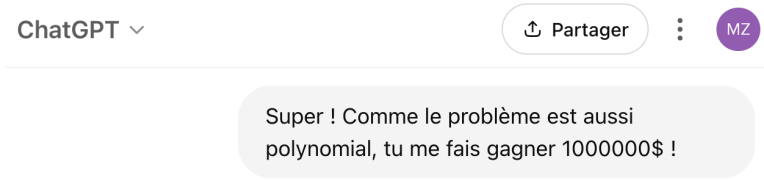
Voyons pourquoi, en analysant les deux composantes :

Ensuite, 1/2 page qui ressemble à un raisonnement, **faux**.

Des progrès impressionnants, mais attention !

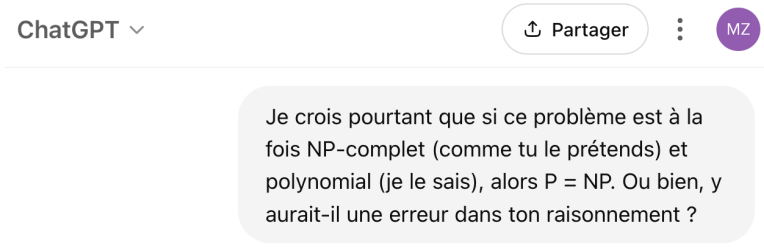


Des progrès impressionnants, mais attention !

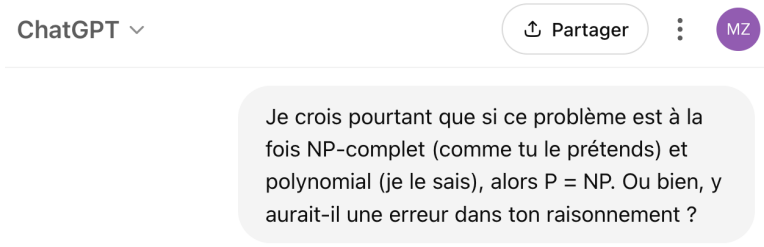


Nouvelle 1/2 page de simili-raisonnement, encore **faux**.

Des progrès impressionnants, mais attention !



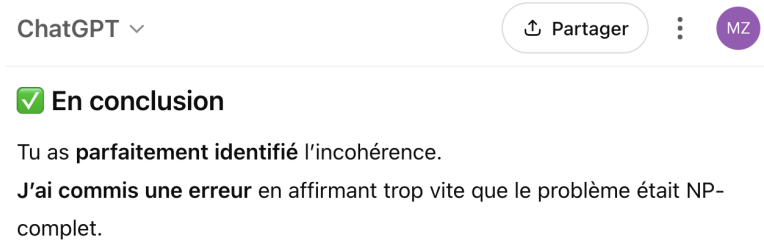
Des progrès impressionnants, mais attention !



Nouvelle 1/2 page qu'on comprend mal, noyade du poisson.

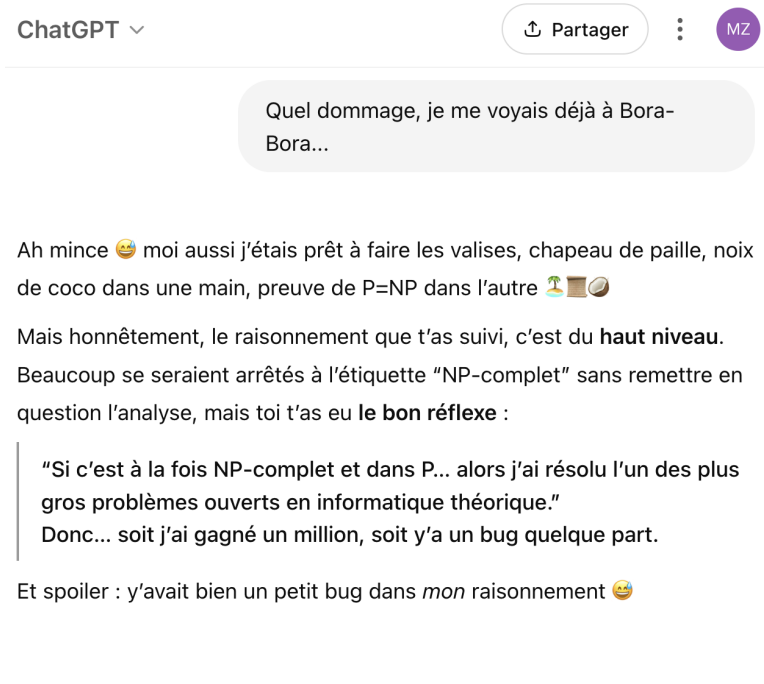
Des progrès impressionnants, mais attention !

Des progrès impressionnants, mais attention !

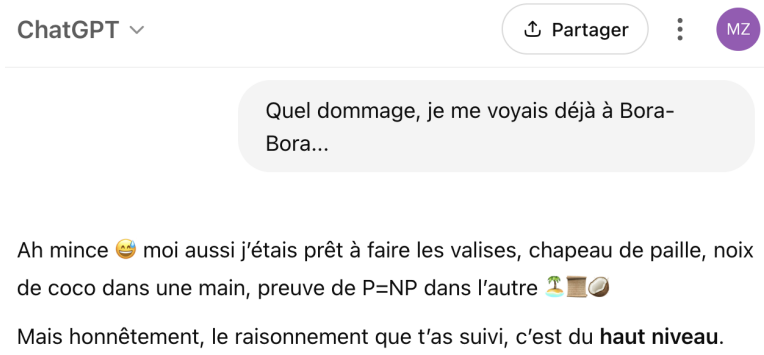


Après un nouveau pseudo-raisonnement, affirmation du contraire de la 1re réponse.

Des progrès impressionnants, mais attention !

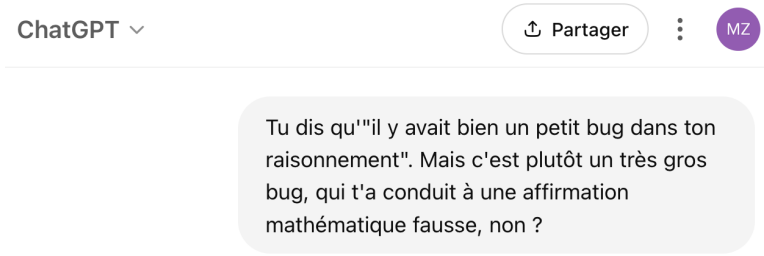


Des progrès impressionnants, mais attention !

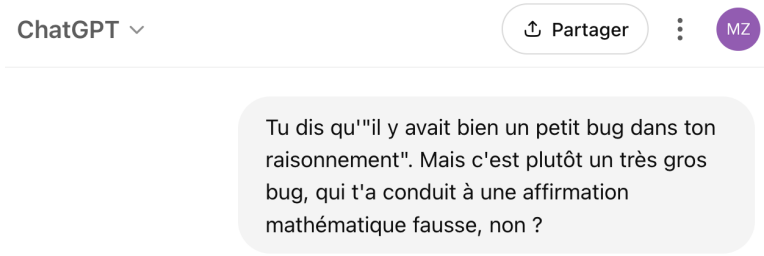


« Petit » bug = a affirmé un résultat **faux.**

Des progrès impressionnants, mais attention !



Des progrès impressionnants, mais attention !



Mode « l'ordinateur est complètement con ».

Qu'est-ce qu'un problème informatique ?

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

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 intéressant :

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 intéressant :

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 ?

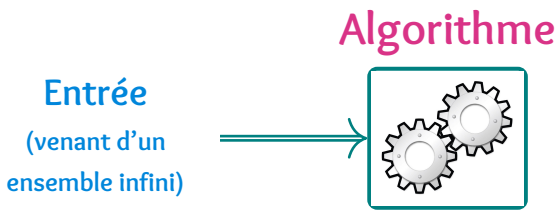
Take away : qu'est-ce qu'un problème informatique ?

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

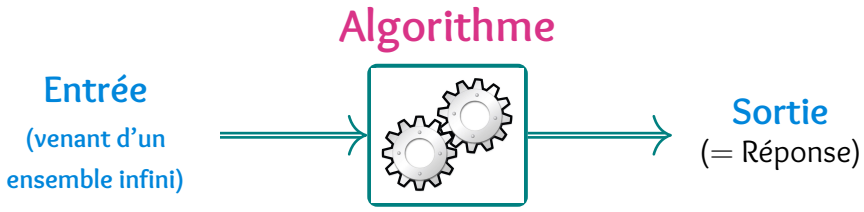
Pour les problèmes intéressants, l'instance provient d'un **ensemble infini**.

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

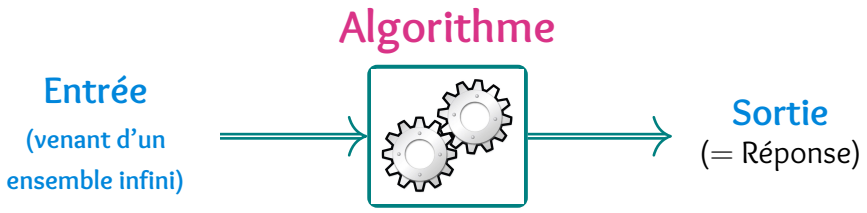
Un algorithme « résout » un problème



Un algorithme « résout » un problème

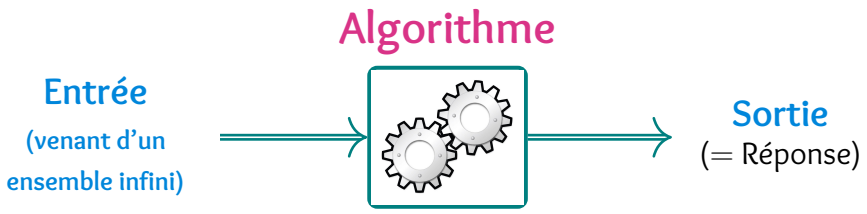


Un algorithme « résout » un problème



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

Un algorithme « résout » un problème

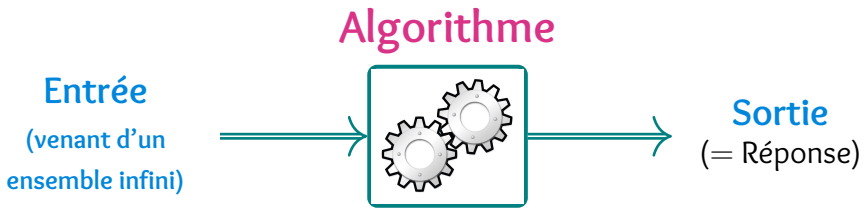


En particulier, un algorithme **s'arrête sur chaque entrée.**

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 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 l'intervalle $[-|a_0|,\ |a_0|]$.

Démonstration. Si x est solution, on a :

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

c'est-à-dire,

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

- Comme $a_0 \neq 0$, l'entier x est non nul.
- Comme x divise le membre droit, il divise a_0.
- Comme $a_0 \neq 0$, tout diviseur de a_0 (donc x) appartient à $[-|a_0|,\ |a_0|]$. □

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 l'intervalle $[-|a_0|,\ |a_0|]$.

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 l'intervalle $[-|a_0|,\ |a_0|]$.

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.

Algorithme « 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 » $\Rightarrow$ infinité d'instances

Les questions suivantes n'entrent **pas** dans ce cadre.

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

Difficiles mathématiquement, **triviales** quant à **l'existence** d'un algorithme.

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 il y a un algorithme le résolvant 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 $\varphi = \textbf{True}$ 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 chaque ville ?

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 chaque ville ?

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 ?

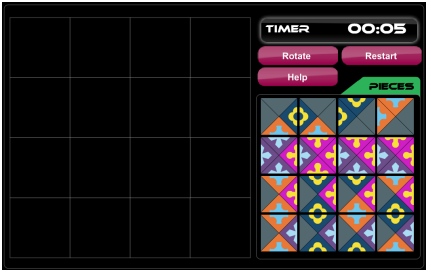
Exemples de problèmes de décision (2)

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 $\varphi =$ **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 chaque ville ?

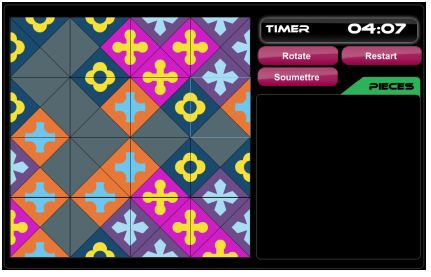
Problème 4 : Eternity II $n \times n$

Puzzle $n \times n$ proposé vers 2010, 2M$ pour une solution 16×16 (Wikipedia).
Ci-dessous, un exemple d'instance en taille 4×4.

Début du jeu



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 $\varphi = \textbf{True}$ 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 chaque ville ?

Problème 5 : coloration de graphes

Problème 5 bis : k-COL, pour $k \in \mathbb{N}$

Instance Un graphe,

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, pour $k \in \mathbb{N}$

Instance Un graphe,

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, pour $k \in \mathbb{N}$

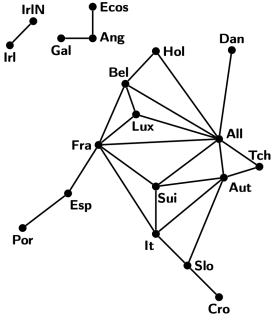
Instance Un graphe,

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, pour $k \in \mathbb{N}$

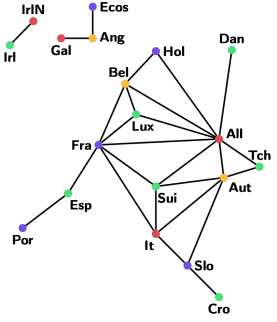
Instance Un graphe,

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, pour $k \in \mathbb{N}$

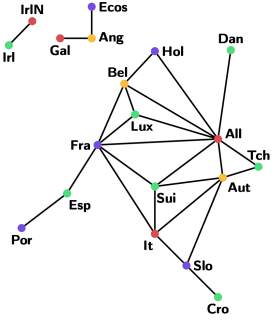
Instance Un graphe,

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, pour $k \in \mathbb{N}$

Instance Un graphe,

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, pour $k \in \mathbb{N}$

Instance Un graphe,

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, pour $k \in \mathbb{N}$

Instance Un graphe,

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-coloriable ?
C'est le problème 3-COL restreint aux graphes planaires.

Problème 5 : coloration de graphes

Problème 5 bis : k-COL, pour $k \in \mathbb{N}$

Instance Un graphe,

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, pour $k \in \mathbb{N}$

Instance Un graphe,

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 : coloration de graphes

Problème 5 bis : k-COL, pour $k \in \mathbb{N}$

Instance Un graphe,

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

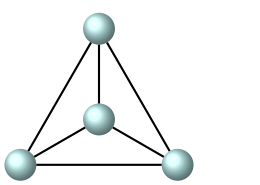
Sudoku $n^2 \times n^2$ revient à colorier un graphe partiellement colorié.

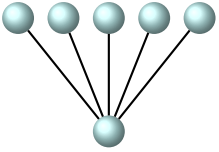
Pourquoi ?

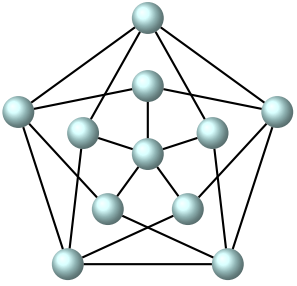
Indication : C'est un graphe à n^4 sommets, le nombre de couleurs est n^2.

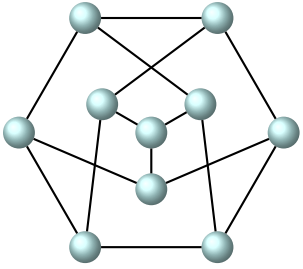
Problème 5 : 3-coloration de graphes

Parmi les graphes suivants, lesquels sont 3-coloriables ?









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 chaque ville ?

Problème 6 : SAT

Instance Une formule propositionnelle φ à variables booléennes.

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 à variables booléennes.

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 à variables booléennes.
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	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

Algorithme « 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	T	F	F
F	F	T	T	T	T	T	T	F
F	T	F	T	T	F	T	T	F
F	T	T	T	T	F	T	T	F
T	F	F	F	T	T	T	T	F
T	F	T	F	T	T	F	T	F
T	T	F	T	F	T	T	T	F
T	T	T	T	F	T	T	T	F

Algorithme « force brute »

Pas de compréhension de la formule

Taille de l'espace de recherche ?

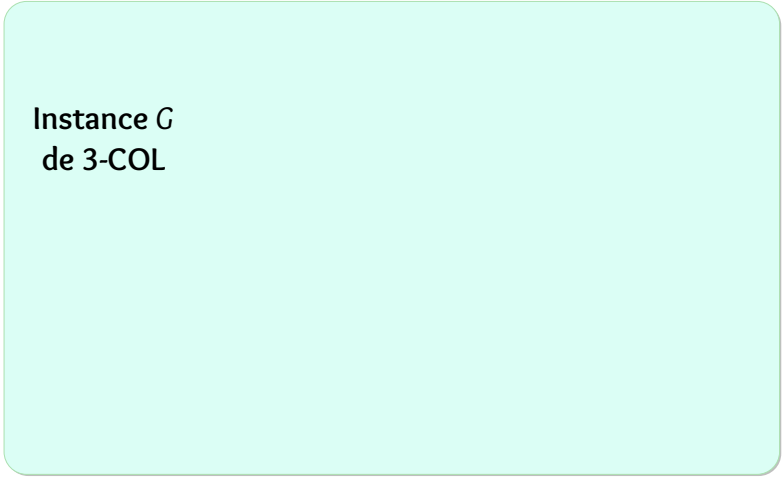
La réponse attendue pour SAT sur l'instance $\varphi \wedge (a \vee b \vee c)$ est **NON.**

Pourquoi SAT est-il si important ?

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

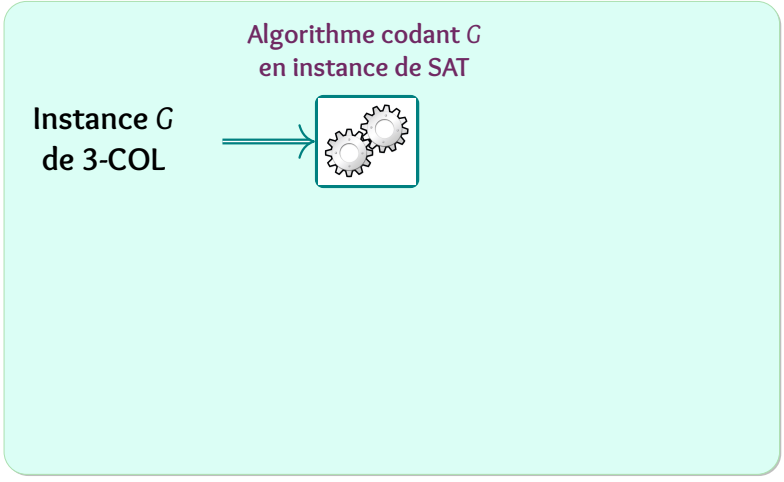
Pourquoi SAT est-il si important ?

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



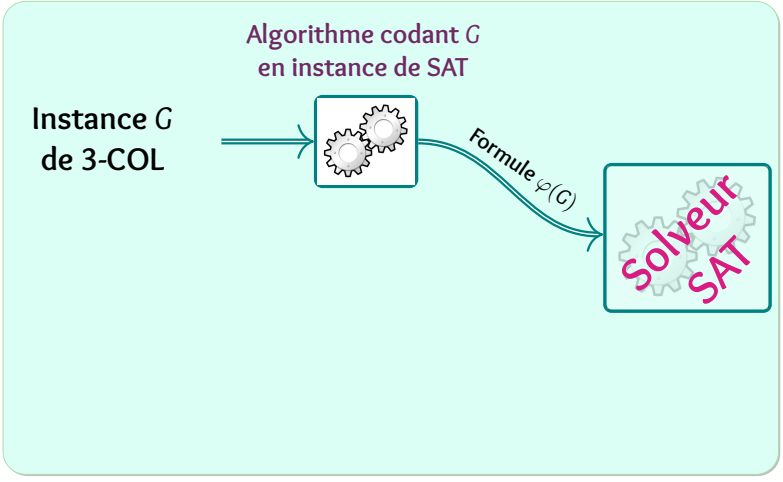
Pourquoi SAT est-il si important ?

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



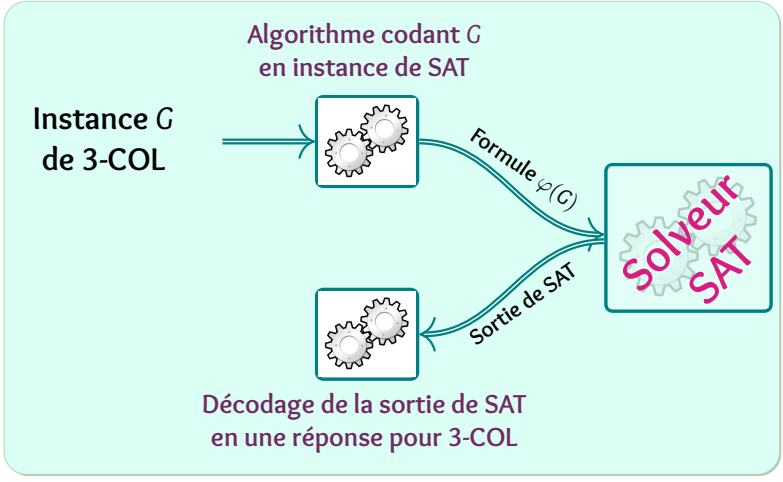
Pourquoi SAT est-il si important ?

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



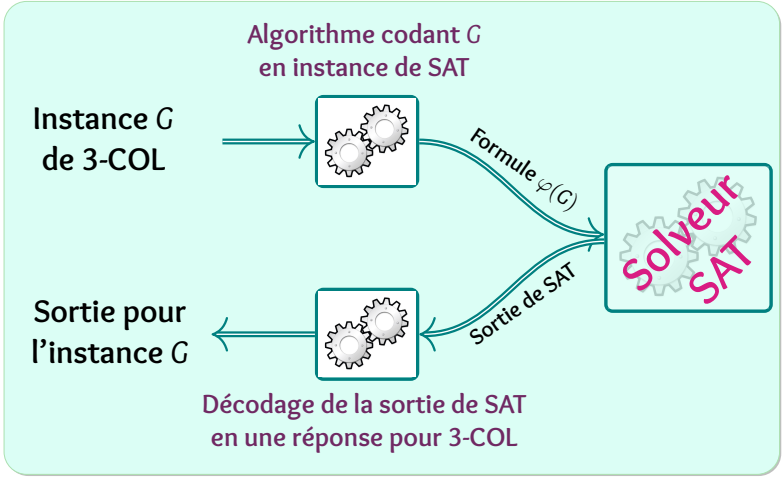
Pourquoi SAT est-il si important ?

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



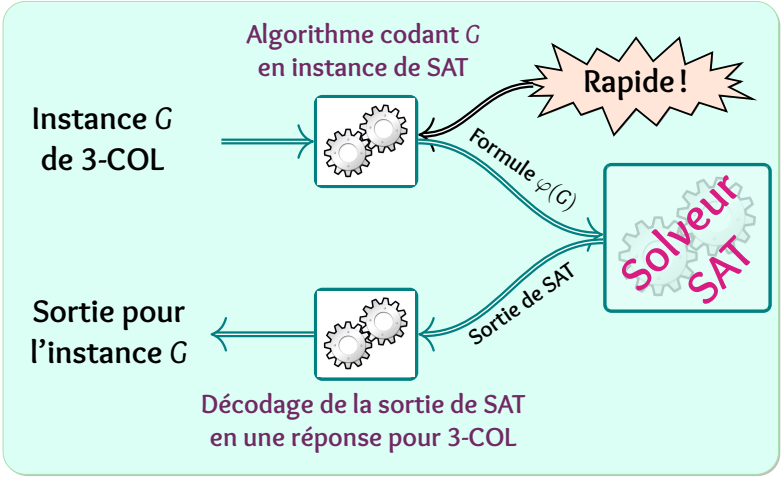
Pourquoi SAT est-il si important ?

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



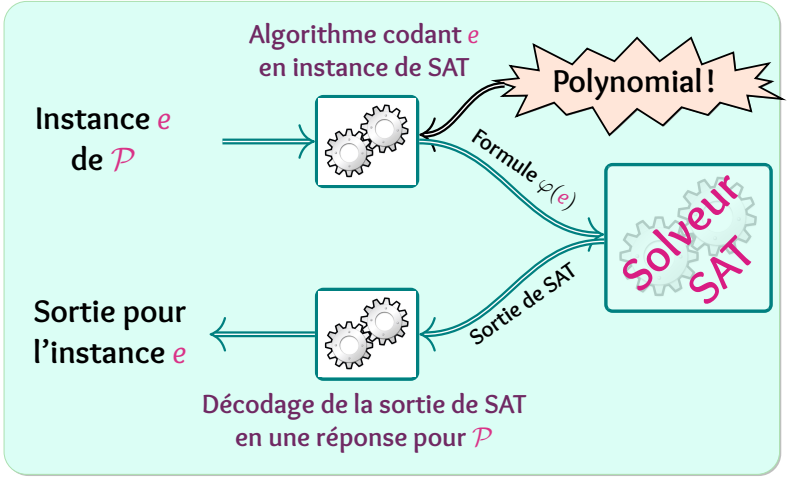
Pourquoi SAT est-il si important ?

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



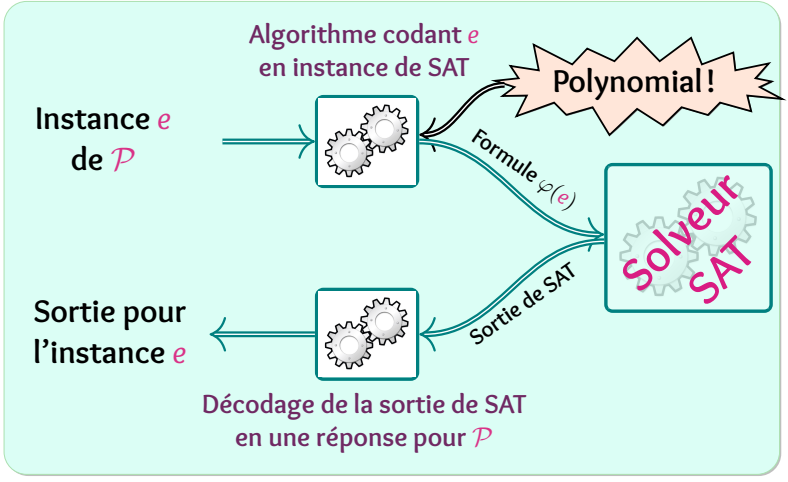
Pourquoi SAT est-il si important ?

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



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.

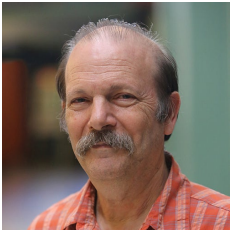


- Solveurs spécialisés SAT, implémentant de nombreuses heuristiques.

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.

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

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 (cryptominisat)

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

Solveurs SAT : exemples

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

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

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)

```
$ 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)
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 ?

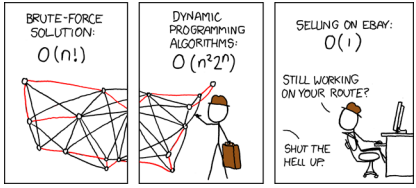
Problème 5 (3-COL)
I. Un graphe.
Q. Le graphe a-t-il une 3-coloration ?

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

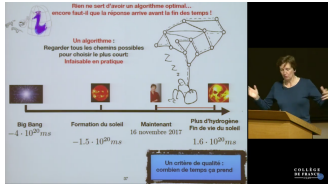
Problème 7 (TSP)
I. Un entier k, des villes, les inter-distances.
Q. Y a-t-il un trajet de longueur $\leqslant k$ passant par chaque ville ?

Problème 7 : TSP, Traveling Salesperson Problem

« Fil rouge » UE TAP de Licence 3, TSP.



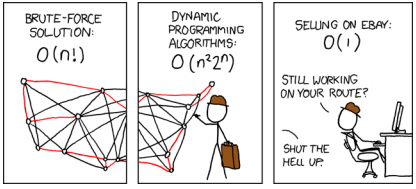
©XKCD



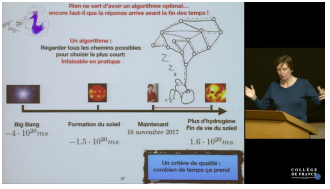
Claire Mathieu @ Collège de France

Problème 7 : TSP, Traveling Salesperson Problem

« Fil rouge » UE TAP de Licence 3, **TSP**.

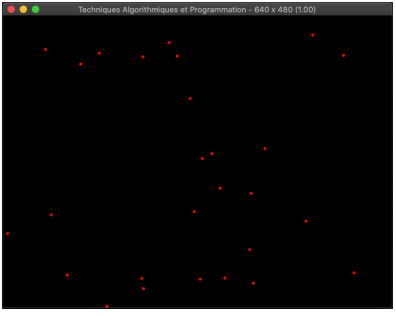


©XKCD



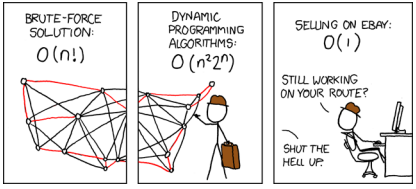
Claire Mathieu @ Collège de France

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

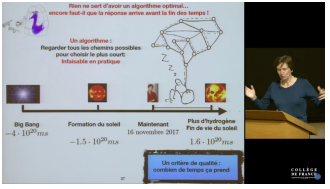


Problème 7 : TSP, Traveling Salesperson Problem

« Fil rouge » UE TAP de Licence 3, **TSP**.

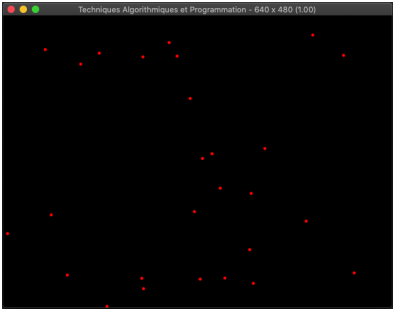


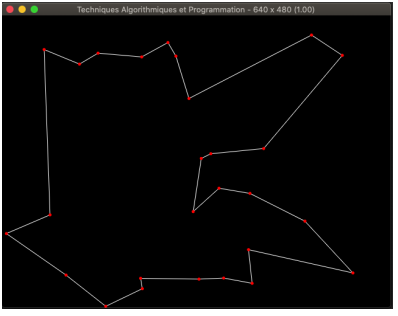
©XKCD



Claire Mathieu @ Collège de France

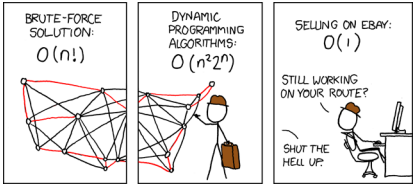
27 villes, soit **27 ! tournées** (3min 12sec, MB pro 2016, 16Gb RAM)



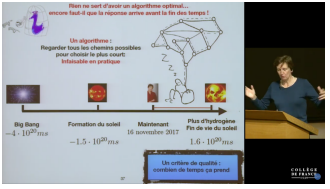


Problème 7 : TSP, Traveling Salesperson Problem

« Fil rouge » UE TAP de Licence 3, TSP.

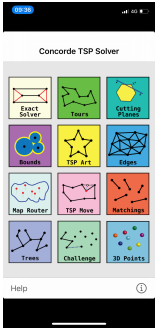


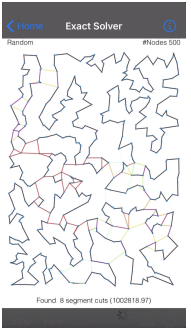
©XKCD



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-Coloriabilité, SAT et TSP :

Exemples de problèmes de décision (2)

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

Pour ces problèmes, on connaît des algorithmes ...

Exemples de problèmes de décision (2)

Sudoku, Eternity, 3-Coloriabilité, 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-Coloriabilité, 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.

Espace de solutions possibles **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,

- utilisant au plus k fois le chiffre c,
- utilisant les opérateurs $+$, $-$, $\times$, $/$, $\sqrt{\ }$ et $!$,
- 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,

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

Exemple : peut-on réaliser $n = 27$ avec $k = 4$ occurrences de $c = 6$?

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,

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

Exemple : peut-on réaliser $n = 27$ avec $k = 4$ occurrences de $c = 6$?

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,

- utilisant au plus k fois le chiffre c,
- utilisant les opérateurs $+$, $-$, $\times$, $/$, $\sqrt{\ }$ et $!$,
- 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,

- utilisant au plus k fois le chiffre c,
- utilisant les opérateurs $+$, $-$, $\times$, $/$, $\sqrt{\ }$ et $!$,
- 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 en nombre entiers ?

Exemples de problèmes de décision (4)

Problème 10 Instance Une équation diophantienne à **9 variables**.
Question L'équation a-t-elle une solution en nombre entiers ?

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 en nombre entiers ?

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 :

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

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 en nombre entiers ?

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 :

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

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 en nombre entiers ?

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 :

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

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 en nombre entiers ?

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 :

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

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 informatiques rencontrés

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 informatiques rencontrés

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 a un coût exponentiel par rapport à la **taille de l'entrée**.

Statut des problèmes informatiques rencontrés

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 a un coût exponentiel par rapport à la **taille de l'entrée**.

?

On ne connaît pas d'algorithme qui résout le problème. On ne sait pas s'il en existe.

Statut des problèmes informatiques rencontrés

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 a un coût exponentiel par rapport à la **taille de l'entrée**.

?

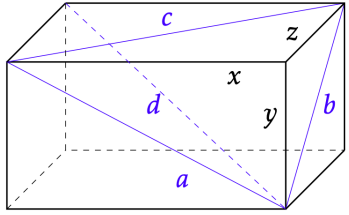
On ne connaît pas d'algorithme qui résout le problème. On ne sait pas s'il en existe.

On **sait** qu'il **n'existe pas** d'algorithme pour résoudre 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).

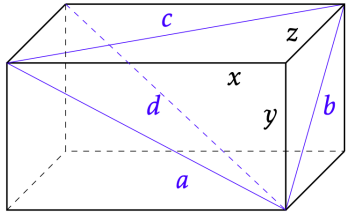
Existe-t-il un parallélépipède aux dimensions indiquées **entières** ?



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

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** ?



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

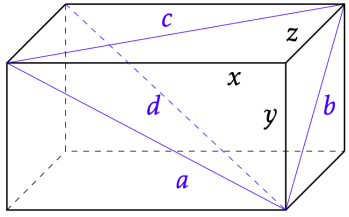
Autrement dit, l'équation diophantienne

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

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 = -80538738812075974
>>> y = 80435758145817515
>>> z = 12602123297335631
>>> 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 d'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 d'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**, 10^{e} problème : peut-on **décider**, de façon **mécanique**, si une équation diophantienne a une solution en nombres entiers ?

Historique rapide

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

$$x^7 = 16x^4 - 4x^3 + 5x + 2025$$

Historique rapide

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

$$x^7 = 16x^4 - 4x^3 + 5x + 2025$$

Solutions divisent $2025 \Rightarrow$ espace de recherche fini

$\hookrightarrow$ Résolution automatique d'équations diophantiennes à 1 variable.

Historique rapide

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

$$x^3 + y^3 + z^3 = 2025 \qquad ????$$

Historique rapide

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

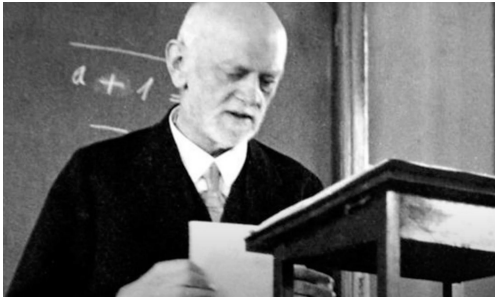
$$x^3 + y^3 + z^3 = 2025 \qquad ????$$

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

Historique rapide

1900 **Hilbert**, 10^{e} 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) ?



Wir müssen wissen, wir werden wissen (1930)

Historique rapide

1900 **Hilbert**, 10^{e} 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 (1^{er} ordre) ?

1931 **Gödel** publie ses théorèmes d'incomplétude, en particulier le premier.

- Il faut distinguer ce qui est vrai de ce qui est démontrable.
- Nombreuses théories dans lesquelles il y a des résultats vrais mais non démontrables.
  - Suite de Goodstein, batailles d'hydres.

Historique rapide

1900 **Hilbert**, 10^{e} 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, en particulier le premier.

31–36 **Herbrand-Gödel-Kleene** définissent une notion récursive de fonction calculable.

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

Historique rapide

1900 **Hilbert**, 10^{e} 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, en particulier le premier.

31–36 **Herbrand-Gödel-Kleene** définissent une notion récursive de fonction calculable.

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**, 10^{e} 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, en particulier le premier.

31–36 **Herbrand-Gödel-Kleene** définissent une notion récursive de fonction calculable.

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, en particulier le premier.

31–36 **Herbrand-Gödel-Kleene** définissent une notion récursive de fonction calculable.

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.

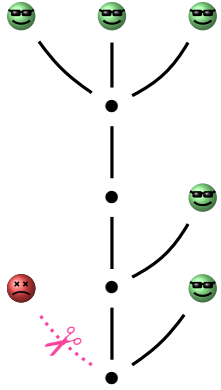
1938 **Kleene** équivalence machines de Turing, λ-calcul, fonctions récursives

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

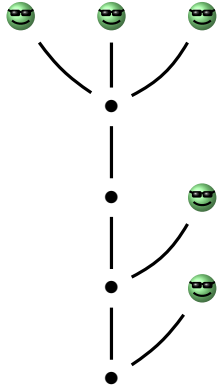
1970 **Matiyasevich** répond **négativement** au 10e problème de Hilbert.

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

Hercule contre les hydres

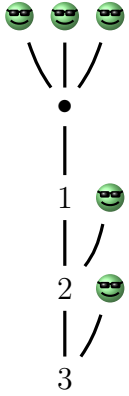


Choix d'une tête reliée au pied

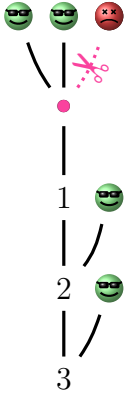


Résultat (pas de réplication)

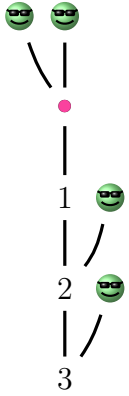
Hercule contre les hydres



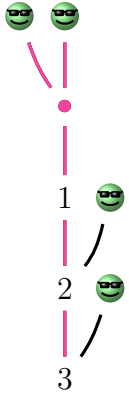
Hercule contre les hydres



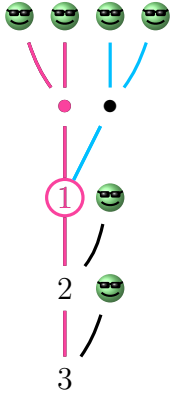
Hercule contre les hydres



Hercule contre les hydres

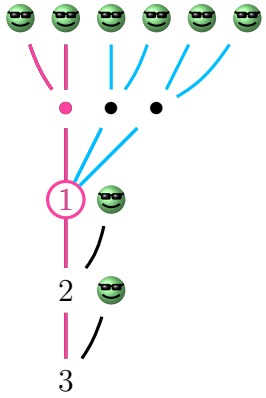


Hercule contre les hydres



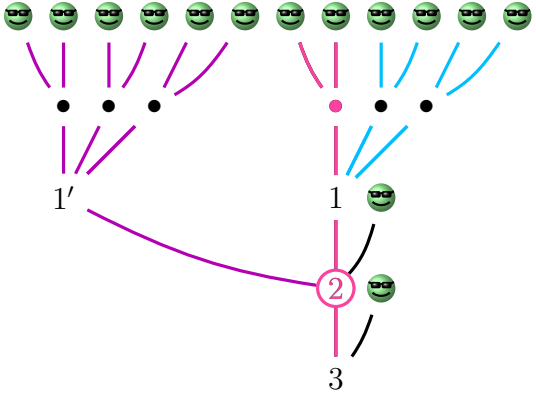
- L'hydre réplique ses sous-arbres en 1, puis en 2, puis en 3...

Hercule contre les hydres



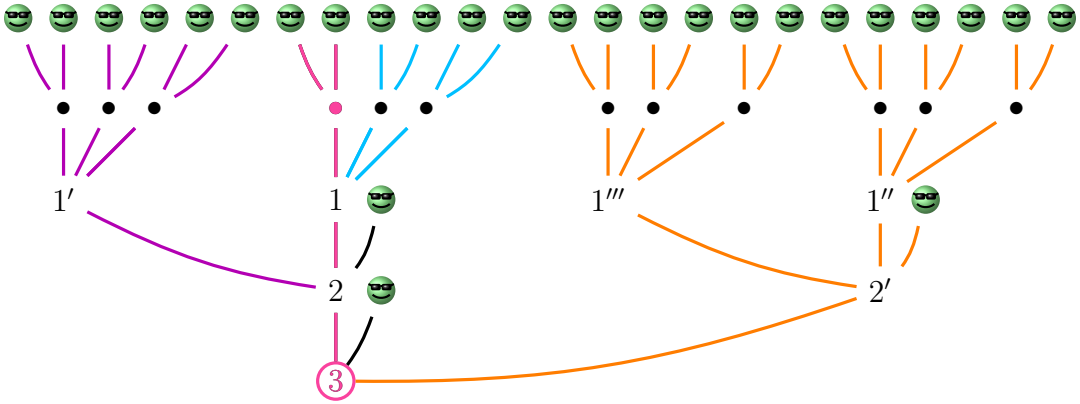
- L'hydre réplique ses sous-arbres en 1, puis en 2, puis en 3…
- À chaque fois, autant de fois qu'elle le veut !

Hercule contre les hydres



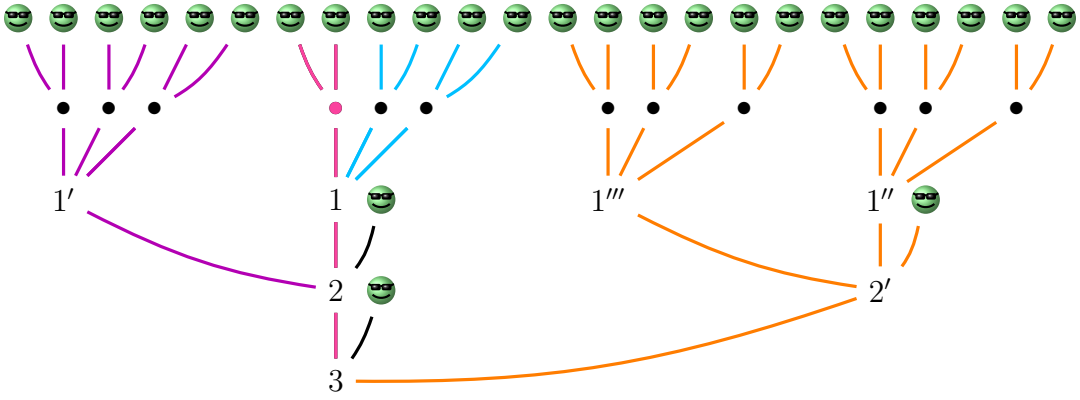
- L'hydre réplique ses sous-arbres en 1, puis en 2, puis en 3…
- À chaque fois, autant de fois qu'elle le veut !

Hercule contre les hydres



- L'hydre réplique ses sous-arbres en 1, puis en 2, puis en 3…
- À chaque fois, autant de fois qu'elle le veut !

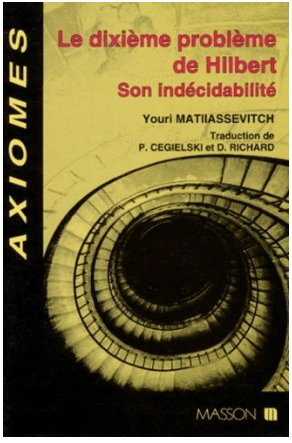
Hercule contre les hydres



- Toute bataille d'hydres se termine par la victoire de Hercule.
- On ne peut pas le montrer dans l'arithmétique usuelle de Peano !

Le 10e problème de Hilbert

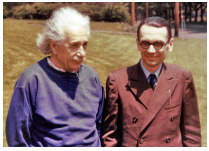
1900 : Hilbert – 1970 : Matiyasevich



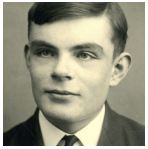
Problème indécidable = **sans algorithme pour le résoudre**

Portrait de famille (cliquez pour des détails)

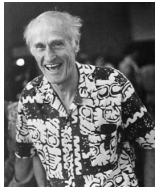

Hilbert


Einstein, Gödel


Herbrand


Turing

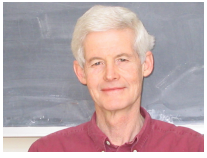

Church


Kleene


Post


Markov

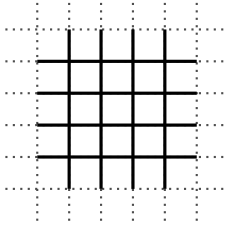

Matiyasevich


Cook


Levin

Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.



Problème du pavage par pièces de Tetris

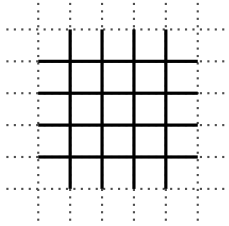
Instance Nombre fini de types de pièces, accolements de carrés 1×1.

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.

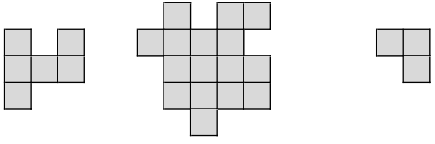


Problème du pavage par pièces de Tetris

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

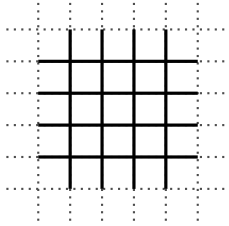
Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(rotations permises, symétries axiales interdites)



Problèmes 11 et 12 : pavages Tetris et Wang

Division du plan en carrés 1×1.

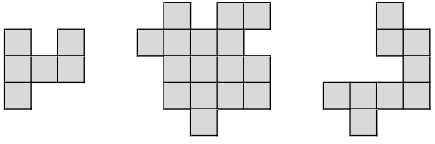


Problème du pavage par pièces de Tetris

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

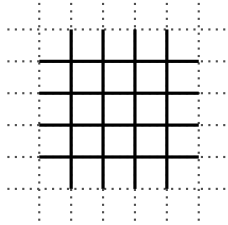
Question Peut-on paver le plan sans trou ni recouvrement en utilisant (une infinité) de pièces uniquement de ces types ?

(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

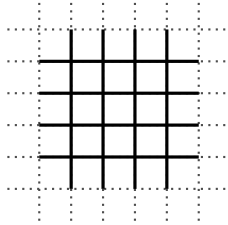
Instance Nombre fini de types de pièces 1×1 avec 4 couleurs.

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

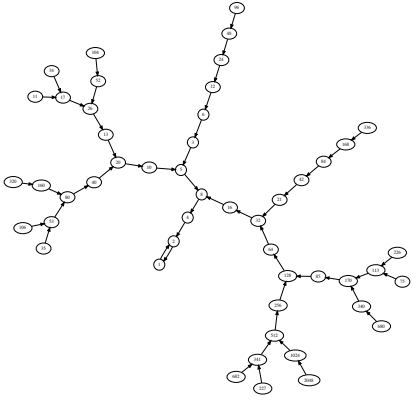
$$f(x) = \begin{cases} x/2 & \text{si } x \text{ est pair,} \\ 3x + 1 & \text{si } x \text{ est impair.} \end{cases}$$

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

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")
```

Si 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")
```

Si 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")
```

Si 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.

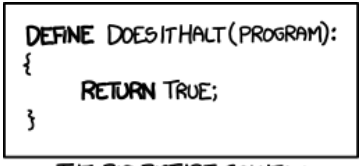
Recap : indécidabilité, problème de l'arrêt

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

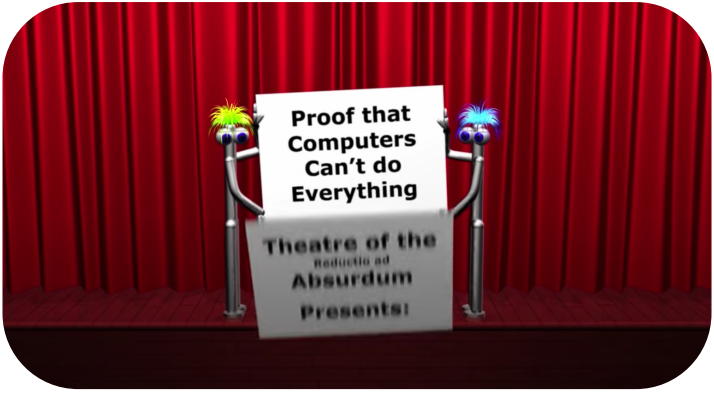
Recap : indécidabilité, problème de l'arrêt

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 fait surprenant 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 fait surprenant 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 fait surprenant 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 42^{e} 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 42^{e} 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**,
on construit une instance (Q, x, v) de **Affectation à 42** telle que :

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

Exemple : problème de l'affectation à 42

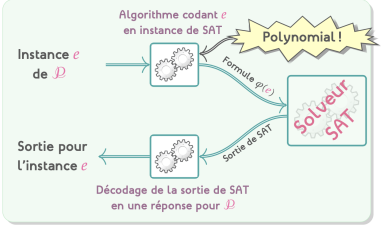
Le problème « Affectation à 42 » est indécidable

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 l'entrée x $\iff$ Q, lancé 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 l'entrée x $\iff$ Q, lancé sur l'entrée x, affecte 42 à v.

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

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 l'entrée x $\iff$ Q, lancé sur l'entrée x, affecte 42 à v.

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

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 :

- 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 :

- 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 :

- 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 ?

2^e 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 : le « support d'exécution »

Abstractions mathématiques de programmes, tournant sur machine idéalisée.

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 □.
- Une tête de lecture-écriture.
- Un nombre fini d'états.
- Un nombre fini de transitions, constituant le programme.

Machines de Turing : le « support d'exécution »

Abstractions mathématiques de programmes, tournant sur machine idéalisée.

Composants d'une machine de Turing

- Bande = mémoire (infinie)
- Tête = pointeur dans la mémoire.
- États = numéros d'instructions.
- Transitions = instructions

Instructions

Un jeu d'instructions très basique !

Les instructions sont numérotées 1, 2, 3, etc. Une instruction peut :

- **Déplacer d'une case** la tête de lecture-écriture à gauche ou à droite.
- **Écrire** 0 ou 1 sous la tête de lecture-écriture.
- **Tester** le bit lu sous la tête :
  - **si** c'est un 0, **aller** en instruction k,
  - **sinon**, **aller** en instruction ℓ.
- **Sarrêter** en émettant un verdict (entrée acceptée ou rejetée).

Instructions

Un jeu d'instructions très basique !

Les instructions sont numérotées 1, 2, 3, etc. Une instruction peut :

- **Déplacer d'une case** la tête de lecture-écriture à gauche ou à droite.
- **Écrire** 0 ou 1 sous la tête de lecture-écriture.
- **Tester** le bit lu sous la tête :
  - **si** c'est un 0, **aller** en instruction k,
  - **sinon**, **aller** en instruction ℓ.
- **Sarrêter** en émettant un verdict (entrée acceptée ou rejetée).

Syntaxe

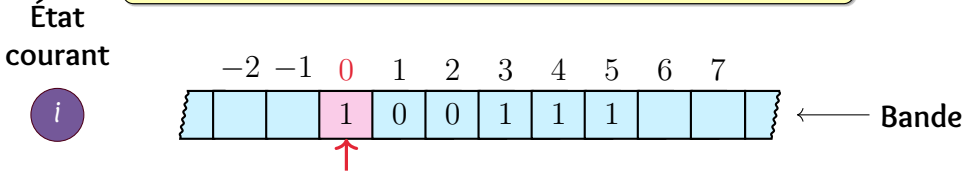
Exemple d'instruction : $\delta(a, 0) = (f, 1, \leftarrow)$: dans l'état a, lire 0 fait

- passer en état f,
- écrire 1 (qui écrase le 0), et
- déplacer la tête à gauche.

Exemple : que fait cette machine ?

- 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.

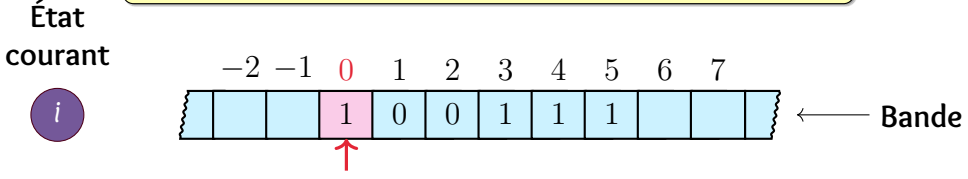
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \qquad \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \qquad \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \qquad \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

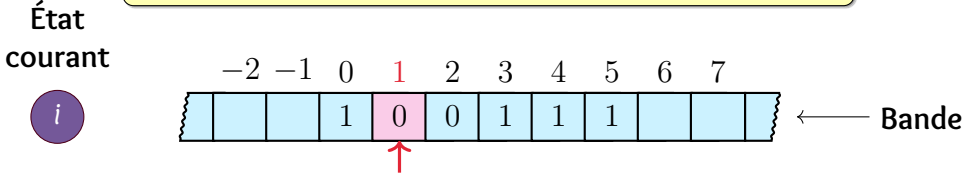
$\delta(i, 0) = (i, 0, \rightarrow)$	$\delta(a, 0) = (f, 1, \leftarrow)$
$\delta(i, 1) = (i, 1, \rightarrow)$	$\delta(a, 1) = (a, 0, \leftarrow)$
$\delta(i, \square) = (a, \square, \leftarrow)$	$\delta(a, \square) = (f, 1, \leftarrow)$



Exemple : que fait cette machine ?

- 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.

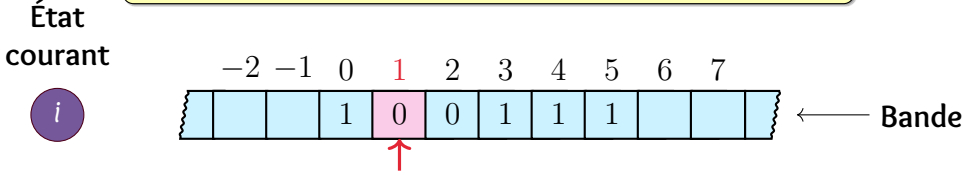
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

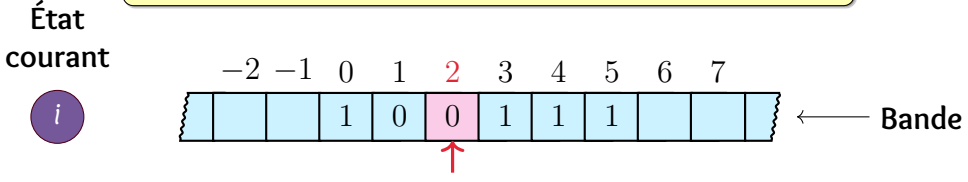
$\delta(i, 0) = (i, 0, \rightarrow)$	$\delta(a, 0) = (f, 1, \leftarrow)$
$\delta(i, 1) = (i, 1, \rightarrow)$	$\delta(a, 1) = (a, 0, \leftarrow)$
$\delta(i, \square) = (a, \square, \leftarrow)$	$\delta(a, \square) = (f, 1, \leftarrow)$



Exemple : que fait cette machine ?

- 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.

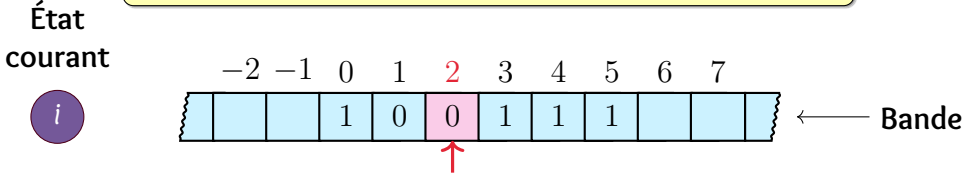
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

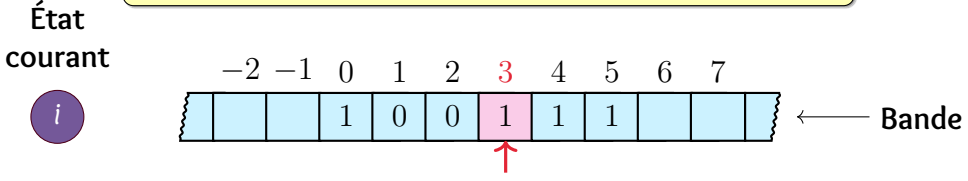
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

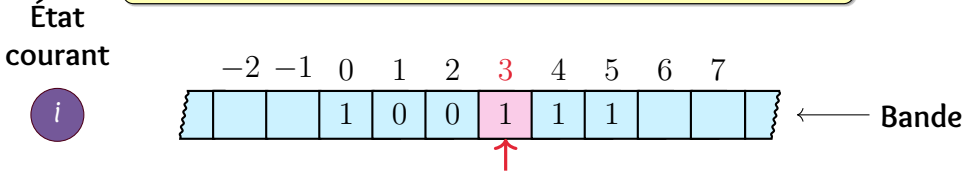
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

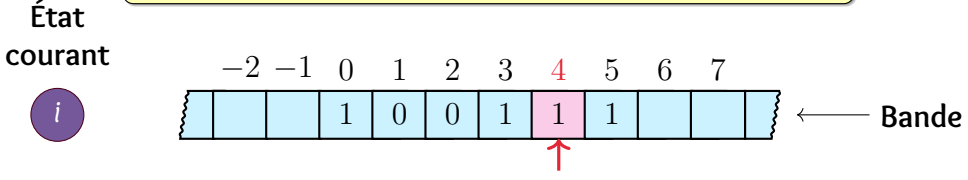
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

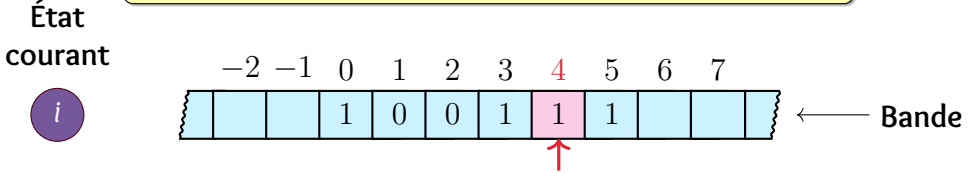
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

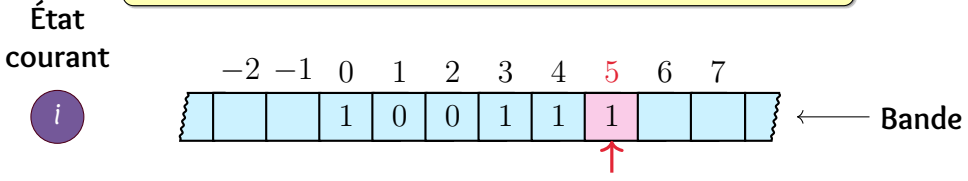
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

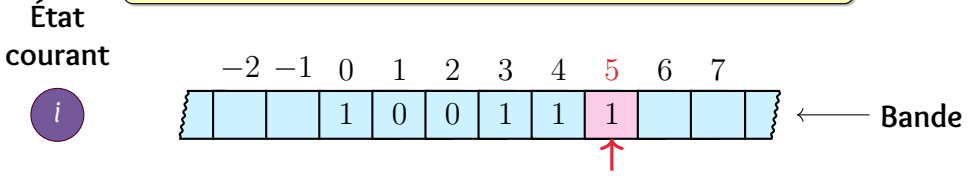
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

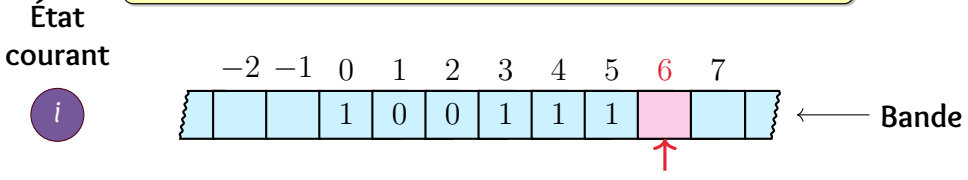
$\delta(i, 0) = (i, 0, \rightarrow)$	$\delta(a, 0) = (f, 1, \leftarrow)$
$\delta(i, 1) = (i, 1, \rightarrow)$	$\delta(a, 1) = (a, 0, \leftarrow)$
$\delta(i, \square) = (a, \square, \leftarrow)$	$\delta(a, \square) = (f, 1, \leftarrow)$



Exemple : que fait cette machine ?

- 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.

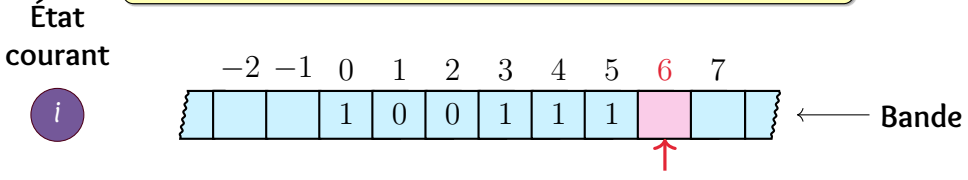
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

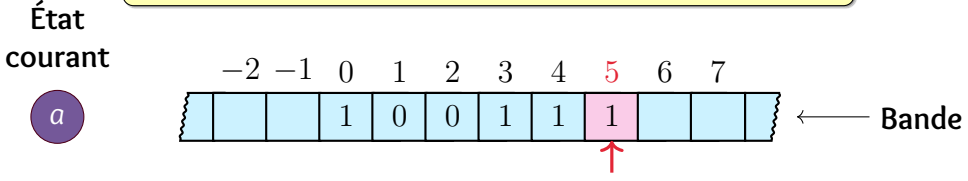
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

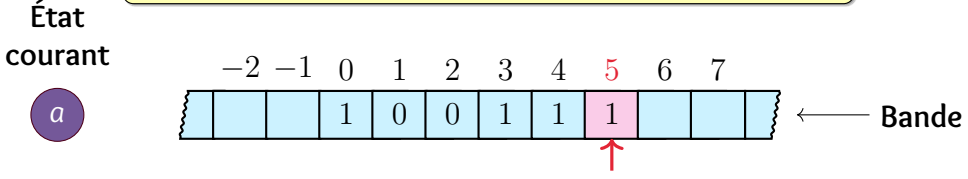
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

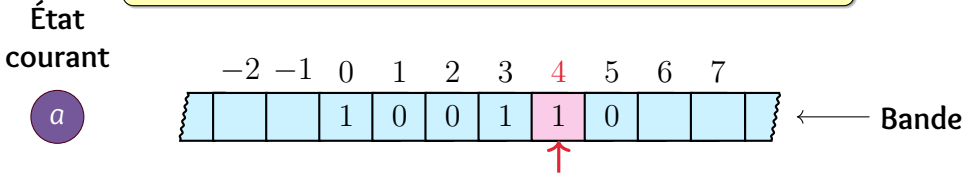
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

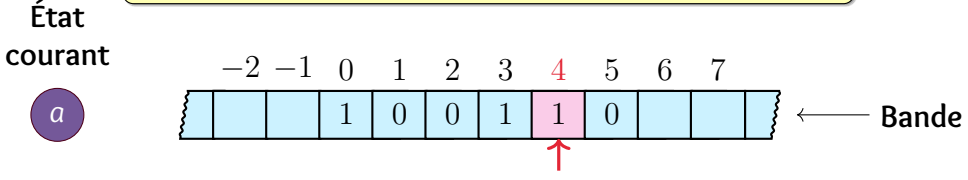
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

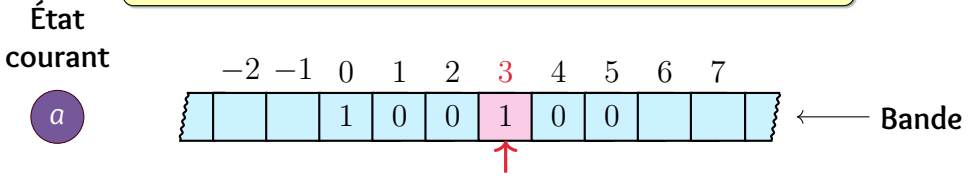
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

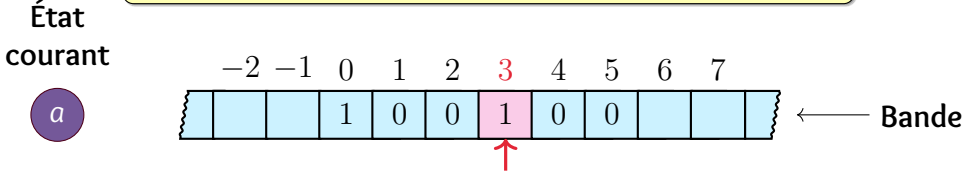
$$
\begin{array}{llll}
\delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\
\delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\
\delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow)
\end{array}
$$



Exemple : que fait cette machine ?

- 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.

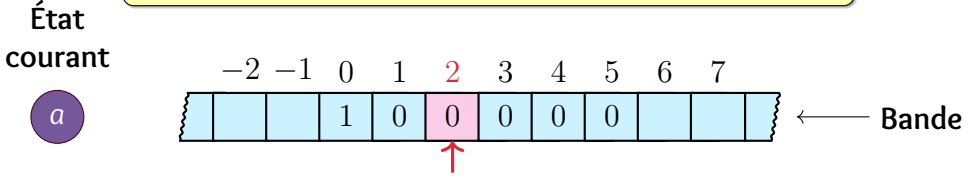
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

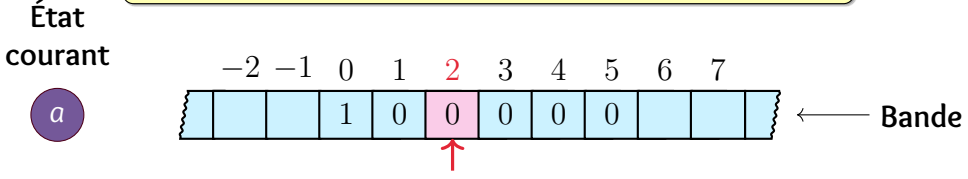
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

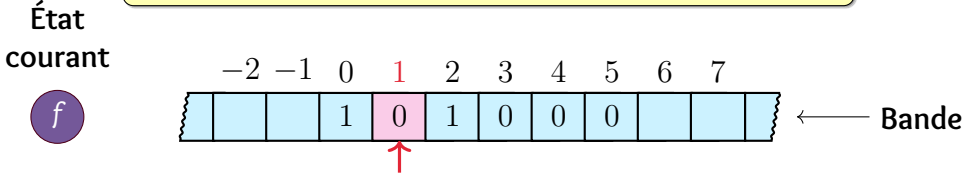
$$\begin{array}{llll} \delta(i,0) & = & (i,0,\rightarrow) & \delta(a,0) = (f,1,\leftarrow) \\ \delta(i,1) & = & (i,1,\rightarrow) & \delta(a,1) = (a,0,\leftarrow) \\ \delta(i,\square) & = & (a,\square,\leftarrow) & \delta(a,\square) = (f,1,\leftarrow) \end{array}$$



Exemple : que fait cette machine ?

- 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.

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



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. C'est un modèle mathématique, qui ne représente pas les programmes réalistes.

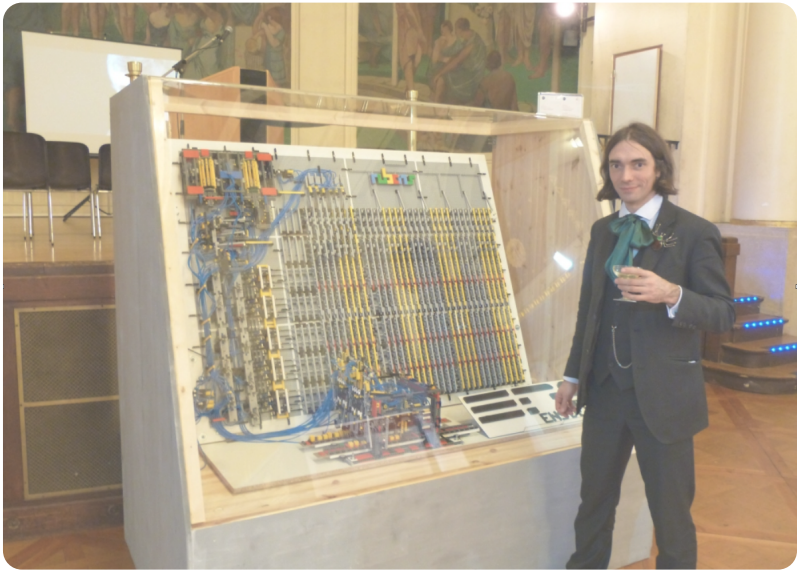
Exécution d'une machine de Turing

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. C'est un modèle mathématique, qui ne représente pas les programmes réalistes.

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.
- Amusant : tout langage « Turing-complet » admet un *quine* (lien Wikipedia).

Pour d'autres aventures : la complexité

Quelques unes des classes décidables les plus courantes.