



# Calculabilité, logique et complexité

## Chapitre 3

### Calculabilité : résultats fondamentaux

# Chapitre 3 : Calculabilité : résultats fondamentaux

1. Programmes et fonctions
2. Fonctions calculables, ensembles rékursifs et récursivement énumérables
3. Thèse de Church-Turing
4. Programmes et fonctions
5. Existence de fonctions non calculables
6. Problème de l'arrêt : un exemple de fonction non calculable
7. Insuffisance des fonctions totales
8. Extension de fonctions partielles
9. Théorème de Rice
10. Théorème de la paramétrisation
11. Théorème du point fixe
12. Autres problèmes non calculables
13. Codage et représentation
14. Nombres calculables

# Acquis d'apprentissage (1)

A l'issue de ce chapitre, les étudiants seront capables de

- Comprendre, expliquer et appliquer les notions de fonction calculable, d'ensemble récursif et d'ensemble récursivement énumérable ainsi que leurs propriétés
- Expliquer le rôle d'une numérotation des programmes et de la fonction qu'ils calculent
- Expliquer et démontrer pourquoi le problème de l'arrêt est non calculable ; en mesurer les conséquences
- Enoncer, expliquer et démontrer le théorème de Hoare-Allison (insuffisance des fonctions totales), analyser ce résultat ainsi que ses conséquences
- Expliquer la limitation liée à l'impossibilité d'étendre certaines fonctions partielles calculables en fonctions totales calculables
- Décrire et appliquer la méthode de raisonnement de la réduction à halt pour prouver la non calculabilité d'une fonction
- Expliquer la méthode de preuve de non calculabilité de réduction à halt ; appliquer cette méthode à des nouveaux exemples de fonctions

# Acquis d'apprentissage (2)

A l'issue de ce chapitre, les étudiants seront capables de

- Enoncer, expliquer et démontrer le théorème de Rice ; analyser ce résultat ainsi que ses conséquences, déterminer quand ce théorème peut être appliqué pour démontrer la non calculabilité d'une fonction ; appliquer et justifier son application dans de nouvelles situations
- Expliquer pourquoi une fonction peut être vue comme un transformateur de programmes
- Enoncer et expliquer le théorème S-m-n ; expliquer la différence entre S-m-n et sa version affaiblie S
- Enoncer, expliquer et démontrer le théorème du point fixe; analyser pourquoi ce théorème est central en calculabilité
- Enoncer quelques problèmes importants non calculables : correspondance de Post, équation diophantines
- Décrire ce qu'est un nombre réel calculable

# 1. Programmes et fonctions

Qu'est-ce qu'un algorithme ?

*Exemples d'algorithmes*

- Multiplication par tableau (11ème siècle)
- Algorithme d'Euclide (3ème siècle av. J.C.)

## Algorithme: première définition



Un algorithme est une procédure qui peut être appliquée à n'importe quelle donnée (appartenant à une classe de données symboliques) et qui a pour effet de produire un résultat (appartenant à une classe de données symboliques)

- un algorithme *n'est pas* une fonction
- un algorithme *calcule* une fonction

# 1. Programmes et fonctions

## Caractéristiques d'un algorithme



- Ensemble *fini* d'instructions (**Programme**)
- Existence d'un “*calculateur*” qui peut exécuter ces instructions

Autres aspects *non considérés* :

pas de limite

- à la taille des données
- à la taille des instructions
- à la taille de la mémoire disponible, mais son utilisation est finie

# 1. Programmes et fonctions

## Approche suivie

Pour être totalement rigoureux, la théorie de la calculabilité devrait se construire à partir d'un *modèle* ou *formalisme* définissant la notion de "calculable"

→ présentation très technique, difficulté de comprendre la portée des résultats

- Calculabilité sans formalisme (ou plutôt basé sur la notion intuitive de langage de programmation)
  - pas de formalisme  $\neq$  non rigoureux
- Présentation de différents modèles de calcul
  - le formalisme n'influence pas la calculabilité
- Analyse de la calculabilité

# 1. Programmes et fonctions

## L'univers des programmes

Notre approche : calculabilité basée sur la notion de programmes exprimés dans un langage de programmation (sans limite de mémoire)

Par exemple choix du *langage Java*

*Les résultats de calculabilités sont indépendant du langage choisi*

## L'univers des fonctions

Limitation (pour l'instant) aux fonctions

- de  $N \rightarrow N$  
- ou de manière plus générale de  $N^n \rightarrow N$  



## 2. Fonctions calculables, ensembles récurrents et récursivement énumérables

### 1. Fonction calculable

Une fonction  $f : N \rightarrow N$  est **calculable** ssi il existe un programme Java qui, recevant comme donnée (une représentation décimale de) n'importe quel nombre naturel  $x$ ,

- fourni (tôt ou tard) comme résultat (une représentation décimale de)  $f(x)$  si  $f(x)$  est défini
- ne se termine pas (ou message d'erreur) si  $f(x) = \perp$

#### Remarques

- Existence d'un programme  $\neq$  capacité d'écrire ce programme
- Fonction non calculable : *il n'existe pas de programme* pour la calculer !
- Définition généralisable pour une fonction  $f$  de  $N^n \rightarrow N$
- Fonction partielle calculable : fonction partielle qui est calculable
- Fonction totale calculable : fonction totale qui est calculable



## 2. Fonctions calculables, ensembles récurrents et récursivement énumérables

### 2. Ensemble récurrent

Soit  $A \subseteq \mathbb{N}$

A est **récursif** si il existe un programme Java qui, recevant comme donnée (une représentation décimale de) n'importe quel nombre naturel  $x$ , fourni (tôt ou tard) comme résultat (une représentation décimale de)

- 1 si  $x \in A$
- 0 si  $x \notin A$

### Remarques

- algorithme *décide* si  $x$  est dans  $A$  ou non
- Le programme se termine toujours avec une réponse (1 ou 0)
- récurrent  $\neq$  récursivité

## 2. Fonctions calculables, ensembles rékursifs et rékursivement énumérables

### 2. Ensemble rékursivement énumérable

Soit  $A \subseteq \mathbb{N}$

A est **rékursivement énumérable** si il existe un programme Java qui, recevant comme donnée (une représentation décimale de) n'importe quel nombre naturel  $x$ , fourni (tôt ou tard) comme résultat (une représentation décimale de) 1 ssi  $x \in A$

#### Remarques

- si  $x \notin A$ , le programme fourni un résultat ( $\neq 1$ ) ou *ne se termine pas*

## 2. Fonctions calculables, ensembles rékursifs et récursivement énumérables

### Autres définitions

Fonction caractéristique de  $A \subseteq \mathbb{N}$

$\chi_A : \mathbb{N} \rightarrow \mathbb{N}$  tel que  $\chi_A(x) = 1$  si  $x \in A$   
 $0$  si  $x \notin A$

A est un **ensemble rékursif** ssi  $\chi_A$  est une fonction (totale) calculable



A est un **ensemble récursivement énumérable** ssi il existe une fonction  $f$  calculable ( $f$  éventuellement partielle) tel que  $A = \text{dom}(f)$ ,

A est un **ensemble récursivement énumérable** ssi  $A$  est vide, ou  $A = \text{image}(f)$ , avec  $f$  fonction **totale** calculable


- $f$  est une fonction d'énumération de  $A$
- énumération effective de  $A$  :  $f(0), f(1), f(2), \dots, f(k), \dots$
- possibilité de répétitions

## 2. Fonctions calculables, ensembles rékursifs et rékursivement énumérables

### Questions

- Existe-t-il des fonctions non calculables ?
- Existe-t-il des ensembles non rékursifs ?
- Existe-t-il des ensembles non rékursivement énumérables ?
- A quoi servent les fonctions partielles ?

### 3. Propriétés

- $A$  rékursif  $\Rightarrow A$  rékursivement énumérable
- $A$  rékursif  $\Rightarrow (N \setminus A)$  rékursivement énumérable
- $A$  rékursif  $\Rightarrow (N \setminus A)$  rékursif 
- $A$  rékursivement énumérable et  $(N \setminus A)$  rékursivement énumérable  $\Rightarrow A$  rékursif
- $A$  fini  $\Rightarrow A$  rékursif
- $(N \setminus A)$  fini  $\Rightarrow A$  rékursif

# 3. Thèse de Church-Turing

Définition de fonction calculable permet de montrer qu'une fonction est calculable

Comment montrer qu'une fonction n'est pas calculable ?




Nécessité d'une définition de calculabilité couvrant la totalité des "fonctions calculables"

Utilisation d'un modèle particulier (programme Java) peut-être restrictif ?

## Exemples de modèle

- logique mathématique (Gödel)
- lambda calcul (Church)
- fonction récursive (Kleene)
- machine de Turing
- algorithme de Markov
- système de Post
- langages de programmation

# 3. Thèse de Church-Turing

1. Aucun modèle de la notion de fonction calculable n'est plus puissant que les Machines de Turing 
2. Toute fonction calculable (au sens intuitif) est calculable par une machine de Turing
3. Toutes les définitions formelles de la calculabilité connues à ce jour sont équivalentes (Théorème) 
4. Toutes les formalisations de la calculabilité établies par la suite seront équivalentes aux définitions connues 

Seul (3) peut être démontré

(1), (2) et (4) sont des *thèses* ou conjectures

Rien n'est jamais venu infirmer cette thèse

Thèse universellement reconnue comme vraie

# 3. Thèse de Church-Turing

## Version moderne de la calculabilité :

5. Une fonction est calculable s'il existe un programme d'ordinateur qui calcule cette fonction

Une fonction est non calculable s'il n'existe pas de programme (Java) permettant de calculer cette fonction.

Si non claculable en Java, alors impossible à calculer quel que soit le formalisme de calcul utilisé !




## 4. Numérotation

Notre approche : calculabilité basée sur la notion de programmes exprimés dans un langage de programmation (sans limite de mémoire)

Choix (arbitraire et sans conséquence) du *langage Java*

Soit  $P$  l'ensemble des programmes (Java) syntaxiquement corrects, qui reçoivent 1 données entières et qui impriment un résultat entier

- $P$  est un ensemble (infini dénombrable) récursif 
- $P = P_0, P_1, P_2, \dots, P_k, \dots$  (sans répétition)
- Il existe une fonction de numérotation de programmes  $f : \mathbb{N} \rightarrow P$  telle que
  - $f(k) = P_k$
  - $f$  calculable
  - $k$  et  $P_k$  sont deux représentations distinctes d'un *même* objet
- Généralisable pour des programmes avec plusieurs arguments

## 4. Numérotation

Choisissons une telle fonction

- $P_k$  dénote le programme  $k$  dans  $P$

$P_k$  : programme

- $\varphi_k$  dénote par convention la fonction calculée par  $P_k$

$\varphi_k$  : fonction

- $\varphi_k : N \rightarrow N$

- Généralisable pour des programmes à plusieurs arguments  $\varphi_k^{(n)} : N^n \rightarrow N$

Exercice: construire un algorithme calculant  $f$

- Une numérotation permet de désigner chaque programme par son numéro
- Une numérotation permet de désigner la fonction calculée ( $\varphi_k$ ) par un programme ( $P_k$ )

## 5. Existence de fonctions non calculables

- Nombre de fonctions de  $N$  dans  $N$  : non dénombrable
  - Nombre de programmes Java: dénombrable
- il existe (beaucoup) de fonctions non calculables

Une fonction est définie par sa table (infinie)

Nous ne pouvons nous intéresser qu'aux fonctions dont la table peut être définie de manière *finie*

Restriction aux fonctions pouvant être définies de manière finie

**Infinité dénombrable** de telles fonctions

**Question** : si la table d'une fonction peut être définie de manière finie, cette fonction est-elle nécessairement calculable ?

# 6. Problème de l'arrêt : un exemple de fonction non calculable

## Problème de l'arrêt



Soit la fonction  $\text{halt} : P \times \mathbb{N} \rightarrow \mathbb{N}$  telle que

$\text{halt}(n, x) = 1$  si  $\varphi_n(x) \neq \perp$

0 sinon

$\text{halt}(n, x) = 1$  si l'exécution du programme  $P_n(x)$   
se termine (i.e. imprime un résultat)

$\text{halt}(n, x) = 0$  sinon

- $\text{halt}$  est une fonction bien définie
- la fonction  $\text{halt}$  est totale
- table infinie, mais décrite de manière finie

La fonction  $\text{halt}$  est-elle calculable (dans  $P$ ) ?

# 6. Problème de l'arrêt : un exemple de fonction non calculable

## Théorème

- $\text{halt}$  n'est pas calculable

*Preuve :*

Supposons  $\text{halt}$  calculable

### 1. Construction d'une table

	0	1	2	...	k	...
$P_0$	$\text{halt}(0,0)$	$\text{halt}(0,1)$	$\text{halt}(0,2)$	...	$\text{halt}(0,k)$	...
$P_1$	$\text{halt}(1,0)$	$\text{halt}(1,1)$	$\text{halt}(1,2)$	...	$\text{halt}(1,k)$	...
$P_2$	$\text{halt}(2,0)$	$\text{halt}(2,1)$	$\text{halt}(2,2)$	...	$\text{halt}(2,k)$	...
:	:	:	:	:	:	:
$P_k$	$\text{halt}(k,1)$	$\text{halt}(k,2)$	$\text{halt}(k,3)$	...	$\text{halt}(k,k)$	...
:	:	:	:	:	:	:

## 6. Problème de l'arrêt : un exemple de fonction non calculable

### 2. Sélection de la diagonale

diag :  $\text{halt}(0,0), \text{halt}(1,1), \dots, \text{halt}(k,k), \dots$

$\text{diag}(n) = \text{halt}(n,n)$

### 3. Modification de diag

$\text{diag\_mod}(n) = 1 \quad \text{si} \quad \text{halt}(n,n) = 0$

$\perp \quad \text{si} \quad \text{halt}(n,n) = 1$

- $\text{diag\_mod}$  est calculable



## 6. Problème de l'arrêt : un exemple de fonction non calculable

### 4. Contradiction

- $\text{diag\_mod} = P_d$  (pour un certain  $d$ )
- quelle est la valeur de  $\text{diag\_mod}(d)$  ?

$$\begin{aligned} \text{diag\_mod}(d) &= 1 && \text{si } \text{halt}(d,d) = 0 \\ &\perp && \text{si } \text{halt}(d,d) = 1 \end{aligned}$$

si  $\text{diag\_mod}(d) = 1$

alors  $\text{diag\_mod}(d) = 1$  alors  $\text{halt}(d,d) = 0$

alors  $P_d(d)$  ne se termine pas

alors  $\text{diag\_mod}(d)$  ne se termine pas alors  $\text{diag\_mod}(d) = \perp$

si alors  $\text{diag\_mod}(d) = \perp$

alors  $\text{halt}(d,d) = 1$  alors  $P_d(d)$  se termine

alors  $\text{diag\_mod}(d)$  se termine

alors  $\text{diag\_mod}(d) = 1$

## 6. Problème de l'arrêt : un exemple de fonction non calculable

### 5. Conclusion

- $\text{diag\_mod}$  n'est pas calculable
- $\text{diag}$  n'est pas calculable
- $\text{halt}$  n'est pas calculable



# 6. Problème de l'arrêt : un exemple de fonction non calculable

## Conclusion



- Aucun algorithme ne permet de déterminer pour tout programme  $P_n$  et donnée  $x$  si  $P_n(x)$  se termine ou non
- Seule possibilité serait d'avoir un langage de programmation dans lequel tous les programmes se terminent.  
La fonction *halt* est alors calculable pour les programmes de ce formalisme
- *halt* non calculable ne signifie pas que pour un programme  $k$  donné,  $halt(k,x)$  est non calculable



La fonction halt joue un rôle central en calculabilité

## 6. Problème de l'arrêt : un exemple de fonction non calculable

### Existence d'un ensemble non récursif

- $HALT = \{ (n, x) \mid halt(n, x) = 1 \}$    
=  $\{ (n, x) \mid P_n(x) \text{ se termine} \}$
- $K = \{ n \mid (n, n) \in HALT \}$    
=  $\{ n \mid halt(n, n) = 1 \}$   
=  $\{ n \mid diag(n) = 1 \}$   
=  $\{ n \mid P_n(n) \text{ se termine} \}$

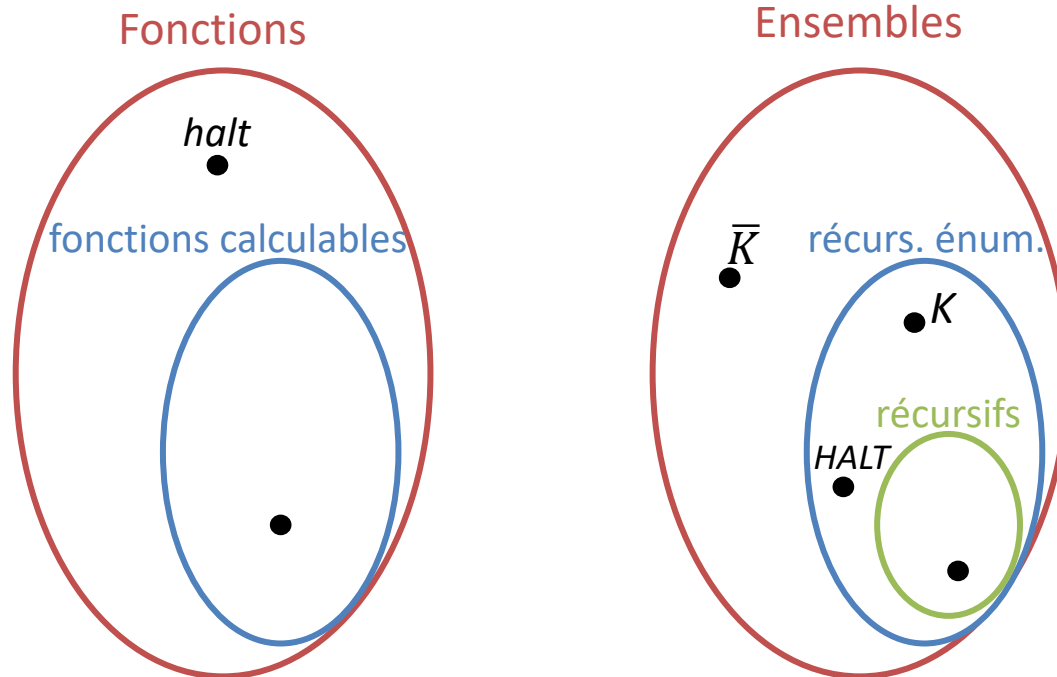
## 6. Problème de l'arrêt : un exemple de fonction non calculable

### Propriétés

- K et HALT ne sont pas rékursifs
- K et HALT sont récursivement énumérables
- $\overline{HALT} = \{ (n,x) \mid halt(n,x) = 0 \}$   
 $= \{ (n,x) \mid P_n(x) \text{ ne se termine pas} \}$   
 $\overline{HALT}$  n'est pas récursivement énumérable
- $\overline{K} = \{ n \mid (n,n) \in HALT \}$   
 $= \{ n \mid halt(n,n) = 0 \}$   
 $= \{ n \mid diag(n) = 0 \}$   
 $= \{ n \mid P_n(n) \text{ ne se termine pas} \}$   
 $\overline{K}$  n'est pas récursivement énumérable

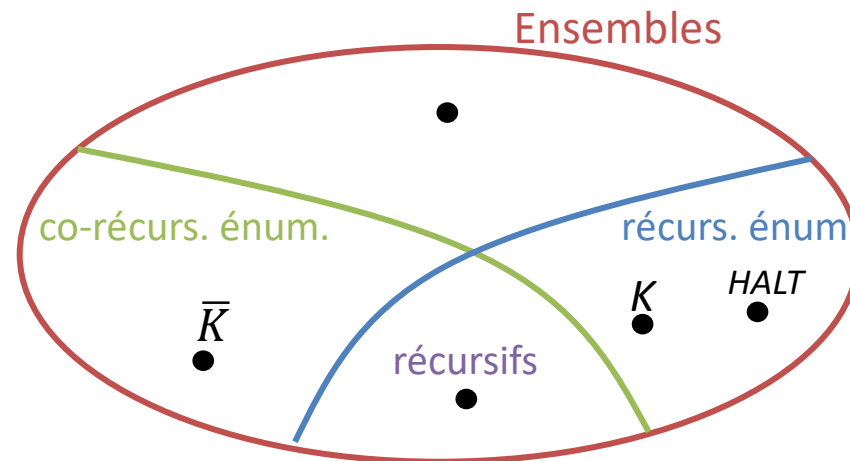
## 6. Problème de l'arrêt : un exemple de fonction non calculable

### Structures des fonctions ensembles



## 6. Problème de l'arrêt : un exemple de fonction non calculable

### Ensemble co-récurсивement énumérable



# 6. Problème de l'arrêt : un exemple de fonction non calculable

## Diagonalisation: principes

### 1. Construction d'une table

Liste de tous les grands mathématiciens

- DE MORGAN
- ABEL
- BOOLE
- BROUWER
- SIERPINSKI
- WEIERSTRASS

### 2. Sélection de la diagonale

- DBOUPS

### 3. Modification de cet élément CANTOR

### 4. Contradiction

- CANTOR n'est pas dans la liste

## 6. Problème de l'arrêt : un exemple de fonction non calculable

### 5. Conclusion

- Soit on sait que la liste est complète  
→ CANTOR n'est pas un grand mathématicien (problème de l'arrêt)
- Soit on sait que CANTOR est un grand mathématicien  
→ la liste n'est pas complète ( $R$  non énumérable)


# 7. Insuffisance des fonctions totales

Pourquoi ne pas limiter un langage de programmation aux seules fonctions totales ? 

- Les problèmes “pratiques” sont des fonctions totales
- La fonction halt de ce langage est calculable (fonction constante 1)


Soit un langage de programmation (non trivial)  $Q$  qui ne calcule que des fonctions totales

$Q$  ensemble des programmes de ce langage

- $Q = Q_0, Q_1, Q_2, \dots, Q_k, \dots$  

$\varphi'_k$  dénote la fonction calculée par  $Q_k$

- tous les programmes de  $Q$  se terminent toujours
- $Q$  possède un interpréteur calculable (fonction totale)
- $\text{interpret}(n, x) = \varphi'_n(x)$   
(résultat de l'exécution du programme  $Q_n$  sur la donnée  $x$ )


*mini Java*  
*while*  
*goto*  
*récurssive* 



# 7. Insuffisance des fonctions totales

## Théorème de Hoare-Allison

Soit un langage de programmation (non trivial)  $Q$  qui ne calcule que des fonctions totales

- L'interpreteur  $interpret(n,x)$  de ce langage  $Q$  est calculable (par exemple en Java)
- La fonction  $halt(n,x)$  pour le langage de programmation  $Q$  est calculable (fonction constante 1)
- Mais  $interpret(n,x)$  n'est pas calculable dans  $Q$  

# 7. Insuffisance des fonctions totales

## Preuve

Supposons  $\text{interpret}$  calculable

### 1. Construction d'une table

	0	...	k	...
$Q_0$	$\text{interpret}(0,0)$	...	$\text{interpret}(0,k)$	...
:	:	:	:	:
$Q_k$	$\text{interpret}(k,1)$	...	$\text{interpret}(k,k)$	...
:	:	:	:	:

données programmes

programmes

entier

### 2. Sélection de la diagonale $\text{diag}(n) = \text{interpret}(n,n)$

# 7. Insuffisance des fonctions totales

## 3. Modification de diag

$\text{diag\_mod}(n) = \text{interpret}(n, n) + 1$

$\text{diag\_mod}$  calculable dans  $Q$ .

$\text{DiAG\_MOD}(n) \equiv \begin{array}{l} R = \text{interpret}(n, n) \\ R = R + 1 \\ \text{print}(R) \end{array}$

## 4. Contradiction $\text{diag\_mod} = Q_d$ (pour un certain $d$ )

quelle est la valeur de  $\text{diag\_mod}(d)$  ?

$\text{diag\_mod}(d) = \text{interpret}(d, d) + 1$

$\text{diag\_mod}(d) = \varphi'_d(d) = \text{interpret}(d, d)$

$\rightarrow$  déf. de  $\text{diag\_mod}$   
 $\rightarrow$   $\text{diag\_mod}$  calculé par  $Q_d$

## 5. Conclusion

$\text{interpret}$  n'est pas calculable dans  $Q$

# 7. Insuffisance des fonctions totales

## Interpréteur

- Un interpréteur est un programme utilisant des principes “simples” de programmation
  - lecture d’un fichier
  - structures de données
  - gestion de tables
  - opérations élémentaires
- Les techniques utilisées dans un interpréteur sont utilisées dans beaucoup de problèmes pratiques

# 7. Insuffisance des fonctions totales

## Analyse

- Si un langage de programmation (non trivial) ne permet *que le calcul de fonctions totales*, alors
  - l'interpréteur de ce langage n'est pas programmable dans ce langage
  - il existe des fonctions totales non programmables dans ce langage
  - ce langage est **restrictif**
- Si un langage de programmation (non trivial) permet de programmer son propre interpréteur, alors ce langage ne permet pas la programmation de la fonction halt
- Dans un langage de programmation (non trivial), il est impossible que l'interpréteur **et** la fonction halt puissent être programmées dans ce langage.
- Si on veut qu'un langage de programmation permette la programmation de toutes les fonctions totales calculables, alors ce langage doit également permettre la programmation de fonctions non totales

# 7. Insuffisance des fonctions totales

## Conséquences

- La fonction  $\text{tot}(n) = 1$  si  $\varphi_n$  est totale  
0 sinon  
n'est pas calculable
- L'ensemble  $\{n \mid \varphi_n \text{ est totale}\}$  n'est pas récursif

# 7. Insuffisance des fonctions totales

## Interpréteur

Tout formalisme complet utilisé pour étudier la calculabilité doit permettre de définir son propre interpréteur

### *Théorème :*

$$\exists z \forall n, x : \varphi_z(n, x) = \varphi_n(x) \quad \text{💬}$$

- $\varphi_z$  : *fonction universelle* calculable
- $P_z$  : programme universel (interpréteur)

Convention :

$\theta(n, x)$  dénote la ***fonction universelle***

# 8. Extension de fonctions partielles

Nouvelle insuffisance des fonctions totales calculables

## *Théorème :*

Il existe une fonction partielle calculable  $g$  telle qu'aucune fonction totale calculable n'est une extension de  $g$ .

## *Preuve :*

par diagonalisation

- Impossible donc de pouvoir toujours étendre un programme calculant une fonction partielle pour que ce programme se termine aussi (avec un résultat quelconque) pour les inputs où la fonction initiale n'est pas définie.

Exemple :  $\text{sqrt}(n) = \begin{cases} \text{arrondi}(\sqrt{n}, 5) & \text{si } n \geq 0 \\ \perp & \text{si } n < 0 \end{cases}$

$\text{SQRT}(n)$





Démo:

$nbStep(n, x) =$  nombre d'instructions du programme  $P_n(x)$   
avant l'arrêt de  $P_n(x)$  Si  $P_n(x) \neq \perp$   
•  $\perp$  si  $P_n(x) = \perp$

$nbStep^*(n, x)$  fct totale calculable

$HALT(n, x) \equiv$   $\left[ \begin{array}{l} k = NBSTEP^*(n, x) \\ \text{exécute } P_n(x) \text{ pour } k \text{ instructions} \\ \text{si exéc se termine Print(1)} \\ \text{sinon Print(0)} \end{array} \right.$

# 9. Théorème de Rice

Il est intéressant de pouvoir analyser un programme afin d'établir ses propriétés

Les propriétés des programmes peuvent-elles être déterminées par un algorithme ?

Raisonnement sur mesure

## Méthode de réduction à halt

Comment prouver qu'une fonction  $f(x)$  est non calculable ?

- Supposons  $f(x)$  calculable
- Montrons que sous cette hypothèse la fonction  $\text{halt}(n,x)$  est alors calculable
- Comme nous savons que  $\text{halt}(n,x)$  est non calculable, nous concluons que  $f(x)$  ne peut pas être calculable

# 9. Théorème de Rice

## Exemples

Les fonctions suivantes sont-elles calculables ?

$f(n) = 1$  si  $\varphi_n(x) = \perp$  pour tout  $x$  ( $P_n$  calcule la fonction partielle vide)  
0 sinon

$f(n) = 1$  si  $\varphi_n(x) = x$  pour tout  $x$  ( $P_n$  calcule la fonction identité)  
0 sinon

$f(n,m) = 1$  si  $\varphi_n = \varphi_m$  ( $P_n$  et  $P_m$  sont équivalents)  
0 sinon

$f(n) = 1$  si  $f_n(x) = 1 \quad \forall x$   
0 sinon



① Supposons  $f(x)$  calculable  $\exists$  pgm  $F$  calcule  $f(x)$

② Montre que  $\text{halt}(n, x)$  est calculable

$\text{HALT}(n, x) \equiv$  {

- constrains un programme  
 $P(z) \equiv \boxed{P_n(x); \text{print}(1)}$
- $d =$  numéro du pgm  $P(z)$
- if  $F(d) = 1$  car  $P_n(x)$  ne se termine pas  
then  $\text{print}(0)$
- else  $\text{print}(1)$   $P_n(x)$  se termine

③  $f(x)$  est non calculable

$$f(n) = \begin{cases} 1 & \text{if } p_n(x) = x \text{ for all } x \\ 0 & \text{otherwise} \end{cases}$$

$p_n(x) ; \text{ print}(x)$

Exercises:

- 1: supp  $f$  calculable
- 2:  $\text{halt}(n, x)$  is also calculable
- 3: Done  $f$  non calculable

$\text{halt non calc} \Rightarrow f \text{ non calculable}$

$f \text{ calculable} \Rightarrow \text{halt calculable (contrapositive)}$

def (n, x) =

- construire pgm  
 $P(z) = P_n(x); \text{print}(Z)$
- d num du pgm  $P(z)$
- si  $F(0) = 1$   
    alors  $\text{print}(1)$   
    sinon  $\text{print}(0)$

F: pgm qui calcule f

Si  $P_n(x)$  se termine  $\rightarrow P(z)$  calcule f identitè  
 $\rightarrow f_d(z) = z \quad \forall z$   
 $\rightarrow f(d) = 1$

Si  $P_n(x)$  ne se termine pas  
 $\rightarrow P(z)$  calcule f partielle vide  
 $\rightarrow f_0(z) = \perp \quad \forall z$   
 $\rightarrow f(d) = 0$

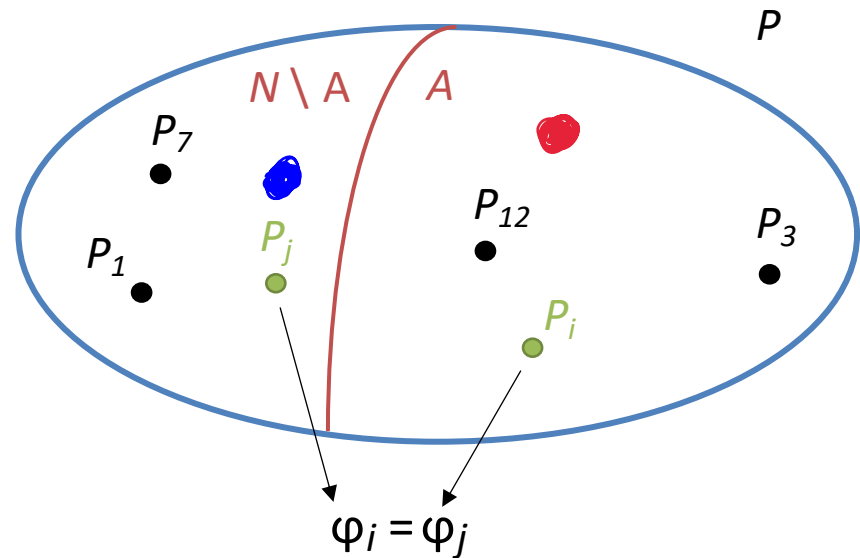
# 9. Théorème de Rice

## Théorème de Rice (1953)

Soit  $A \subseteq N$

Si  $A$  récursif **et**  $A \neq \emptyset$  **et**  $A \neq N$

Alors  $\exists i \in A$  et  $j \in N \setminus A$   
tel que  $\varphi_i = \varphi_j$





Autre formulation

Si  $\forall i \in A$  et  $\forall j \in N \setminus A : \varphi_i \neq \varphi_j$

Alors  $A$  non récursif **ou**  $A = \emptyset$  **ou**  $A = N$

# 9. Théorème de Rice

## Analyse

-  Si une propriété de programmes, vérifiée par certains programmes mais pas tous, est décidable, alors il existe deux programmes équivalents (calculant la même fonction) dont un vérifie la propriété et l'autre pas
-  Si une propriété de la fonction calculée par un programme est vérifiée par certains programmes, mais pas par tous, alors cette propriété ne peut être décidée par un algorithme
- S'il existe un algorithme permettant de déterminer si un programme quelconque calcule une fonction ayant cette propriété, alors toutes les fonctions calculables ont cette propriété ou aucune fonction calculable n'a cette propriété



# 9. Théorème de Rice

## Exemples



$A_1 = \{ i \mid \varphi_i \text{ est totale} \}$  ( $P_i$  s'arrête toujours)

$A_2 = \{ i \mid \varphi_i = f \}$  avec  $f$  fixé ( $P_i$  calcule la fonction  $f$ )

$A_3 = \{ i \mid a \in \text{dom}(\varphi_i) \}$  avec  $a$  fixé ( $P_i(a)$  se termine)


$A_4 = \{ i \mid \varphi_i(x) = \perp \text{ pour tout } x \}$  ( $P_i$  calcule la fonction partielle vide)

$A_5 = \{ i \mid a \in \text{image}(\varphi_i) \}$  avec  $a$  fixé ( $P_i$  donne au moins une fois le résultat  $a$ )

$A_6 = \{ i \mid \text{image}(\varphi_i) = N \}$  ( $P_i$  donne tous les résultats possible)


$A_7 = \{ i \mid \varphi_i \text{ est une fonction injective} \}$  ( $P_i$  calcule une fonction injective)

- $A_1, \dots, A_7$  : ensembles non récursifs
- $\bar{A}_1, \dots, \bar{A}_7$  : ensembles non récursifs

•  $A_1 \neq \emptyset \rightarrow p_h(x) \equiv \text{print}(1)$  

$\Rightarrow k \in A_1 \Rightarrow A_1 \neq \emptyset$

•  $A_1 \neq N$

$p_d(x) \equiv \text{while true};$   
 $0! \notin A_1$  

$A_1 \neq N$

•  $\forall i \in A_1, \forall j \in \overline{A_1}$

$f_i \neq f_j$

$f_i$  totale  
(défini sur  $A_1$ )

$\neq$

$f_j$  non totale

• Rice

$A_1$  n'est pas récursif

# 9. Théorème de Rice

## Analyse

- Aucune question relative aux programmes, vus *sous l'angle de la fonction* qu'ils calculent, ne peut être décidée par l'application d'un algorithme
- Les propriétés intéressantes d'un programme concernent la fonction qu'il calcule, non pas la forme (syntaxe) du programme
- La plupart des problèmes intéressants au sujet des programmes sont non calculables



# 9. Théorème de Rice

## Preuve du théorème de Rice

Si  $A \neq \emptyset$  et  $A \neq \mathbb{N}$

$\forall i \in A, \forall j \in \bar{A} : \varphi_i \neq \varphi_j$

Alors  $A$  est non récursif



étape:

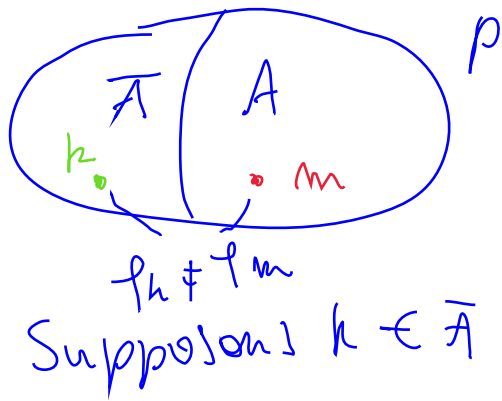
Réduct<sup>o</sup> à halt

x Supposons  $A$  récursif

x halt est calculable

x  $A$  est non récursif





$P_n(x) \equiv \text{while true};$

On sait que  $A \neq \emptyset, \exists m \in A$

Choisissons  $m$  quelconque

• Réduct° à HALT

$HALT(m, x) \equiv$  {

- construire un programme

$P(z) \equiv P_m(x); P_m(z)$

- $d = \text{mem du prog } P(z)$
- if  $d \in A$ 
  - then print(1)
  - else print(0)

}



Si  $P_n(x)$  se termine,  $f_d = P_n$

„ „ ne se termine pas,  $f_d = P_h$

$\Rightarrow$  on a terminé la réduction à halt



# 10. Théorème de la paramétrisation

Problème de l'arrêt, théorème Hoare-Allison, théorème de Rice : résultats négatifs sur la calculabilité !

Existe-t-il des résultats positifs ?

## Transformation de programmes

Une fonction  $f : N \rightarrow N$  peut être vue comme une fonction de  $P \rightarrow P$  

$$f(a) = b \qquad P_a \rightarrow P_b$$

A partir d'un programme numéro  $a$ ,  
la fonction donne un programme numéro  $b$

Si  $P_k$  calcule la fonction  $f$

$P_k$  peut être vu comme un *transformateur de programmes*

# 10. Théorème de la paramétrisation

## Théorème S-m-n (Kleene, 1952) [version S-1-1]

il existe une fonction totale calculable  $S_1^1 : N^2 \rightarrow N$

telle que pour tout  $k$  :

$$\varphi_k^{(2)}(x_1, x_2) = \varphi_{S_1^1(k, x_2)}(x_1)$$

### Analyse

Il existe un *transformateur de programme*, (appelé  $S_1^1$ ), qui recevant comme **données** :

- un programme  $P_k$  à 2 arguments
- 1 valeur  $v_2$

fournit comme **résultat** :

- un programme  $P(x_1)$  qui calcule la *même* fonction que  $P_k(x_1, v_2)$



# 10. Théorème de la paramétrisation

## Théorème S-m-n (Kleene, 1952)

Pour tout  $m, n \geq 0$ , il existe une fonction totale calculable  $S_n^m : N^{m+1} \rightarrow N$  telle que pour tout  $k$  :

$$\varphi_k^{(n+m)}(x_1, \dots, x_n, x_{n+1}, \dots, x_{n+m}) = \varphi_{S_n^m(k, x_{n+1}, \dots, x_{n+m})}^{(n)}(x_1, \dots, x_n)$$

## Analyse

Etant donné  $m, n \geq 0$ , il existe un *transformateur de programme*, (appelé  $S_n^m$ ), qui recevant comme **données** :

- un programme  $P_k$  à  $m+n$  arguments
- $m$  valeurs  $v_1, \dots, v_m$

fournit comme **résultat** :

- un programme  $P$  à  $n$  arguments  
tel que  $P(x_1, \dots, x_n)$  calcule la *même* fonction que  $P_k(x_1, \dots, x_n, v_1, \dots, v_m)$

# 10. Théorème de la paramétrisation

Le transformateur de programme  $S^m_n$  particularise un programme à  $m+n$  arguments en rendant constant les  $m$  derniers arguments du programme

- $S^m_n$  est calculable en toute généralité (en fonction de  $m$  et  $n$ ), *indépendamment des programmes à transformer*
- Le transformateur  $S^m_n$  est applicable à tous les programmes à  $m+n$  arguments dont on veut particulariser, d'une quelconque manière, les  $m$  derniers arguments

S-m-n est un théorème dont l'expression précise est difficile, mais dont la signification est très intuitive

# 10. Théorème de la paramétrisation

Preuve du théorème S-m-n

$S'$

Evident!

$S'_1 \times \text{Pgm } P_k(x_1, \dots, x_2)$

$\times$  valeur  $V_2$

comment spécialiser  $P_k$ ?

→ par exemple, insérer l'instruction  $x_2 = V_2$  au début du pgm

# 10. Théorème de la paramétrisation

**S-1-1**  $\exists \dots \forall$

Il existe une fonction totale calculable  $S^1_1 : N^2 \rightarrow N$   
telle que pour tout  $k$  :  $\varphi_k(x_1, x_2) = \varphi_{S^1_1(k, x_2)}(x_1)$

**S: version affaiblie de S-1-1**

Pour tout programme  $k$   $\forall \dots \exists$

il existe une fonction totale calculable  $S : N \rightarrow N$

telle que  $\varphi_k(x_1, x_2) = \varphi_{S(k)}(x_1)$

**Analyse**

$S(x_2)$

- La fonction  $S$  est spécifique à la spécialisation du programme  $P_k$

# 11. Théorème du point fixe

Résultat positif de la calculabilité

Théorème dont l'expression précise est facile, mais dont la signification est peu intuitive

## Théorème du point fixe (Kleene, 1952)

Soient

- $n \geq 0$
- $f$  : fonction totale calculable

il existe  $k$  tel que  $\varphi_k^{(n)} = \varphi_{f(k)}^{(n)}$

## Analyse

Pour tout transformateur (i.e. fonction totale calculable) de programme  $T$ , il existe deux programmes  $P_k$  et  $P_j$  tels que :

- $P_j$  est la transformation de  $P_k$  via  $T$
- $P_k$  et  $P_j$  calculent la même fonction (qui n'est pas nécessairement totale !)

# 11. Théorème du point fixe

## Démonstration du théorème

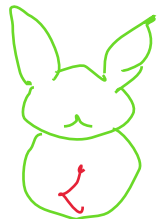
Soit  $f$  totale calculable

$\exists$  programme  $k : \uparrow_k = \uparrow_{f(k)}$



$$h(u, x) = \begin{cases} \uparrow_{u(u)}^{(x)} & \text{si } \uparrow_u(u) \neq \perp \\ \perp & \text{sinon} \end{cases}$$

$h$  est calculable



on applique la propriété 5 sur  $h(u, u)$

$\left[ h(u, u) = \uparrow_{y(u)}^{(u)} \text{ [Prop 5]} \rightarrow y \text{ existe et totale calculable} \right]$



Sei  $k(x) = f(g(x))$   totale calculable

$$\exists k' : k(x) = f_{k'}(x) = f(g(x))$$

$\forall x : k(k', x) =$    $f_{g(k')}(x)$  Lemma 2

$$\forall x \quad k(k', x) = f_{\underbrace{f_{k'}(k')}}(x) \quad \text{Lemma 1}$$

$$= f_{f(g(k'))}(x) \quad \text{Lemma 3}$$

$$\text{Posons } k = g(k')$$

$$\rightarrow \forall x : f_k(x) = f_{f(k)}(x)$$

# 11. Théorème du point fixe

## Théorème de Rice via théorème du point fixe

Si  $A \neq \emptyset$  et  $A \neq \mathbb{N}$

$$\boxed{\forall i \in A, \forall j \in \bar{A} : \varphi_i \neq \varphi_j}$$



alors A non récursif

$$A \neq \emptyset : m \in A$$

$$A \neq \mathbb{N} : m \in \bar{A}$$

$$\text{Définissons } f(i) = \begin{cases} m & \text{si } i \in A \\ n & \text{si } i \in \bar{A} \end{cases}$$

$f$  est totale calculable si A est récursif



Point fixe :  $\exists \text{ pgm } k : \varphi_k = \varphi_{f(k)}$

$k \in A$  ?

Si  $k \in A$  alors  $f(k) = m$  (défini<sup>o</sup> de  $f$ )

contradict<sup>o</sup> alors  $\varphi_k = \varphi_m$

alors  $k \in \bar{A}$  car  $m \in \bar{A}$  et \*

Si  $k \in \bar{A}$  alors  $f(k) = n$  (défini<sup>o</sup> de  $f$ )

contradict<sup>o</sup> alors  $\varphi_k = \varphi_n$

alors  $k \in A$  car  $n \in A$  et \*

Donc  $A$  non récur<sup>sif</sup> :

# 11. Théorème du point fixe

## Applications

- Il existe un programme  $P_n$  tel que  $P_n$  ne s'arrête que pour la donnée  $n$
- Il existe un programme  $P_n$  tel que  $P_n$  donne toujours  $n$  comme résultat
- $K = \{ n \mid \varphi_n(n) \neq \perp \}$  n'est pas récursif
- Transformateur de programmes qui remplace les  $+$  par des  $-$

Soient  $f_n(x) = 1 \quad \forall x$

$f_m(x) = x \quad \forall x$

définissons  $f(x) = \begin{cases} 1 & \text{si } x \in K \\ m & \text{si } x \notin K \end{cases}$

$f$  est totale calculable (si  $K$  récursif)

Point fixe :  $\exists \text{ pgm } h : f_h = f_{f(h)}$

$k \in K$  ?

si  $k \in K$  alors  $f(k) = 1$  (définit<sup>n</sup> de  $f$ )

$\Rightarrow f_h(k) = f_m(k)$  (point fixe)

$\swarrow \neq 1$

$\rightarrow 1$

$\Rightarrow \text{contradiction}$

si  $k \notin K$  alors  $f(k) = m$

$\Rightarrow f_h(k) = f_m(k)$

$\swarrow = 1$

$\rightarrow k \Rightarrow \text{contradiction}$

# 11. Théorème du point fixe

## Application (récréative)

Il existe un programme Java  $P$  tel que l'exécution de  $P$  donne toujours la représentation de  $P$  comme résultat

Représentation du programme par son texte

Il existe un programme Java  $P$  tel que

l'exécution de  $P$  donne toujours comme résultat le texte du programme  $P$

## Spécification du programme selfrep

Precondition : /

Postcondition : impression à la sortie standard du texte du programme selfrep

# 11. Théorème du point fixe

## Analyse

- Théorème du point fixe: résultat central de la calculabilité
- Utilise uniquement la propriété S des langage de programmation
- Implique théorème de Rice
- Implique la non récursivité de K
- Implique la non calculabilité de fonction HALT

# 12. Autres problèmes non calculables

## Problème de correspondance de Post

Soient deux listes  $U$  et  $V$  de  $k$  mots non vides sur un alphabet  $\Sigma$

- $U = \{ u_1, u_2, \dots, u_k \}$
- $V = \{ v_1, v_2, \dots, v_k \}$

Existe-t-il une suite d'entiers  $i_1, i_2, \dots, i_n$  telle que les mots

$$u_{i_1} u_{i_2} \dots u_{i_n}$$
$$v_{i_1} v_{i_2} \dots v_{i_n}$$

soient **identiques**

## 12. Autres problèmes non calculables

### Exemple

- $\Sigma = \{ a, b \}$
- $U = \{ u_1 = b, u_2 = babb, u_3 = ba \}$
- $V = \{ v_1 = bbb, v_2 = ba, v_3 = a \}$

Il existe une suite d'entiers : 2 1 1 3 telle que les mots

- $u_2 u_1 u_1 u_3 = babb b b ba$
- $v_2 v_1 v_1 v_3 = ba bbb bbb a$

sont identiques

Il **n'**existe **pas** d'algorithme qui, recevant  $U$  et  $V$  comme données, permette de décider de l'existence d'une telle suite

Problème **non calculable (indécidable)**

Indécidabilité du problème de correspondance de Post très utilisé pour démontrer l'indécidabilité d'autres problèmes

# 12. Autres problèmes non calculables

## Equations diophantines

Une équation diophantine est une équation de la forme

$$D(x_1, \dots, x_n) = 0$$

où  $D$  est un polynôme à coefficients entiers

*Existe-t-il un algorithme qui, recevant une équation diophantine, détermine si cette équation possède une solution entière ?*

(10ème problème de Hilbert, 1900)

Ce problème est **non calculable (indécidable)**

Si  $D$  est de degré inférieur ou égale à 4, le problème est aussi indécidable



# 12. Autres problèmes non calculables

## Equations diophantines

Une propriété  $P(a)$  est diophantine ssi il existe  $D$

$$P(a) \Leftrightarrow \exists x_1, \dots, x_n [ D(a, x_1, \dots, x_n) = 0 ]$$

Un ensemble  $E$  est diophantine ssi il existe  $D$

$$a \in E \Leftrightarrow \exists x_1, \dots, x_n [ D(a, x_1, \dots, x_n) = 0 ]$$

*La classe des ensembles diophantines est égale à la classe des ensembles récursivement énumérables*

# 13. Codage et représentation

Jusqu'à présent, uniquement

- fonctions de  $N^n$  dans  $N$
- représentation décimale des nombres

Notion informelle d'algorithme utilise d'autres types de données et de résultats

Comment inclure cet aspect dans la théorie de la calculabilité ?

- utiliser une fonction de codage / représentation
- (ou inclure dès le départ d'autres types de données)

# 13. Codage et représentation

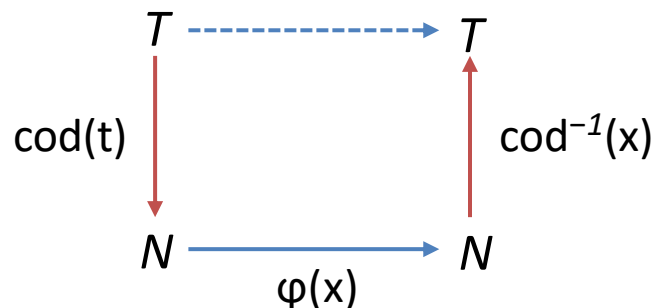
## Codage

Définition d'une bijection entre le nouveau type et les entiers

Chaque objet peut être identifié par un entier (distinct)

Propriété d'un codage  $\text{cod} : T \rightarrow N$

1. la fonction  $\text{cod}$  est bijective
2. la fonction  $\text{cod}$  est calculable
3. la fonction  $\text{cod}^{-1}$  est calculable



La numérotation des programmes est un codage particulier

# 14. Nombres calculables

Tout nombre réel est défini  
comme la limite d'une suite (convergente) de nombres rationnels ( $Q$ )

Pour tout  $x \in R$ ,  
il existe une fonction totale  $s : N \rightarrow Q$   
telle que

$$\lim_{n \rightarrow \infty} |x - s(n)| = 0$$

## Définition

Un nombre réel  $x$  est *calculable* si  
il existe une fonction totale *calculable*  $s : N \rightarrow Q$   
telle que

$$\text{pour tout } n : |x - s(n)| \leq 2^{-n}$$

# 14. Nombres calculables

## Analyse

- Un nombre réel est calculable s'il existe un programme qui peut approximer ce nombre réel aussi précisément que l'on veut
- Les nombres réels algébriques sont calculables
- Les nombres réels non algébriques  $\pi$  et  $e$  sont calculables

## Propriétés

- L'ensemble des nombres réels calculables est énumérable
- Il existe des nombres réels non calculables
- Il existe des nombres réels non calculables qui peuvent être définis de manière finie
- Le réel

$$x = \sum_{0 \leq n \leq \infty} \chi_k(n) 3^{-n}$$

n'est pas calculable