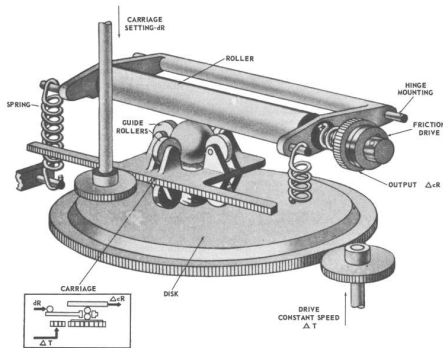


# Cours 7: Compléments sur la calculabilité. Preuve du théorème d'incomplétude. Introduction à la complexité.



Olivier Bournez  
bournez@lix.polytechnique.fr

Ecole Polytechnique  
INF412

# Au menu

Quelques résultats amusants de la calculabilité

Théorèmes d'incomplétude de Gödel

Décidabilité de logiques

Esprit du second théorème

Introduction à la complexité

# Plus précisément

## Quelques résultats amusants de la calculabilité

Quines

Théorème de récursion

Applications

Problème de l'arrêt généralisé

# Quines

## Proposition

*Il existe une machine de Turing  $A^*$  qui écrit son propre programme : elle produit en sortie  $\langle A^* \rangle$ .*

En shell SH par exemple, le programme suivant

```
z=\' a=\'z=\\$z a=$z$a$z\; eval echo \$a\; eval echo $a
```

produit

```
z=\' a=\'z=\\$z a=$z$a$z\; eval echo \$a\; eval echo $a
```

■ Démonstration :

- ▶ On considère des machines qui terminent sur toute entrée.
  - Pour deux telles machines  $A$  et  $A'$ , on note  $AA'$  la machine qui est obtenue en composant de façon séquentielle  $A$  et  $A'$ .

$$AA' : \boxed{A} \cdot \boxed{A'}$$

■ Démonstration :

- ▶ On considère des machines qui terminent sur toute entrée.
  - Pour deux telles machines  $A$  et  $A'$ , on note  $AA'$  la machine qui est obtenue en composant de façon séquentielle  $A$  et  $A'$ .

$$AA' : \boxed{A} \cdot \boxed{A'}$$

- ▶ On construit les machines suivantes :

■ Démonstration :

- ▶ On considère des machines qui terminent sur toute entrée.
  - Pour deux telles machines  $A$  et  $A'$ , on note  $AA'$  la machine qui est obtenue en composant de façon séquentielle  $A$  et  $A'$ .

$$AA' : \boxed{A} \cdot \boxed{A'}$$

- ▶ On construit les machines suivantes :
  1. La machine  $Print_w$  termine avec le résultat  $w$ .



■ Démonstration :

- ▶ On considère des machines qui terminent sur toute entrée.
  - Pour deux telles machines  $A$  et  $A'$ , on note  $AA'$  la machine qui est obtenue en composant de façon séquentielle  $A$  et  $A'$ .

$$AA' : \boxed{A} \boxed{A'}$$

- ▶ On construit les machines suivantes :
  1. La machine  $Print_w$  termine avec le résultat  $w$ .
  2. Pour une entrée  $w$  de la forme  $w = \langle X \rangle$ , où  $X$  est une machine de Turing, la machine  $B$  produit en sortie le codage de la machine  $Print_w X$ .

$$B : \langle X \rangle \mapsto \langle \boxed{Print_{\langle X \rangle}} \boxed{X} \rangle$$

■ Démonstration :

- ▶ On considère des machines qui terminent sur toute entrée.
  - Pour deux telles machines  $A$  et  $A'$ , on note  $AA'$  la machine qui est obtenue en composant de façon séquentielle  $A$  et  $A'$ .

$$AA' : \boxed{A} \boxed{A'}$$

- ▶ On construit les machines suivantes :
  1. La machine  $Print_w$  termine avec le résultat  $w$ .
  2. Pour une entrée  $w$  de la forme  $w = \langle X \rangle$ , où  $X$  est une machine de Turing, la machine  $B$  produit en sortie le codage de la machine  $Print_w X$ .

$$B : \langle X \rangle \mapsto \langle \boxed{Print_{\langle X \rangle}} \boxed{X} \rangle$$

- ▶ On considère alors la machine  $A^*$  donnée par  $Print_{\langle B \rangle} B$ .

$$A^* : \boxed{Print_{\langle B \rangle}} \boxed{B}$$

■ Démonstration :

- ▶ On considère des machines qui terminent sur toute entrée.
  - Pour deux telles machines  $A$  et  $A'$ , on note  $AA'$  la machine qui est obtenue en composant de façon séquentielle  $A$  et  $A'$ .

$$AA' : \boxed{A} \boxed{A'}$$

- ▶ On construit les machines suivantes :
  1. La machine  $Print_w$  termine avec le résultat  $w$ .
  2. Pour une entrée  $w$  de la forme  $w = \langle X \rangle$ , où  $X$  est une machine de Turing, la machine  $B$  produit en sortie le codage de la machine  $Print_w X$ .

$$B : \langle X \rangle \mapsto \langle \boxed{Print_{\langle X \rangle}} \boxed{X} \rangle$$

- ▶ On considère alors la machine  $A^*$  donnée par  $Print_{\langle B \rangle} B$ .

$$A^* : \boxed{Print_{\langle B \rangle}} \boxed{B}$$

- ▶ Déroulons le résultat de cette machine : la machine  $Print_{\langle B \rangle}$  produit en sortie  $\langle B \rangle$ . La composition par  $B$  produit alors le codage de  $Print_{\langle B \rangle} B$ , qui est bien le codage de la machine  $A^*$ .

# Plus précisément

## Quelques résultats amusants de la calculabilité

Quines

Théorème de récursion

Applications

Problème de l'arrêt généralisé

# Théorème de récursion

## Théorème (Théorème de récursion)

*Soit  $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  une fonction calculable. Alors il existe une machine de Turing  $R$  qui calcule une fonction  $r : \Sigma^* \rightarrow \Sigma^*$  telle que pour tout mot  $w$*

$$r(w) = t(\langle R \rangle, w).$$

► Démonstration

### ■ Comment l'utiliser ?

- ▶ Pour obtenir une machine de Turing qui utilise sa propre description,
  - il suffit d'avoir une machine, que l'on appelle  $T$ , qui calcule la fonction  $t$  dans l'énoncé, qui prend comme entrée supplémentaire la description de la machine.
- ▶ Le théorème produit une machine  $R$  qui fait exactement comme  $T$ , mais avec son propre codage  $\langle R \rangle$  rempli automatiquement.

# Plus précisément

## Quelques résultats amusants de la calculabilité

Quines

Théorème de récursion

**Applications**

Problème de l'arrêt généralisé

## Exemple : Quines revisitées

- Pour obtenir SELF =
  - ▶ sur toute entrée  $w$ ,
    - obtenir par le théorème de récursion sa propre description  $\langle SELF \rangle$ .
    - afficher  $\langle SELF \rangle$ .

## Exemple : Quines revisitées

- Pour obtenir  $SELF =$ 
  - ▶ sur toute entrée  $w$ ,
    - obtenir par le théorème de récursion sa propre description  $\langle SELF \rangle$ .
    - afficher  $\langle SELF \rangle$ .
- On prend  $T =$ 
  - ▶ sur toute entrée  $(\langle X \rangle, w)$ 
    - afficher  $\langle X \rangle$ .



## Exemple : Quines revisitées

- Pour obtenir SELF =
  - ▶ sur toute entrée  $w$ ,
    - obtenir par le théorème de récursion sa propre description  $\langle SELF \rangle$ .
    - afficher  $\langle SELF \rangle$ .
  
- On prend T=
  - ▶ sur toute entrée  $(\langle X \rangle, w)$ 
    - afficher  $\langle X \rangle$ .
  
- En résumé :
  - ▶ “obtenir par le théorème de récursion sa propre description” devient une instruction valide dans un algorithme.

- On peut interpréter les preuves des résultats précédents en lien avec les virus informatiques :
  - ▶ En effet, un virus est un programme qui vise à se diffuser, c'est-à-dire à s'autoreproduire, sans être détecté.
  - ▶ Le principe de la preuve du théorème de récursion est un moyen de s'autoreproduire, en dupliquant son code.

## Exercice

### Théorème (Du point fixe de Kleene)

*Soit une fonction calculable qui à chaque mot  $\langle A \rangle$  codant une machine de Turing associe un mot  $\langle A' \rangle$  codant une machine de Turing. Notons  $A' = f(A)$ .*

*Alors il existe une machine de Turing  $A^*$  tel que  $L(A^*) = L(f(A^*))$ .*

► Démonstration

# Plus précisément

## Quelques résultats amusants de la calculabilité

Quines

Théorème de récursion

Applications

Problème de l'arrêt généralisé

# Le problème de l'arrêt généralisé

- Problème  $\forall$ HALTING-PROBLEM:

**Donnée:** Le codage  $\langle M \rangle$  d'une machine de Turing  $M$

**Réponse:** Décider si la machine  $M$  accepte tout mot  $w$ .

## Théorème

*Le problème  $\forall$ HALTING-PROBLEM n'est pas décidable.*

# Le problème de l'arrêt généralisé

- Problème  $\forall$ HALTING-PROBLEM:

**Donnée:** Le codage  $\langle M \rangle$  d'une machine de Turing  $M$

**Réponse:** Décider si la machine  $M$  accepte tout mot  $w$ .

## Théorème

*Le problème  $\forall$ HALTING-PROBLEM n'est pas décidable.*

- ▶ Application du Théorème de Rice.

# Le problème de l'arrêt généralisé

- Problème  $\forall$ HALTING-PROBLEM:

**Donnée:** Le codage  $\langle M \rangle$  d'une machine de Turing  $M$

**Réponse:** Décider si la machine  $M$  accepte tout mot  $w$ .

## Théorème

*Le problème  $\forall$ HALTING-PROBLEM n'est pas décidable.*

▶ Application du Théorème de Rice.

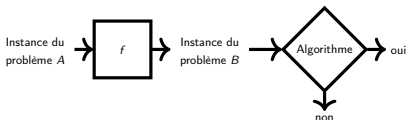
- En fait, il n'est pas récursivement énumérable, et ni son complémentaire !

■ Problème  $\overline{L_{\text{univ}}}$ :

**Donnée:** Le codage  $\langle M \rangle$  d'une machine de Turing  $M$  et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  n'accepte pas le mot  $w$ .  
n'est pas récursivement énumérable.

- Pour montrer que  $\forall$ HALTING-PROBLEM n'est pas récursivement énumérable, il suffit de montrer que  $\overline{L_{\text{univ}}} \leq_m \forall$ HALTING-PROBLEM.





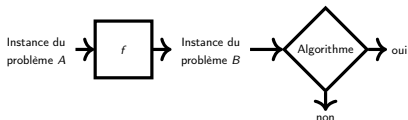
■ Problème  $\overline{L_{\text{univ}}}$ :

**Donnée:** Le codage  $\langle M \rangle$  d'une machine de Turing  $M$  et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  n'accepte pas le mot  $w$ .  
n'est pas récursivement énumérable.

- Pour montrer que  $\forall$ HALTING-PROBLEM n'est pas récursivement énumérable, il suffit de montrer que  $\overline{L_{\text{univ}}} \leq_m \forall$ HALTING-PROBLEM.

- ▶ En effet,  $\overline{L_{\text{univ}}}$  n'est pas récursivement énumérable ;



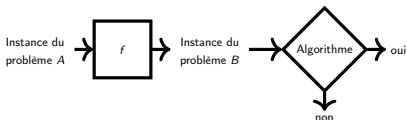
■ Problème  $\overline{L_{\text{univ}}}$ :

**Donnée:** Le codage  $\langle M \rangle$  d'une machine de Turing  $M$  et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  n'accepte pas le mot  $w$ .  
n'est pas récursivement énumérable.

■ Pour montrer que  $\forall$ HALTING-PROBLEM n'est pas récursivement énumérable, il suffit de montrer que  $\overline{L_{\text{univ}}} \leq_m \forall$ HALTING-PROBLEM.

- ▶ En effet,  $\overline{L_{\text{univ}}}$  n'est pas récursivement énumérable ;
- ▶ Et si  $A \leq_m B$  et si  $A$  n'est pas récursivement énumérable, alors  $B$  non plus.



■  $\overline{L_{\text{univ}}} \leq_m \forall \text{HALTING-PROBLEM}$  :

- ▶ A partir du codage  $\langle M \rangle$  d'une machine de Turing, et de  $w$ , on peut considérer la machine  $A_M$  qui sur l'entrée  $u$ 
  - exécute  $M$  sur  $w$  pendant  $|u|$  étapes.
  - si cette simulation atteint un état final acceptant, boucle ensuite indéfiniment.
  - sinon, soit la simulation a atteint un état final refusant, soit elle n'a pas terminé en  $|u|$  étapes, et alors  $A_M$  termine en acceptant.

■  $(\langle M \rangle, w) \in \overline{L_{\text{univ}}}$  si et seulement si  
 $\langle A_M \rangle \in \forall \text{HALTING-PROBLEM}$

■ La fonction qui à  $(\langle M \rangle, w)$  associe  $\langle A_M \rangle$  est bien calculable.

- Pour montrer que son complémentaire  $\overline{\forall\text{HALTING-PROBLEM}}$  n'est pas récursivement énumérable, il suffit de montrer de façon similaire que  $\overline{L_{\text{univ}}} \leq_m \overline{\forall\text{HALTING-PROBLEM}}$ .
  - ▶ A partir du codage  $\langle M \rangle$  d'une machine de Turing, et de  $w$ , on peut considérer la machine  $A_M$  qui ignore son entrée et exécute  $M$  sur le mot  $w$ .
- $(\langle M \rangle, w) \in \overline{L_{\text{univ}}}$  si et seulement si  $\langle A_M \rangle \in \overline{\forall\text{HALTING-PROBLEM}}$
- La fonction qui à  $(\langle M \rangle, w)$  associe  $\langle A_M \rangle$  est bien calculable.

# Au menu

Quelques résultats amusants de la calculabilité

**Théorèmes d'incomplétude de Gödel**

Décidabilité de logiques

Esprit du second théorème

Introduction à la complexité

# Plus précisément

## Théorèmes d'incomplétude de Gödel

### Rappels

Diagrammes espace-temps : définition

Diagrammes espace-temps : caractérisation & Localité du calcul

Prouver le premier théorème

- On note  $Th(\mathbb{N})$  l'ensemble des formules closes  $F$  qui sont vraies sur les entiers.

### Théorème (Incomplétude)

*Il existe des formules closes de  $Th(\mathbb{N})$  qui ne sont pas prouvables, à partir des axiomes de Peano, ou de toute axiomatisation "raisonnable"<sup>1</sup> des entiers.*

### Théorème (Second théorème d'incomplétude)

*La cohérence de l'arithmétique (ou sa négation) est un exemple de telle formule.*

---

1. Formellement, "récurivement énumérable" : voir suite du cours.

# Plus précisément

## Théorèmes d'incomplétude de Gödel

Rappels

**Diagrammes espace-temps : définition**

Diagrammes espace-temps : caractérisation & Localité du calcul

Prouver le premier théorème

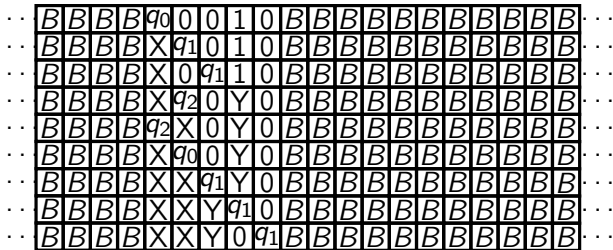


## Diagramme Espace-temps

- On représente souvent une suite de configurations d'une machine de Turing ligne par ligne : la ligne numéro  $i$  représente la  $i$ ème configuration du calcul (en utilisant la notation 2 pour les configurations).
- Exemple : Diagramme espace-temps du calcul sur 0011.

|    |   |   |   |   |       |       |       |       |       |       |   |   |   |   |   |   |   |   |   |    |    |
|----|---|---|---|---|-------|-------|-------|-------|-------|-------|---|---|---|---|---|---|---|---|---|----|----|
| .. | B | B | B | B | $q_0$ | 0     | 0     | 1     | 1     | B     | B | B | B | B | B | B | B | B | B | .. |    |
| .. | B | B | B | B | X     | $q_1$ | 0     | 1     | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | 0     | $q_1$ | 1     | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | $q_2$ | 0     | Y     | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | $q_2$ | X     | 0     | Y     | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | $q_0$ | 0     | Y     | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | $q_1$ | Y     | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | Y     | $q_1$ | 1     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | $q_2$ | Y     | Y     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | $q_2$ | X     | Y     | Y     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | $q_0$ | Y     | Y     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | Y     | $q_3$ | Y     | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | Y     | Y     | $q_3$ | B     | B | B | B | B | B | B | B | B | B | B  | .. |
| .. | B | B | B | B | X     | X     | Y     | Y     | B     | $q_4$ | B | B | B | B | B | B | B | B | B | B  | .. |

- Exemple : Diagramme espace-temps du calcul sur 0010 :



- ▶ sur la dernière configuration plus aucune évolution n'est possible, et donc il n'y a pas de calcul acceptant partant de 0010.

# Plus précisément

## Théorèmes d'incomplétude de Gödel

Rappels

Diagrammes espace-temps : définition

Diagrammes espace-temps : caractérisation & Localité du calcul

Prouver le premier théorème

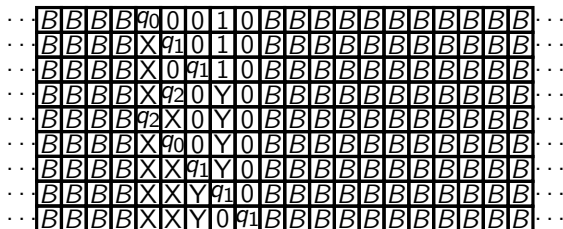
# Diagramme espace-temps

- Considérons

- ▶ un diagramme espace-temps d'une machine de Turing  $M$ .

- Exemple :

- ▶ Diagramme espace-temps du calcul sur 0010 de la machine reconnaissant  $0^n 1^n$ ,  $n \in \mathbb{N}$  :



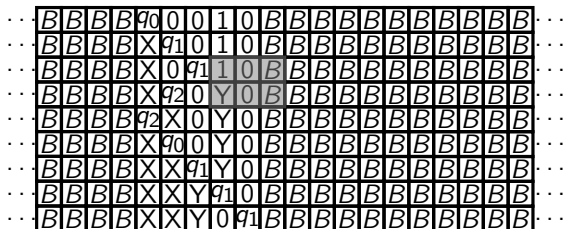
# Diagramme espace-temps

- Considérons

- ▶ un diagramme espace-temps d'une machine de Turing  $M$ .
- ▶ et un sous-rectangle  $3 \times 2$  dans ce diagramme, que l'on appellera **fenêtre**.

- Exemple :

- ▶ Diagramme espace-temps du calcul sur 0010 de la machine reconnaissant  $0^n 1^n, n \in \mathbb{N}$  :



- ▶ Fenêtre correspondante :



# Fixons la machine $M$

- Propriétés :

|     |   |   |   |   |       |       |       |       |       |   |   |   |   |   |   |   |   |   |   |   |     |
|-----|---|---|---|---|-------|-------|-------|-------|-------|---|---|---|---|---|---|---|---|---|---|---|-----|
| ... | B | B | B | B | $q_0$ | 0     | 0     | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_1$ | 0     | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | 0     | $q_1$ | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_2$ | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | $q_2$ | X     | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_0$ | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | $q_1$ | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | Y     | $q_1$ | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | $q_2$ | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_2$ | X     | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | $q_0$ | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | Y     | $q_3$ | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | Y     | Y     | $q_3$ | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | Y     | Y     | Y     | $q_1$ | B | B | B | B | B | B | B | B | B | B | B | ... |

## Fixons la machine $M$

- Propriétés :

- il y a un nombre fini de contenus possibles, que l'on peut appeler **légaux**.

|     |   |   |   |   |       |       |       |       |       |   |   |   |   |   |   |   |   |   |   |   |     |
|-----|---|---|---|---|-------|-------|-------|-------|-------|---|---|---|---|---|---|---|---|---|---|---|-----|
| ... | B | B | B | B | $q_0$ | 0     | 0     | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_1$ | 0     | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | 0     | $q_1$ | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_2$ | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | $q_2$ | X     | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_0$ | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | $q_1$ | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | Y     | $q_1$ | 1     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | $q_2$ | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | $q_2$ | X     | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | $q_0$ | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | Y     | $q_3$ | Y     | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | X     | Y     | Y     | $q_3$ | B | B | B | B | B | B | B | B | B | B | B | ... |
| ... | B | B | B | B | X     | Y     | Y     | Y     | $q_1$ | B | B | B | B | B | B | B | B | B | B | B | ... |

## Fixons la machine $M$

### ■ Propriétés :

- ▶ il y a un nombre fini de contenus possibles, que l'on peut appeler **légaux**.
- ▶ cela fournit même une caractérisation des diagrammes espace-temps de  $M$  :
  - un tableau est un diagramme espace-temps de  $M$  sur une certaine configuration initiale  $C_0$  ssi d'une part sa première ligne correspond à  $C_0$ , et d'autre part le contenu de tous les sous-rectangles  $3 \times 2$  est parmi les fenêtres légaux.

|    |   |   |   |   |       |       |       |       |       |   |   |   |   |   |   |   |   |   |   |   |    |
|----|---|---|---|---|-------|-------|-------|-------|-------|---|---|---|---|---|---|---|---|---|---|---|----|
| .. | B | B | B | B | $q_0$ | 0     | 0     | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | $q_1$ | 0     | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | 0     | $q_1$ | 1     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | $q_2$ | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | $q_2$ | X     | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | $q_0$ | 0     | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | X     | $q_1$ | Y     | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | X     | Y     | $q_1$ | 1     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | X     | $q_2$ | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | $q_2$ | X     | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | X     | $q_0$ | Y     | Y     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | X     | Y     | $q_3$ | Y     | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | X     | Y     | Y     | $q_3$ | B | B | B | B | B | B | B | B | B | B | B | .. |
| .. | B | B | B | B | X     | Y     | Y     | Y     | $q_1$ | B | B | B | B | B | B | B | B | B | B | B | .. |



- Quelques fenêtres **légales** pour une machine  $M$  :

(a) 

|       |       |   |
|-------|-------|---|
| a     | $q_1$ | b |
| $q_2$ | a     | c |

(b) 

|   |       |       |
|---|-------|-------|
| a | $q_1$ | b     |
| a | a     | $q_2$ |

(c) 

|   |   |       |
|---|---|-------|
| a | a | $q_1$ |
| a | a | b     |

(d) 

|   |   |   |
|---|---|---|
| B | b | a |
| B | b | a |

(e) 

|   |   |       |
|---|---|-------|
| a | b | a     |
| a | b | $q_2$ |

(f) 

|   |   |   |
|---|---|---|
| b | b | b |
| c | b | b |

- Quelques fenêtres non-légales pour une machine  $M$  avec  $\delta(q_1, b) = (q_1, c, \leftarrow)$  :

(a) 

|   |   |   |
|---|---|---|
| a | b | a |
| a | a | a |

(b) 

|       |       |   |
|-------|-------|---|
| a     | $q_1$ | b |
| $q_1$ | a     | a |

(c) 

|       |       |       |
|-------|-------|-------|
| b     | $q_1$ | b     |
| $q_2$ | b     | $q_2$ |

# Plus précisément

## Théorèmes d'incomplétude de Gödel

Rappels

Diagrammes espace-temps : définition

Diagrammes espace-temps : caractérisation & Localité du calcul

Prouver le premier théorème

# Prouver le premier théorème

- Il suffit de prouver en fait que :
  1. L'ensemble des formules closes prouvables à partir des axiomes de Peano (ou de toute axiomatisation récursivement énumérable des entiers) est récursivement énumérable.

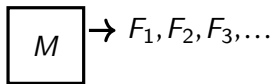
# Prouver le premier théorème

- Il suffit de prouver en fait que :
  1. L'ensemble des formules closes prouvables à partir des axiomes de Peano (ou de toute axiomatisation récursivement énumérable des entiers) est récursivement énumérable.
  2. L'ensemble  $Th(\mathbb{N})$  des formules closes  $F$  vraies sur les entiers n'est pas récursivement énumérable.

# Prouver le premier théorème

- Il suffit de prouver en fait que :
  1. L'ensemble des formules closes prouvables à partir des axiomes de Peano (ou de toute axiomatisation récursivement énumérable des entiers) est récursivement énumérable.
  2. L'ensemble  $Th(\mathbb{N})$  des formules closes  $F$  vraies sur les entiers n'est pas récursivement énumérable.
  
- Par conséquent, les deux ensembles ne peuvent pas être les mêmes :
  - ▶ il y a donc des formules closes de  $Th(\mathbb{N})$  qui ne sont pas prouvables.

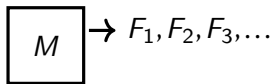
# Etape 1 : l'ensemble des formules closes prouvables est récursivement énumérable



## ■ Facile à établir :

- ▶ on peut énumérer les formules closes prouvables :
  - il suffit d'énumérer les axiomes et d'appliquer systématiquement toutes les règles de déduction dans toutes les façons possibles : on produit en sortie chaque formule close que l'on arrive à dériver.

## Etape 1 : l'ensemble des formules closes prouvables est récursivement énumérable



### ■ Facile à établir :

- ▶ on peut énumérer les formules closes prouvables :
  - il suffit d'énumérer les axiomes et d'appliquer systématiquement toutes les règles de déduction dans toutes les façons possibles : on produit en sortie chaque formule close que l'on arrive à dériver.
- ▶ Cela reste vrai en fait dès que l'on suppose que l'on peut énumérer les formules de l'axiomatisation dont on part.

Etape 2 :  $Th(\mathbb{N})$  n'est pas récursivement énumérable.

- Il suffit de montrer que  $\overline{L_{univ}} \leq_m Th(\mathbb{N})$ .



Etape 2 :  $Th(\mathbb{N})$  n'est pas récursivement énumérable.

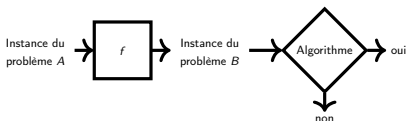
■ Il suffit de montrer que  $\overline{L_{univ}} \leq_m Th(\mathbb{N})$ .

▶ En effet,  $\overline{L_{univ}}$  n'est pas récursivement énumérable ;

## Etape 2 : $Th(\mathbb{N})$ n'est pas récursivement énumérable.

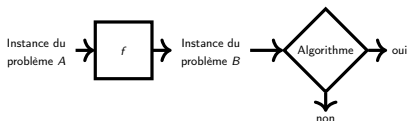
- Il suffit de montrer que  $\overline{L_{univ}} \leq_m Th(\mathbb{N})$ .

- ▶ En effet,  $\overline{L_{univ}}$  n'est pas récursivement énumérable ;
- ▶ Et si  $A \leq_m B$  et si  $A$  n'est pas récursivement énumérable, alors  $B$  non plus.



## Etape 2 : $Th(\mathbb{N})$ n'est pas récursivement énumérable.

- Il suffit de montrer que  $\overline{L_{univ}} \leq_m Th(\mathbb{N})$ .
  - ▶ En effet,  $\overline{L_{univ}}$  n'est pas récursivement énumérable ;
  - ▶ Et si  $A \leq_m B$  et si  $A$  n'est pas récursivement énumérable, alors  $B$  non plus.

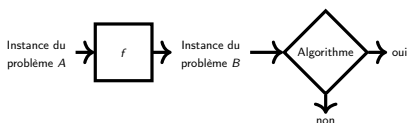


- Autrement dit,
  - ▶ Étant donné  $(\langle M \rangle, w)$ , produire effectivement une formule close  $\gamma_{M,w}$  telle que

$$(\langle M \rangle, w) \in \overline{L_{univ}} \Leftrightarrow \gamma_{M,w} \in Th(\mathbb{N}).$$

## Etape 2 : $Th(\mathbb{N})$ n'est pas récursivement énumérable.

- Il suffit de montrer que  $\overline{L_{univ}} \leq_m Th(\mathbb{N})$ .
  - ▶ En effet,  $\overline{L_{univ}}$  n'est pas récursivement énumérable ;
  - ▶ Et si  $A \leq_m B$  et si  $A$  n'est pas récursivement énumérable, alors  $B$  non plus.



- Autrement dit,
  - ▶ Étant donné  $(\langle M \rangle, w)$ , produire effectivement une formule close  $\gamma_{M,w}$  telle que

$$(\langle M \rangle, w) \in \overline{L_{univ}} \Leftrightarrow \gamma_{M,w} \in Th(\mathbb{N}).$$

- ▶ il suffit que cette formule close  $\gamma_{M,w}$  exprime le fait que  $M$  n'accepte pas le mot  $w$ .

## Quelques formules de l'arithmétique

### ■ Exprimer quelques faits élémentaires. . .

- ▶  $\text{INTDIV}(x, y, q, r)$  : “ $q$  est le quotient et  $r$  le reste de la division euclidienne de  $x$  par  $y$ ” :

$$(x = q * y + r \wedge r < y).$$

- ▶  $\text{DIV}(y, x)$  : “ $y$  divise  $x$ ” :

$$\exists q \text{ INTDIV}(x, y, q, 0).$$

- ▶  $\text{EVEN}(x)$  : “ $x$  est pair” :  $\text{DIV}(2, x)$ .
- ▶  $\text{ODD}(x)$  : “ $x$  est impair” :  $\neg \text{EVEN}(x)$ .
- ▶  $\text{PRIME}(x)$  : “ $x$  est premier” :

$$(x \geq 2 \wedge \forall y (\text{DIV}(y, x) \Rightarrow (y = 1 \vee y = x))).$$

- ▶  $\text{POWER}_p(x)$  : “Le nombre  $x$  est une puissance de  $p$ ” ( $p$  nombre premier fixé) :

$$\forall y ((\text{DIV}(y, x) \wedge \text{PRIME}(y)) \Rightarrow y = p).$$

## Quelques formules de l'arithmétique

- Un mot  $w = a_1 a_2 \cdots a_n$  sur l'alphabet  $\Sigma = \{0, 1, \dots, p-1\}$  peut toujours être vu comme l'entier

$$v = a_1 + a_2 p + \cdots + a_{n-1} p^{n-2} + a_n p^{n-1}.$$

- Exprimer quelques faits sur un mot :

- ▶  $\text{LENGTH}_p(v, d)$  : "Le nombre  $d$  est une puissance de  $p$  qui donne la longueur de  $v$  vu comme un mot sur l'alphabet  $\Sigma$  à  $p$  lettres" :

$$(\text{POWER}_p(d) \wedge v < d \wedge p * v \geq d).$$

- ▶  $\text{DIGIT}_p(v, K, b)$  : "Le ' $k$ ème' chiffre de  $v$  écrit en base  $p$  est  $b$  (où  $K = p^k$ )" :

$$\exists u \exists a (v = a + b * K + u * p * K \wedge a < K \wedge b < p).$$

- ▶  $\text{3DIGIT}_p(v, K, b, c, d)$  : "Les 3 chiffres consécutifs de  $v$  à la position  $k$  sont  $b$ ,  $c$  et  $d$  (où  $K = p^k$ )" :

$$\begin{aligned} \exists u \exists a (v = a + b * K + c * p * K + d * p * p * K + u * p * p * p * K \\ \wedge a < K \wedge b < p \wedge c < p \wedge d < p). \end{aligned}$$

## Quelques formules de l'arithmétique

- Exprimer le diagramme espace temps d'une machine :
  - ▶  $\text{MATCH}_p(v, L, M)$  : "Les 3 chiffres de  $v$  à la position  $\ell$  sont  $a$ ,  $b$  et  $c$  et correspondent aux 3 chiffres de  $v$  à la position  $m$  selon la fonction de transition  $\delta$  de la machine de Turing (où  $L = p^\ell$  et  $M = p^m$ )" :

$$\bigwedge_{(a,b,c,d,e,f) \in \text{LEGAL}} 3\text{DIGIT}_p(v, L, a, b, c) \wedge 3\text{DIGIT}_p(v, M, d, e, f).$$

## Quelques formules de l'arithmétique

- Exprimer le diagramme espace temps d'une machine :
  - ▶  $\text{MATCH}_p(v, L, M)$  : "Les 3 chiffres de  $v$  à la position  $\ell$  sont  $a$ ,  $b$  et  $c$  et correspondent aux 3 chiffres de  $v$  à la position  $m$  selon la fonction de transition  $\delta$  de la machine de Turing (où  $L = p^\ell$  et  $M = p^m$ )" :

$$\bigwedge_{(a,b,c,d,e,f) \in \text{LEGAL}} 3\text{DIGIT}_p(v, L, a, b, c) \wedge 3\text{DIGIT}_p(v, M, d, e, f).$$

- $(a, b, c, d, e, f) \in \text{LEGAL}$  désigne le fait que

|   |   |   |
|---|---|---|
| a | b | c |
| d | e | f |

est une fenêtre légale.



## Quelques formules de l'arithmétique

- Exprimer le diagramme espace temps d'une machine :
  - ▶  $\text{MATCH}_p(v, L, M)$  : "Les 3 chiffres de  $v$  à la position  $\ell$  sont  $a$ ,  $b$  et  $c$  et correspondent aux 3 chiffres de  $v$  à la position  $m$  selon la fonction de transition  $\delta$  de la machine de Turing (où  $L = p^\ell$  et  $M = p^m$ )" :

$$\bigwedge_{(a,b,c,d,e,f) \in \text{LEGAL}} 3\text{DIGIT}_p(v, L, a, b, c) \wedge 3\text{DIGIT}_p(v, M, d, e, f).$$

- $(a, b, c, d, e, f) \in \text{LEGAL}$  désigne le fait que

|   |   |   |
|---|---|---|
| a | b | c |
| d | e | f |

est une fenêtre légale.

- On note évidemment ici,  $\bigwedge_{(a,b,c,d,e,f) \in \text{LEGAL}}$  pour la conjonction pour chacun des 6-uplets de LEGAL.

- Exprimer le diagramme espace temps d'une machine (suite) :
  - ▶  $\text{MOVE}_p(v, C, D)$  : "la suite  $v$  décrit<sup>2</sup> une suite de configurations successives de  $M$  de longueur  $c$  jusqu'à  $d$  (où  $C = p^c$  et  $D = p^d$ ) : toutes les paires de suites de 3-chiffres exactement écartées de  $c$  positions dans  $v$  se correspondent selon  $\delta$ " :

$$\forall y((\text{POWER}_p(y) \wedge y * p * p * C < D) \Rightarrow \text{MATCH}_p(v, y, y * C)).$$

---

2. On voit un tableau à deux dimensions comme un unique mot en mettant les lignes bout à bout.

■ Exprimer le diagramme espace temps d'une machine (suite) :

- ▶  $START_p(v, C)$  : "la suite  $v$  débute avec la configuration initiale de  $M$  sur l'entrée  $w = a_1 a_2 \dots a_n$  auxquelles on a ajouté des blancs  $B$  jusqu'à la longueur  $c$  ( $C = p^c$  ;  $n, p^i, 0 \leq i \leq n$  sont des constantes fixées qui ne dépendent que de  $w$ )" :

$$\bigwedge_{i=0}^n DIGIT_p(v, p^i, a_i) \wedge p^n < C$$

$$\wedge \forall y (POWER_p(y) \wedge p^n < y < C \Rightarrow DIGIT_p(v, y, B)).$$

- ▶  $HALT_p(v, D)$  : "La suite  $v$  possède un état d'acceptation quelque part" :

$$\exists y (POWER_p(y) \wedge y < D \wedge DIGIT_p(v, y, q_a)).$$

- ▶  $VALCOMP_{M,w}(v)$  : "La suite  $v$  est un calcul de  $M$  valide sur  $w$ " :

$$\exists c \exists d (POWER_p(c) \wedge c < d \wedge LENGTH_p(v, d) \\ \wedge START_p(v, c) \wedge MOVE_p(v, c, d) \wedge HALT_p(v, d)).$$

- $\gamma_{M,w}$  : “La machine  $M$  n’accepte pas  $w$ ” :

$$\neg \exists v \text{ VALCOMP}_{M,w}(v).$$

- La fonction  $f : (\langle M \rangle, w) \mapsto \gamma_{M,w}$  est bien calculable, car on peut bien produire  $\gamma_{M,w}$  à partir de  $M$  et  $w$  par un algorithme.
- Ce qui termine la preuve.

# Au menu

Quelques résultats amusants de la calculabilité

Théorèmes d'incomplétude de Gödel

Décidabilité de logiques

Esprit du second théorème

Introduction à la complexité

## Remarque importante

- On vient de prouver :  $Th(\mathbb{N})$  n'est pas récursivement énumérable.

### Théorème

*$Th(\mathbb{N})$  n'est donc pas décidable !*

## Remarque importante

- On vient de prouver :  $Th(\mathbb{N})$  n'est pas récursivement énumérable.

### Théorème

*$Th(\mathbb{N})$  n'est donc pas décidable !*

- Retour aux épisodes précédents : “Entscheidungsproblem” :

Peut-on décider mécaniquement  
si un énoncé est démontrable ou non ?

## Remarque importante

- On vient de prouver :  $Th(\mathbb{N})$  n'est pas récursivement énumérable.

### Théorème

*$Th(\mathbb{N})$  n'est donc pas décidable !*

- Retour aux épisodes précédents : “Entscheidungsproblem” :

Peut-on décider mécaniquement  
si un énoncé est démontrable ou non ?

- ▶ La réponse est : NON !



# Au menu

Quelques résultats amusants de la calculabilité

Théorèmes d'incomplétude de Gödel

Décidabilité de logiques

**Esprit du second théorème**

Introduction à la complexité

Exercice : Donner explicitement une formule  $\psi$  qui n'est pas prouvable.

## Exercice : Donner explicitement une formule $\psi$ qui n'est pas prouvable.

- Soit  $S$  la machine de Turing qui fait les choses suivantes :
  - ▶ sur toute entrée  $w$ 
    - obtenir par le théorème de récursion sa propre description  $\langle S \rangle$ .
    - construire la formule  $\psi = \gamma_{S,\epsilon}$ .
    - énumérer les formules prouvables tant que l'on a pas produit  $\gamma_{S,\epsilon}$ .
    - si l'étape d'avant finit par terminer, alors accepter.

## Exercice : Donner explicitement une formule $\psi$ qui n'est pas prouvable.

- Soit  $S$  la machine de Turing qui fait les choses suivantes :
  - ▶ sur toute entrée  $w$ 
    - obtenir par le théorème de récursion sa propre description  $\langle S \rangle$ .
    - construire la formule  $\psi = \gamma_{S,\epsilon}$ .
    - énumérer les formules prouvables tant que l'on a pas produit  $\gamma_{S,\epsilon}$ .
    - si l'étape d'avant finit par terminer, alors accepter.
- La formule  $\psi = \gamma_{S,\epsilon}$  de la deuxième étape n'est pas prouvable :

## Exercice : Donner explicitement une formule $\psi$ qui n'est pas prouvable.

- Soit  $S$  la machine de Turing qui fait les choses suivantes :
  - ▶ sur toute entrée  $w$ 
    - obtenir par le théorème de récursion sa propre description  $\langle S \rangle$ .
    - construire la formule  $\psi = \gamma_{S,\epsilon}$ .
    - énumérer les formules prouvables tant que l'on a pas produit  $\gamma_{S,\epsilon}$ .
    - si l'étape d'avant finit par terminer, alors accepter.
- La formule  $\psi = \gamma_{S,\epsilon}$  de la deuxième étape n'est pas prouvable :
  - ▶  $\psi$  est vraie ssi  $S$  n'accepte pas le mot vide.

## Exercice : Donner explicitement une formule $\psi$ qui n'est pas prouvable.

- Soit  $S$  la machine de Turing qui fait les choses suivantes :
  - ▶ sur toute entrée  $w$ 
    - obtenir par le théorème de récursion sa propre description  $\langle S \rangle$ .
    - construire la formule  $\psi = \gamma_{S,\epsilon}$ .
    - énumérer les formules prouvables tant que l'on a pas produit  $\gamma_{S,\epsilon}$ .
    - si l'étape d'avant finit par terminer, alors accepter.
- La formule  $\psi = \gamma_{S,\epsilon}$  de la deuxième étape n'est pas prouvable :
  - ▶  $\psi$  est vraie ssi  $S$  n'accepte pas le mot vide.
  - ▶ Si  $S$  trouve une preuve de  $\psi$ , alors  $S$  accepte le mot vide, et donc la formule  $\psi$  est fausse.
    - (si l'arithmétique est cohérente) on ne peut pas prouver de formule fausse.
    - ce cas ne peut donc pas se produire.

## Exercice : Donner explicitement une formule $\psi$ qui n'est pas prouvable.

- Soit  $S$  la machine de Turing qui fait les choses suivantes :
  - ▶ sur toute entrée  $w$ 
    - obtenir par le théorème de récursion sa propre description  $\langle S \rangle$ .
    - construire la formule  $\psi = \gamma_{S,\epsilon}$ .
    - énumérer les formules prouvables tant que l'on a pas produit  $\gamma_{S,\epsilon}$ .
    - si l'étape d'avant finit par terminer, alors accepter.
- La formule  $\psi = \gamma_{S,\epsilon}$  de la deuxième étape n'est pas prouvable :
  - ▶  $\psi$  est vraie ssi  $S$  n'accepte pas le mot vide.
  - ▶ Si  $S$  trouve une preuve de  $\psi$ , alors  $S$  accepte le mot vide, et donc la formule  $\psi$  est fausse.
    - (si l'arithmétique est cohérente) on ne peut pas prouver de formule fausse.
    - ce cas ne peut donc pas se produire.
  - ▶ Si  $S$  ne trouve pas une preuve de  $\psi$ , alors  $S$  n'accepte pas le mot vide :
    - et donc  $\psi$  est vraie sans être prouvable!!

## Exercice : Donner explicitement une formule $\psi$ qui n'est pas prouvable.

- Soit  $S$  la machine de Turing qui fait les choses suivantes :
  - ▶ sur toute entrée  $w$ 
    - obtenir par le théorème de récursion sa propre description  $\langle S \rangle$ .
    - construire la formule  $\psi = \gamma_{S,\epsilon}$ .
    - énumérer les formules prouvables tant que l'on a pas produit  $\gamma_{S,\epsilon}$ .
    - si l'étape d'avant finit par terminer, alors accepter.
- La formule  $\psi = \gamma_{S,\epsilon}$  de la deuxième étape n'est pas prouvable :
  - ▶  $\psi$  est vraie ssi  $S$  n'accepte pas le mot vide.
  - ▶ Si  $S$  trouve une preuve de  $\psi$ , alors  $S$  accepte le mot vide, et donc la formule  $\psi$  est fausse.
    - (si l'arithmétique est cohérente) on ne peut pas prouver de formule fausse.
    - ce cas ne peut donc pas se produire.
  - ▶ Si  $S$  ne trouve pas une preuve de  $\psi$ , alors  $S$  n'accepte pas le mot vide :
    - et donc  $\psi$  est vraie sans être prouvable!!
- Bref :  $\psi$  est vraie mais non-prouvable.



# Au menu

Quelques résultats amusants de la calculabilité

Théorèmes d'incomplétude de Gödel

Décidabilité de logiques

Esprit du second théorème

**Introduction à la complexité**

# Plus précisément

## Introduction à la complexité

### Exemples

Complexité d'un problème, d'un algorithme

Exemple d'Etude : Complexité du Tri

- Le problème *MAX*.

- ▶ On se donne une liste d'entiers naturels  $e_1, e_2, \dots, e_n$ .
- ▶ On veut calculer  $\max\{e_1, e_2, \dots, e_n\}$ .

- Le problème *TRI*.

- ▶ On se donne une liste d'entiers naturels  $e_1, e_2, \dots, e_n$ .
- ▶ On veut les trier.

## Une solution pour *MAX*

```
int [] T= new int[100];  
int r = max(T);
```

```
int max(int [] T) {  
    int max = T[1];  
    for (int j=2; j≤ T.length; j++) {  
        if (T[j] > max)  
            max = T[j];  
    }  
    return max;  
}
```

- Nombre de comparaisons (hors variable de boucle)  
 $C(n) = n - 1$ .
- Nombre d'affectations (hors variable de boucle)  $1 \leq B(n) \leq n$ .

$n$  = longueur de la liste.

# Plus précisément

## Introduction à la complexité

Exemples

Complexité d'un problème, d'un algorithme

Exemple d'Etude : Complexité du Tri

# Complexité d'un problème, d'un algorithme

- On fixe une mesure élémentaire :

- ▶ nombre de comparaisons,
- ▶ nombre d'affectations,
- ▶ mémoire utilisée,
- ▶ temps utilisé,
- ▶ ...

- Cette mesure associe à un algorithme  $\mathcal{A}$  et une entrée  $d$  une valeur

$$\text{Complexite}(\mathcal{A}, d).$$

- On cherche souvent à évaluer cette complexité en fonction d'un paramètre naturel  $n = \text{taille}(d)$ , représentatif des entrées.

- Exemple :

- ▶  $\mathcal{A}$  : l'algorithme itératif pour *MAX* précédent.
- ▶  $d$  : une liste donnée en entrée.
- ▶  $n = \text{taille}(d)$ , le nombre d'éléments dans la liste.
- ▶  $\text{Complexite}(\mathcal{A}, d)$ , le nombre de comparaisons de  $\mathcal{A}$  sur  $d$ .

- On peut alors parler de la complexité (au pire cas)
  - ▶ d'un algorithme (on fait varier seulement les entrées)

$$\text{Complexité}_{\mathcal{A}}(n) = \max_{d/\text{taille}(d)=n} \text{Complexité}(\mathcal{A}, d).$$

- ▶ d'un problème (on fait varier l'algorithme, et les entrées)

$$\inf_{\mathcal{A} \text{ correct}} \text{Complexité}_{\mathcal{A}}.$$

- On arrive parfois à parler de la complexité exacte.
- Il est souvent pertinent de se limiter à une étude asymptotique, ou à des bornes asymptotiques.
- Les bornes asymptotiques permettent de capturer l'essentiel de la complexité dans bon nombre de cas et de comparer simplement les algorithmes.

## Exemple d'étude exacte

- Etude du problème *MAX* en nombre de comparaisons :
  - ▶ Le problème *MAX* admet une solution avec  $n-1$  comparaisons.

```
int max(int [] T) {  
    int max = T[1];  
    for (int j=2; j≤ T.length; j++) {  
        if (T[j] > max)  
            max = T[j];  
    }  
    return max;  
}
```



## Exemple d'étude exacte

- Etude du problème *MAX* en nombre de comparaisons :
  - ▶ Le problème *MAX* admet une solution avec  $n-1$  comparaisons.

```
int max(int [] T) {  
    int max = T[1];  
    for (int j=2; j≤ T.length; j++) {  
        if (T[j] > max)  
            max = T[j];  
    }  
    return max;  
}
```

- ▶ Cet algorithme est optimal en nombre de comparaisons (parmi ceux qui fonctionnent avec affectations et comparaisons).

## Preuve

### Proposition

*Dans tout algorithme, tout élément autre que le maximum doit être comparé au moins une fois avec un élément qui lui est plus grand.*

- Conséquence : il faut au moins  $n - 1$  comparaisons pour résoudre *MAX*.

## Proposition

*Dans tout algorithme, tout élément autre que le maximum doit être comparé au moins une fois avec un élément qui lui est plus grand.*

- Preuve :
  - ▶ soit  $i_0$  le rang du maximum  $M$  retourné par l'algorithme sur une liste  $L = e_1.e_2.\dots.e_n$ .
  - ▶ Par l'absurde : soit  $j_0 \neq i_0$  tel que  $e_{j_0}$  n'est pas comparé avec un élément plus grand que lui.
  - ▶  $e_{j_0}$  n'a donc pas été comparé avec  $e_{i_0}$  le maximum
  - ▶ Considérer la liste  $L' = e_1.e_2.\dots.e_{j_0-1}.M+1.e_{j_0+1}.\dots.e_n$  obtenue à partir de  $L$  en remplaçant l'élément d'indice  $j_0$  par  $M+1$ .
  - ▶ L'algorithme effectue exactement les mêmes comparaisons sur  $L$  et  $L'$ , sans comparer  $L'[j_0]$  avec  $L'[i_0]$  et donc retournera  $L'[i_0]$ , ce qui est incorrect.
- Conséquence : il faut au moins  $n-1$  comparaisons pour résoudre MAX.

# Notation de Landau (Pire Cas)

## ■ Notation de Landau :

- ▶  $f(n) = O(g(n))$  si et seulement si il existe deux constantes positives  $n_0$  et  $B$  telles que

$$\forall n \geq n_0, |f(n)| \leq Bg(n)$$

Ce qui signifie que  $f$  ne croît pas plus vite que  $g$ .

## ■ Exemple :

- ▶ Le problème *MAX* admet une solution itérative en  $O(n)$  affectations.
- ▶ L'algorithme itératif précédent fonctionne en  $O(n)$  affectations.

## ■ Un algorithme

- ▶ en temps  $O(1)$  effectue un nombre constant d'opérations.
- ▶ en temps  $O(n)$  est un algorithme (au pire) linéaire.
- ▶ en temps  $O(n^k)$  est un algorithme (au pire) polynomial.

# Notation de Landau

## ■ Notation de Landau (suite)

- ▶  $f(n) = \Omega(g(n))$  si et seulement si il existe deux constantes positives  $n_0$  et  $B \neq 0$  telles que

$$\forall n \geq n_0, |f(n)| \geq Bg(n)$$

- ▶  $f(n) = \Theta(g(n))$  si et seulement si il existe trois constantes positives  $n_0$ ,  $B \neq 0$  et  $C$  telles que

$$\forall n \geq n_0, Bg(n) \leq |f(n)| \leq Cg(n)$$

## ■ Exemple :

- ▶ Le problème *MAX* nécessite  $\Theta(n)$  comparaisons.
- ▶ (à venir) Trier nécessite  $\Omega(n \log n)$  comparaisons.

## Complexité d'un algorithme en moyenne

- Si on fixe une distribution  $\pi$  sur les entrées  $d$ , il est parfois possible d'évaluer la complexité en moyenne d'un algorithme :

$$\text{Complexité-Moyenne}_{\mathcal{A}}(n) = \sum_{d/\text{taille}(d)=n} \pi(d) \text{Complexité}(\mathcal{A}, d)$$

- Exemple : pour l'algorithme  $\mathcal{A}$  suivant.

```
static boolean Dans(int[] T,x) {  
  for (int j=1; j≤ T.length; j++) {  
    if (T[j] == x) return true;  
  }  
  return false; }  
}
```

Nombre de comparaisons (entre entiers) :  $k$ , où  $k$  est le rang de l'élément lorsqu'il est dans la liste,  $n$  sinon.

- Si on suppose que les entrées sont les listes permutations de  $\{1, 2, \dots, n\}$ , et qu'elles sont équiprobables,

$$\begin{aligned} \text{Complexite - Moyenne}_{\mathcal{A}}(n) &= \text{Esperance}_{d/\text{taille}(d)=n}[k] \\ &= \sum_{i=1}^n i \times \text{Proba}(k = i) \end{aligned}$$

Puisque les permutations sont équiprobables,  
 $\text{Proba}(k = i) = 1/n$ .

$$\begin{aligned} \text{Complexite - Moyenne}_{\mathcal{A}}(n) &= \sum_{i=1}^n i \frac{1}{n} \\ &= \frac{n(n+1)}{2n} \\ &= \frac{n+1}{2} \end{aligned}$$

- (poly inf421) D'autres exemples plus compliqués.
- (joli exercice) Pour  $\mathcal{A}$  algorithme itératif précédent pour  $MAX$ ,  $\text{Complexite - Moyenne}_{\mathcal{A}}(n)$  en affectations entre variables entières est en  $\Theta(\log n)$  (de l'ordre de  $H_n = 1 + \frac{1}{2} + \dots + \frac{1}{n}$ ).

# Plus précisément

## Introduction à la complexité

Exemples

Complexité d'un problème, d'un algorithme

Exemple d'Etude : Complexité du Tri



# Complexité du problème du tri

- Tout algorithme de tri

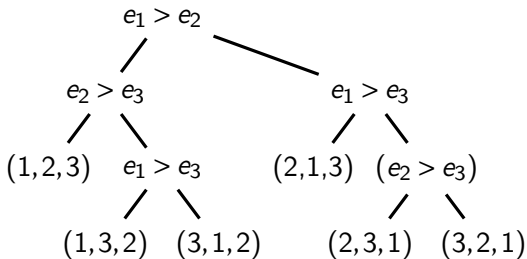
- ▶ qui fonctionne par comparaisons,
- ▶ qui n'a pas d'autres informations sur les données,

effectue  $\Omega(n \log n)$  comparaisons dans le pire des cas.

- Autrement dit, la complexité du problème du tri (avec ces hypothèses) est en  $\Theta(n \log n)$ .
- Le tri par fusion est parmi les tris optimaux en  $O(n \log n)$ .

## Preuve de la borne inférieure $\Omega(n \log n)$ : 1/2

- A un algorithme on peut associer un arbre de décision :



- ▶ En chaque noeud qui n'est pas une feuille : une comparaison effectuée par l'algorithme.
  - La racine est la première comparaison.
  - Fils gauche : récursivement ce qui se passe alors pour un résultat positif.
  - Fils droit : récursivement ce qui se passe alors pour un résultat négatif.
- ▶ En chaque feuille, le résultat produit.

## Preuve de la borne inférieure $\Omega(n \log n)$ : 2/2

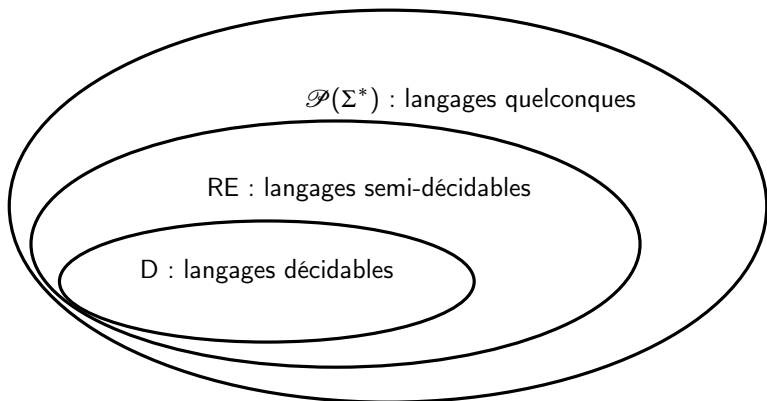
- Un arbre binaire de hauteur  $h$  a au plus  $2^h$  feuilles.
  - ▶ par récurrence.
- Les feuilles doivent contenir au moins les  $n!$  permutations, et donc  $h \geq \log(n!) = \Omega(n \log n)$  par la formule de Stirling.

## La suite de ce cours

- On souhaite parler d'une ressource particulière : le **temps de calcul**.

## La suite de ce cours

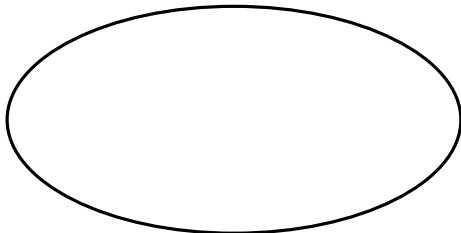
- On souhaite parler d'une ressource particulière : le **temps de calcul**.
- Objectif :
  - ▶ distinguer ce qui est raisonnable de ce qui n'est pas raisonnable en termes de temps de calcul.



## La suite de ce cours

- On souhaite parler d'une ressource particulière : le **temps de calcul**.
- Objectif :
  - ▶ distinguer ce qui est raisonnable de ce qui n'est pas raisonnable en termes de temps de calcul.

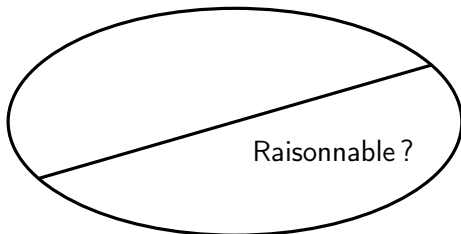
D : langages décidables.



## La suite de ce cours

- On souhaite parler d'une ressource particulière : le **temps de calcul**.
- Objectif :
  - ▶ distinguer ce qui est raisonnable de ce qui n'est pas raisonnable en termes de temps de calcul.

D : langages décidables.



## ANNEXES



## Preuve du théorème de récursion

- Soit  $T$  une machine de Turing calculant la fonction  $t : T$  prend en entrée une paire  $(u, w)$  et produit en sortie un mot  $t(u, w)$ .
- On considère les machines suivantes :
  1. Étant donné un mot  $w$ , la machine  $Print_w$  prend en entrée un mot  $u$  et termine avec le résultat  $(w, u)$ .
  2. Pour une entrée  $w'$  de la forme  $(\langle X \rangle, w)$ , la machine  $B$ 
    - 2.1 calcule  $(\langle Print_{\langle X \rangle} X \rangle, w)$ , où  $Print_{\langle X \rangle} X$  désigne la machine qui compose  $Print_{\langle X \rangle}$  avec  $X$ ,
    - 2.2 puis passe le contrôle à la machine  $T$ .
- On considère alors la machine  $R$  donnée par  $Print_{\langle B \rangle} B$ .
- Déroulons le résultat  $r(w)$  de cette machine  $R$  sur une entrée  $w$  :
  - ▶ la machine  $Print_{\langle B \rangle}$  produit en sortie  $(\langle B \rangle, w)$ .
  - ▶ La composition par  $B$  produit alors le codage de  $(\langle Print_{\langle B \rangle} B \rangle, w)$ , et passe le contrôle à  $T$ .
  - ▶ Ce dernier produit alors  $t(\langle Print_{\langle B \rangle} B \rangle, w) = t(\langle R \rangle, w) = r(w)$ .

## Preuve du théorème du point fixe de Kleene

- Considérons une fonction  $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  telle que  $t(\langle A \rangle, x)$  soit le résultat de la simulation de la machine  $f(A)$  sur l'entrée  $x$ .
- Par le théorème précédent, il existe une machine  $R$  qui calcule une fonction  $r$  telle que  $r(w) = t(\langle R \rangle, w)$ .
- Par construction  $A^* = R$  et  $f(A^*) = f(R)$  ont donc même valeur sur  $w$  pour tout  $w$ .

# Pourquoi trier ?

- Trier est une opération naturelle.
- Travailler sur des données triées est parfois plus efficace.
  - ▶ Exemple :
    - Rechercher un élément dans un tableau à  $n$  éléments :  $O(n)$  (utiliser le principe de la fonction *Dans* précédente sur les listes).
    - Rechercher un élément dans un tableau trié à  $n$  éléments :  $O(\log n)$  (par dichotomie).

# Pourquoi trier ?

- Trier est une opération naturelle.
- Travailler sur des données triées est parfois plus efficace.
  - ▶ Exemple :
    - Rechercher un élément dans un tableau à  $n$  éléments :  $O(n)$  (utiliser le principe de la fonction *Dans* précédente sur les listes).
    - Rechercher un élément dans un tableau trié à  $n$  éléments :  $O(\log n)$  (par dichotomie).
    - Trier devient intéressant dès que le nombre de recherches est en  $\Omega(\text{Complexite}(tri)/n)$ .

## Recherche par dichotomie

```
static boolean trouve(int[] T, int v, int min, int max){
    if(min ≥ max) // vide
        return false;
    int mid = (min + max) / 2;
    if (T[mid] == v) return true;
    else if (T[mid] > v) return trouve(T, v, min, mid);
    else return trouve(T, v, mid + 1, max);
}
```

Nombre de comparaisons

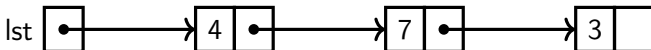
- $C(1) = 2$ ,
- $C(n) \leq 2 + C(n/2)$ , pour  $n$  pair,
- $C(n) \leq 2\log n + 2$ , pour  $n$  puissance de 2.

$$C(n) = O(\log n),$$

dans le cas général.

```
class Liste {  
    int contenu;  
    Liste suivant;  
    Liste (int x, Liste a) {  
        contenu = x;  
        suivant = a;  
    }  
}
```

```
Liste lst = new Liste(4, new Liste (7, new Liste (3, null)));
```



## Trier par insertion

- Méthode souvent utilisée pour trier un jeu de cartes.
- Etape préliminaire : savoir insérer un élément  $x$  dans une liste triée  $e_1.e_2.\dots.e_n$ , avec  $e_j \leq e_{j+1}$ .
  - ▶ Equations récurrentes :

$$\begin{aligned} \text{Insere}(x, \emptyset) &= (x, \emptyset) \\ \text{Insere}(x, (a, L)) &= (x, (a, L)) && \text{si } x \leq a \\ \text{Insere}(x, (a, L)) &= (a, \text{Insere}(x, L)) && \text{sinon} \end{aligned}$$

- ▶ Correction : par induction, si on suppose  $(a, L)$  triée, le résultat est bien trié.

```
static Liste Insere(int x, Liste a) {
    if (a==null) return new Liste(x, null);
    if (x ≤ a.contenu) return new Liste(x, a);
    return new Liste(a.contenu, Insere(x, a.suivant));
}
```

## Tri par insertion

```
static Liste Trie(Liste p) {  
    Liste r = null;  
    for (; p != null; p = p.suivant)  
        r = Insere(p.contenu,r);  
    return r;}  
}
```

- Complexité  $I(n)$  de Insere en comparaisons :  $I(n) = O(n)$ .
- Complexité  $C(n)$  de Trie en comparaisons :  
 $C(n) \leq nI(n) = O(n^2)$ .
- Le pire cas est atteint lorsqu'on trie une liste déjà triée, et mène à  $\Omega(n^2)$  comparaisons. La complexité du tri par insertion est donc bien en  $\Theta(n^2)$ .



# Tri par fusion

- Fusion :
  - ▶ Construire une liste triée qui contienne l'union des éléments de deux listes triées.
- Exemple :  $Fusion(1.4.5.7, 2.4.9) = 1.2.4.4.5.7.9$
- Equations récursives de  $Fusion(X, Y)$  :

$$Fusion(X, \emptyset) = X$$

$$Fusion(\emptyset, Y) = Y$$

$$Fusion((a, L), (b, M)) = (a, Fusion(L, (b, M))) \quad \text{si } a \leq b$$

$$Fusion((a, L), (b, M)) = (b, Fusion((a, L), M)) \quad \text{sinon}$$

- Correction : par induction, si on suppose  $(a, L)$  et  $(b, M)$  triées, le résultat est correct.

# Fusion

```
static Liste Fusion(Liste a, Liste b) {  
    if (a == null)  
        return b;  
    if (b == null)  
        return a;  
    if (a.contenu < b.contenu)  
        return new Liste(a.contenu, Fusion(a.suivant, b));  
    else  
        return new Liste(b.contenu, Fusion(a, b.suivant));  
}
```

- Complexité en nombre de comparaisons :  
 $O(\text{longueur}(a) + \text{longueur}(b))$ .

# Tri fusion

- Une liste de 0 ou 1 élément est triée.
- Toute autre liste  $L$ 
  - ▶ peut se découper en deux sous-listes  $L_1$  et  $L_2$  de même taille (à 1 près).
  - ▶ Les sous-listes  $L_1$  et  $L_2$  sont triées récursivement,
  - ▶ puis fusionnées.

$$TriFusion(L) = Fusion(TriFusion(L_1), TriFusion(L_2)).$$

## Paradigme "Diviser pour régner"

- Supposons  $n$  puissance de 2.
- Complexité  $C(n)$  en nombre de comparaisons en  $O(\text{Fusion}) + 2C(n/2)$ , soit

$$C(n) \leq Kn + 2C(n/2)$$

pour une constante  $K$ , ce qui mène à

$$C(n) \leq O(n \log n).$$

## Paradigme “Diviser pour régner”

- Supposons  $n$  puissance de 2.
- Complexité  $C(n)$  en nombre de comparaisons en  $O(\text{Fusion}) + 2C(n/2)$ , soit

$$C(n) \leq Kn + 2C(n/2)$$

pour une constante  $K$ , ce qui mène à

$$C(n) \leq O(n \log n).$$

- Note :
  - ▶ comment résoudre une telle récurrence : poser  $n = 2^p$  et faire le changement de variable  $C'(p) = C(2^p)$ .
  - ▶  $C'(p) \leq K2^p + 2C'(p-1)$ ,
  - ▶ donc  $C'(p) \leq O(2^p p)$ .

## Tri fusion

```
static Liste MergeSort(Liste l) {
    Liste l1 = null, l2 = null;
    boolean even = true ;
    for ( ; l != null ; l = l.suivant ) {
        if (even) {
            l1 = new Liste (l.contenu, l1) ;
        } else {
            l2 = new Liste (l.contenu, l2) ;
        }
        even = !even ;
    }
    if (l2==null) return l1;
    return Fusion(MergeSort(l1), MergeSort(l2));
}
```

- Note : L1 et L2 sont en fait ici dans l'ordre inverse des éléments pairs et impairs, par commodité.
- Note : le tri fusion fait *toujours* de l'ordre de  $n \log n$  comparaisons, et pas seulement au pire cas.