Rudiments de Calculabilité et de Complexité

Paul Gastin

LSV, ENS Cachan, INRIA, CNRS, FRANCE

Informatique et sciences du numérique, 2 juin 2010

http://www.lsv.ens-cachan.fr/~gastin/

L'informatique : une science ?

L'informatique : une science ?

- Utilisateur

- Conducteur

L'informatique : une science ?

- Utilisateur
- Développeur de logiciels

- Conducteur
- Constructeur de voitures

L'informatique : une science ?

- Utilisateur
- Développeur de logiciels
- Concepteurs : algorithmes, langages de programmation, ...

- Conducteur
- Constructeur de voitures
- Concepteurs : ABS, direction assistée, pneus, ...

L'informatique : une science ?

- Utilisateur
- Développeur de logiciels
- Concepteurs : algorithmes, langages de programmation, ...
- Fondements: Calculabilité et Complexité

- Conducteur
- Constructeur de voitures
- Concepteurs : ABS, direction assistée, pneus, ...
- Fondements: l'énergie, les matériaux, ...

Objectifs

Calculabilité : Cerner les limites des machines . . .

- Peut-on résoudre tous les problèmes avec un ordinateur ?
- Peut-on calculer toutes les fonctions avec un ordinateur ?
- Quels traitements des informations ?
- Pourra-t-on inventer un jour une machine qui pourra résoudre plus de problèmes que nos ordinateurs ?
- . . .

Objectifs

Calculabilité : Cerner les limites des machines . . .

- Peut-on résoudre tous les problèmes avec un ordinateur ?
- Peut-on calculer toutes les fonctions avec un ordinateur ?
- Quels traitements des informations ?
- Pourra-t-on inventer un jour une machine qui pourra résoudre plus de problèmes que nos ordinateurs ?
- . . .

Complexité : . . . et des temps de calculs

- Combien de temps faut-il pour résoudre un problème ?
- Trouvera-t-on un jour des algorithmes radicalement plus efficaces ?
- Est-ce que facile à vérifier équivaut à facile à résoudre ?
- . . .

Plan

Plan

Sudoku

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

Sudoku

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

Peut-on résoudre automatiquement les problèmes de Sudoku ?

Codage : ? 2 ? 6 ? 8 ? ? ? 3 7 ? ...

Sudoku

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

Peut-on résoudre automatiquement les problèmes de Sudoku ?

Codage : ? 2 ? 6 ? 8 ? ? ? 3 7 ? ...

Sudoku

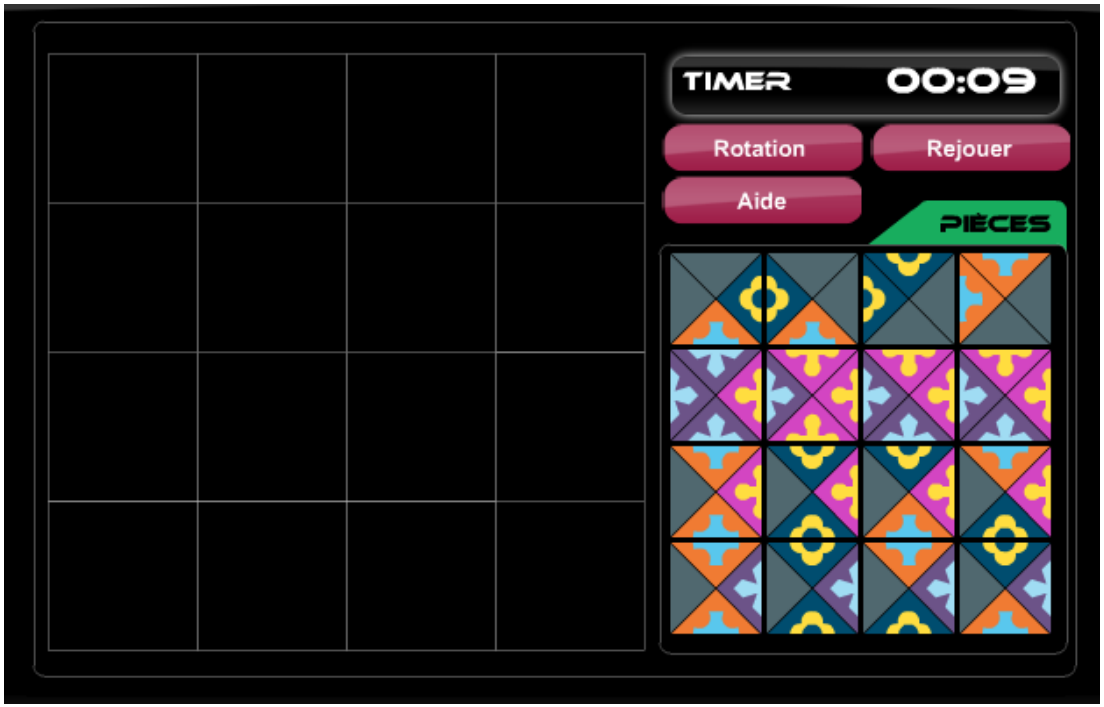
9	2	5	6	3	8	7	1	4
3	7	4	1	2	5	6	9	8
8	1	6	4	7	9	5	3	2
4	9	1	3	5	7	2	8	6
5	3	7	8	6	2	1	4	9
2	6	8	9	4	1	3	7	5
7	8	3	2	9	6	4	5	1
1	5	2	7	8	4	9	6	3
6	4	9	5	1	3	8	2	7

Peut-on résoudre automatiquement les problèmes de Sudoku ?

Codage : ? 2 ? 6 ? 8 ? ? ? 3 7 ? ...

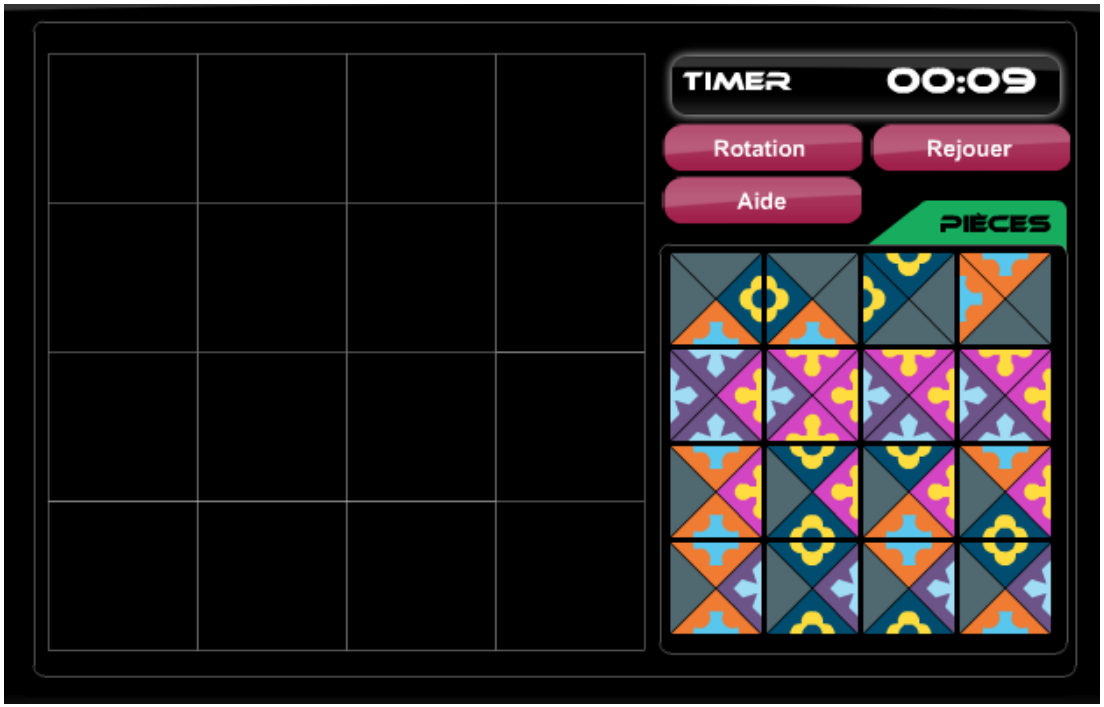
Ethernity $16 = 4 \times 4$

http://fr.eternityii.com/



Ethernity $16 = 4 \times 4$

http://fr.eternityii.com/

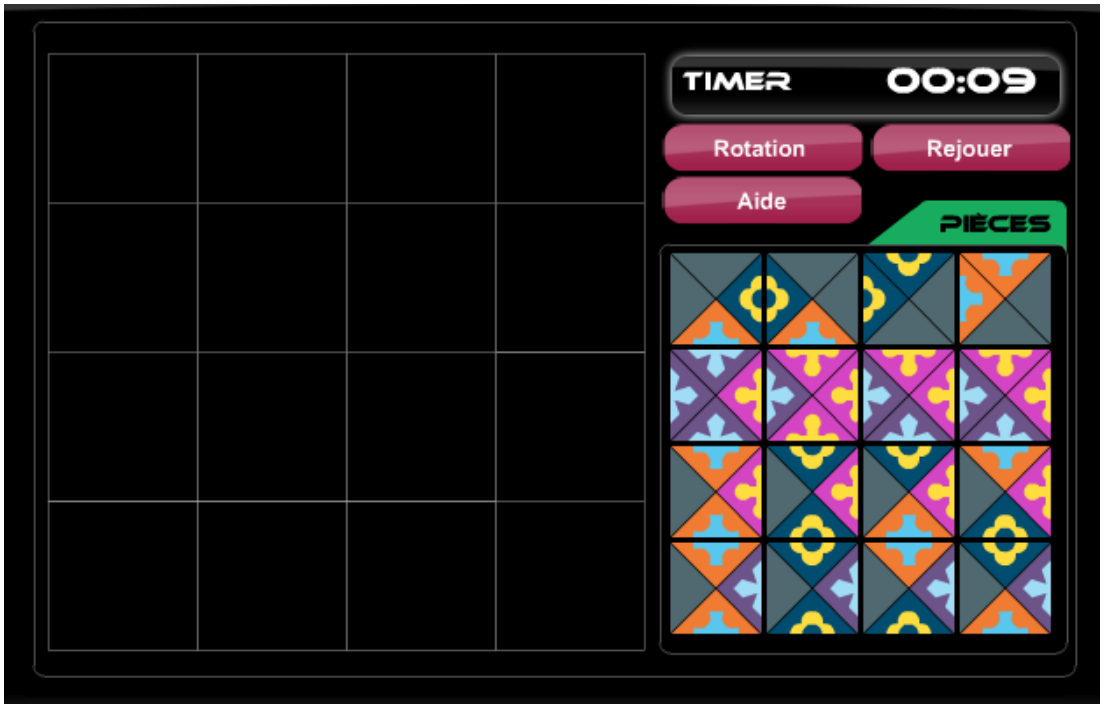


Peut-on résoudre automatiquement ce puzzle ?

Codage : GOBG BOGG BGGB OGGO
VVRV VRRR ...

Ethernity $16 = 4 \times 4$

http://fr.eternityii.com/

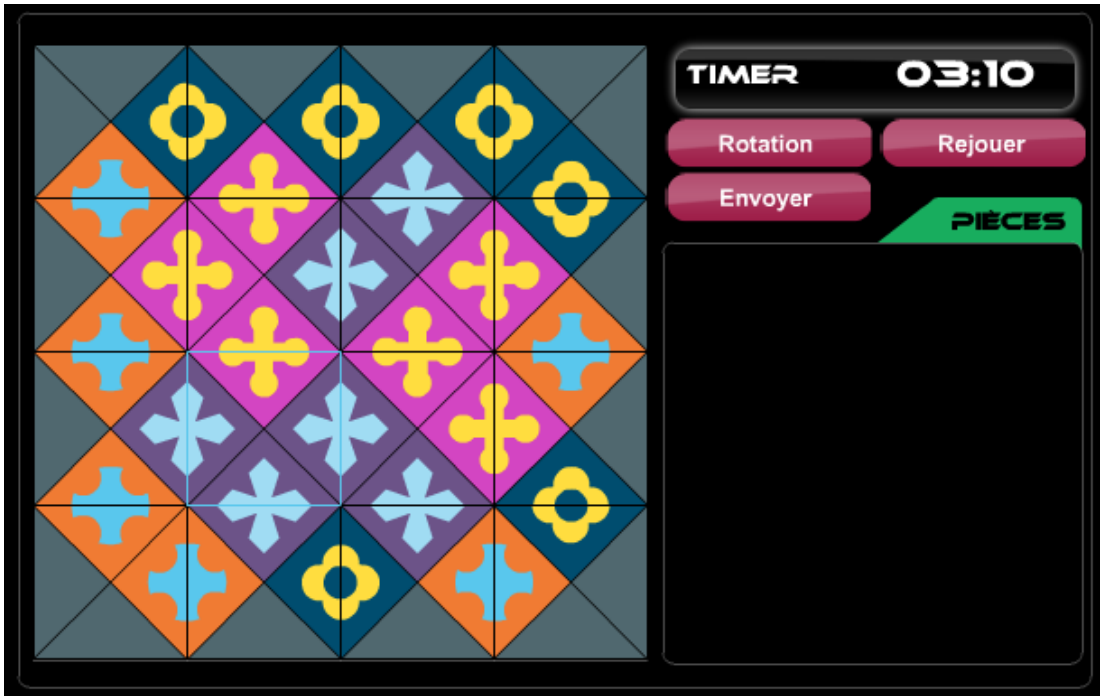


Peut-on résoudre automatiquement ce puzzle ?

Codage : GOBG BOGG BGGB OGGO
VVRV VRRR ...

Ethernity $16 = 4 \times 4$

http://fr.eternityii.com/



Peut-on résoudre automatiquement ce puzzle ?

Codage : GOBG BOGG BGGB OGGO
VVRV VRRR ...

Ethernity II

http://fr.eternityii.com/

Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Ethernity II

http://fr.eternityii.com/

Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Et si on corsait un peu le problème ?

- $36 = 6 \times 6$

Ethernity II

http://fr.eternityii.com/

Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Et si on corsait un peu le problème ?

- $36 = 6 \times 6$

Ethernity II

http://fr.eternityii.com/

Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Et si on corsait un peu le problème ?

- $36 = 6 \times 6$
- $72 = 6 \times 12$

Ethernity II

http://fr.eternityii.com/

Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Et si on corsait un peu le problème ?

- $36 = 6 \times 6$
- $72 = 6 \times 12$

Ethernity II

http://fr.eternityii.com/

Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Et si on corsait un peu le problème ?

- $36 = 6 \times 6$
- $72 = 6 \times 12$
- $256 = 16 \times 16$

Ethernity II

http://fr.eternityii.com/

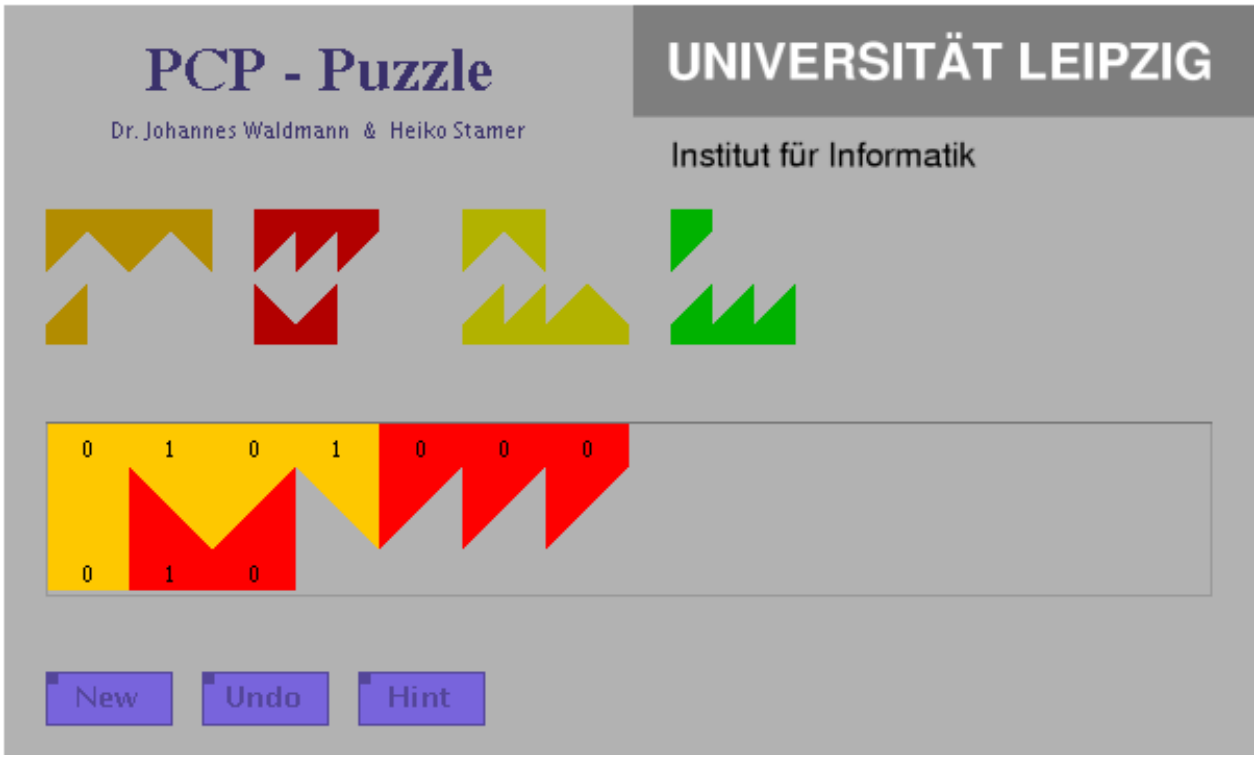
Peut-on résoudre automatiquement tous les puzzles de type *Ethernity* ?

Et si on corsait un peu le problème ?

- $36 = 6 \times 6$
- $72 = 6 \times 12$
- $256 = 16 \times 16$
  Un puzzle à 2 millions de dollars

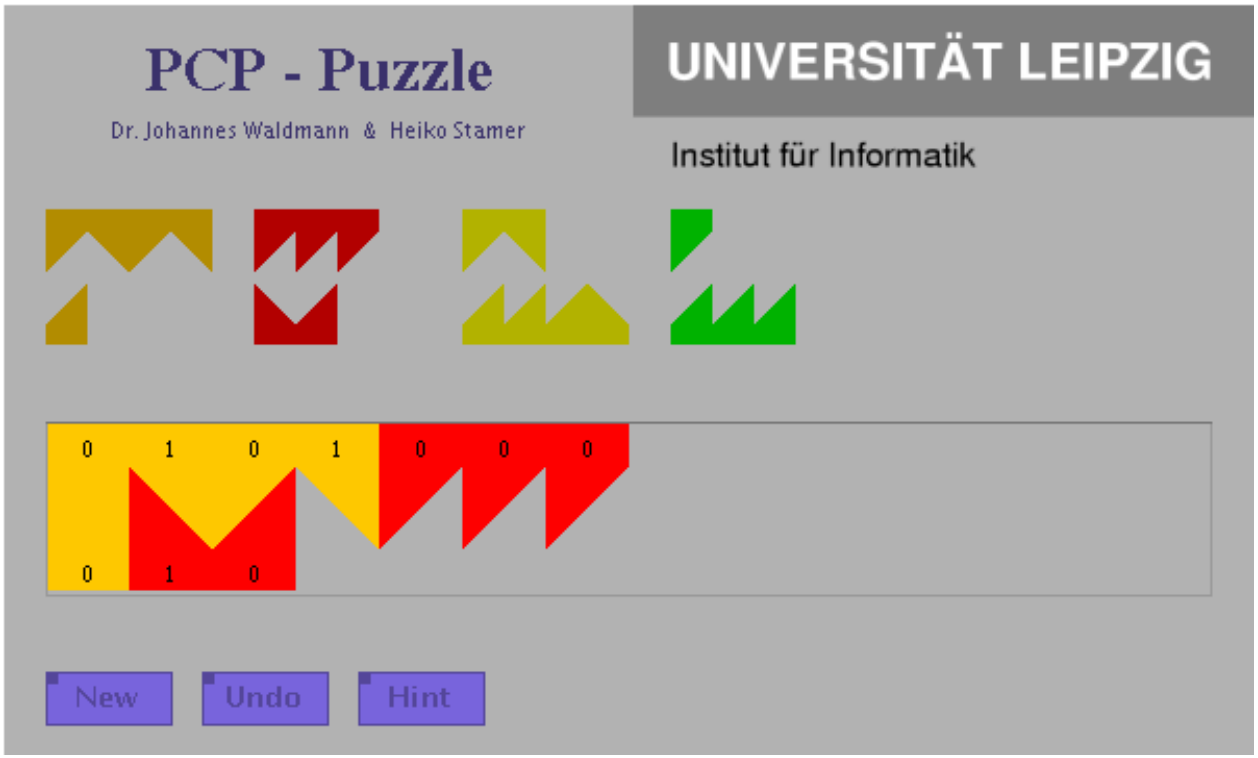
Problème de Correspondance de Post

http://www.theory.informatik.uni-kassel.de/~stamer/pcp/pcpcontest_en.html



Problème de Correspondance de Post

http://www.theory.informatik.uni-kassel.de/~stamer/pcp/pcpcontest_en.html

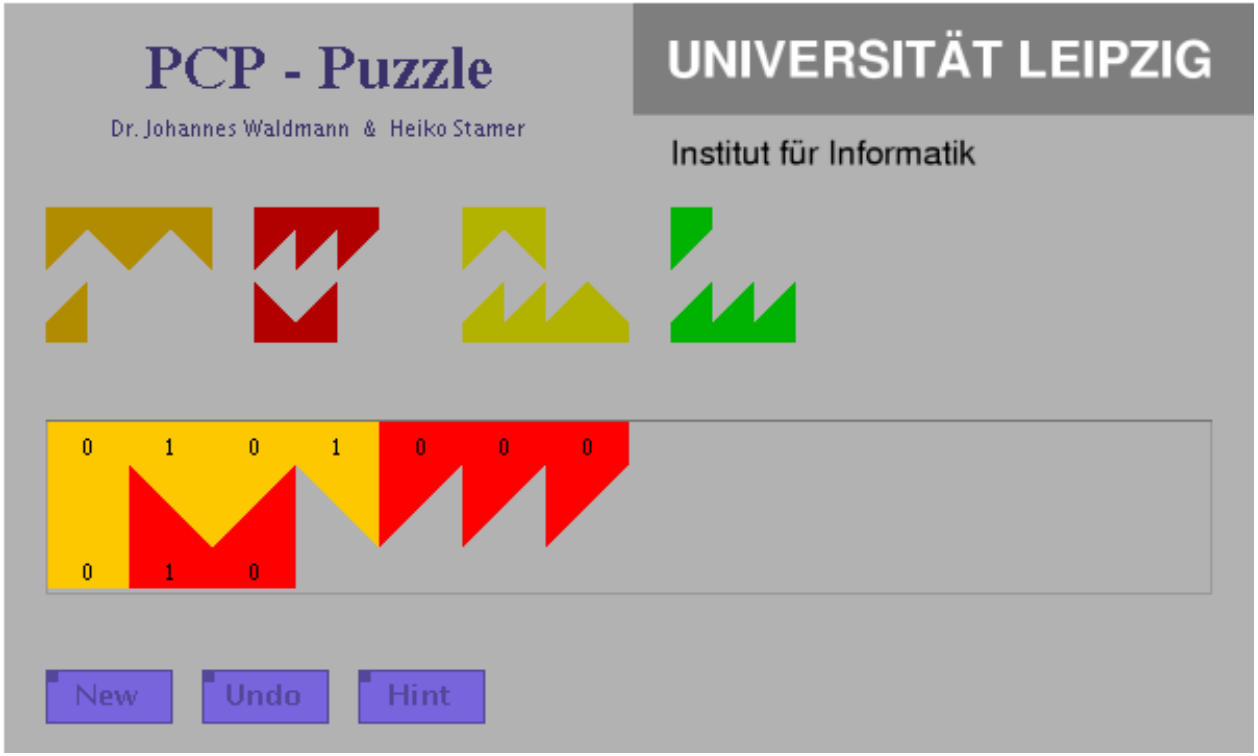


Peut-on résoudre le PCP automatiquement ?

Codage : (0101,0) (000,10) (01,0001) (0,000)

Problème de Correspondance de Post

http://www.theory.informatik.uni-kassel.de/~stamer/pcp/pcpcontest_en.html



Peut-on résoudre le PCP automatiquement ?

Codage : (0101,0) (000,10) (01,0001) (0,000)

Vérifier des programmes

Un problème des plus importants !

Codage : le texte source du programme.

Vérifier des programmes

Un problème des plus importants !

Codage : le texte source du programme.

Commençons simple :

- Mon programme va-t-il s'arrêter ?
  Sur toutes les données ?
  Sur au moins une donnée ?
  Sur cette donnée particulière ?

Vérifier des programmes

Un problème des plus importants !

Codage : le texte source du programme.

Commençons simple :

- Mon programme va-t-il s'arrêter ?
  Sur toutes les données ?
  Sur au moins une donnée ?
  Sur cette donnée particulière ?

Vérifier des programmes

Un problème des plus importants !

Codage : le texte source du programme.

Commençons simple :

- Mon programme va-t-il s'arrêter ?
  Sur toutes les données ?
  Sur au moins une donnée ?
  Sur cette donnée particulière ?
- Ma centrale risque-t-elle d'exploser ?

```
...
// Y a-t-il un risque de division par 0
z <- x / y
...
```

Vérifier des programmes

Un problème des plus importants !

Codage : le texte source du programme.

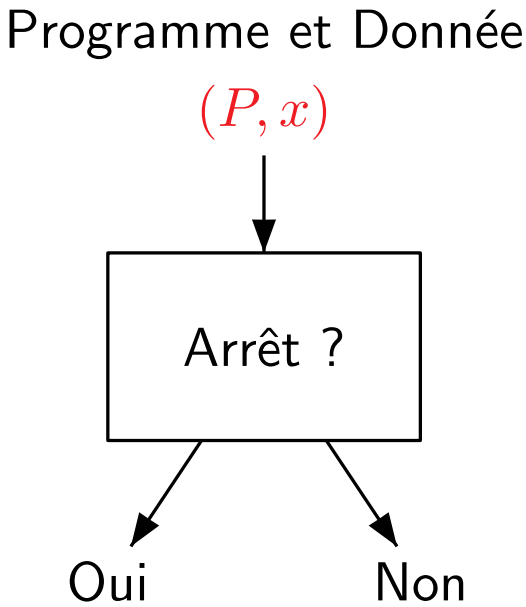
Commençons simple :

- Mon programme va-t-il s'arrêter ?
  Sur toutes les données ?
  Sur au moins une donnée ?
  Sur cette donnée particulière ?
- Ma centrale risque-t-elle d'exploser ?

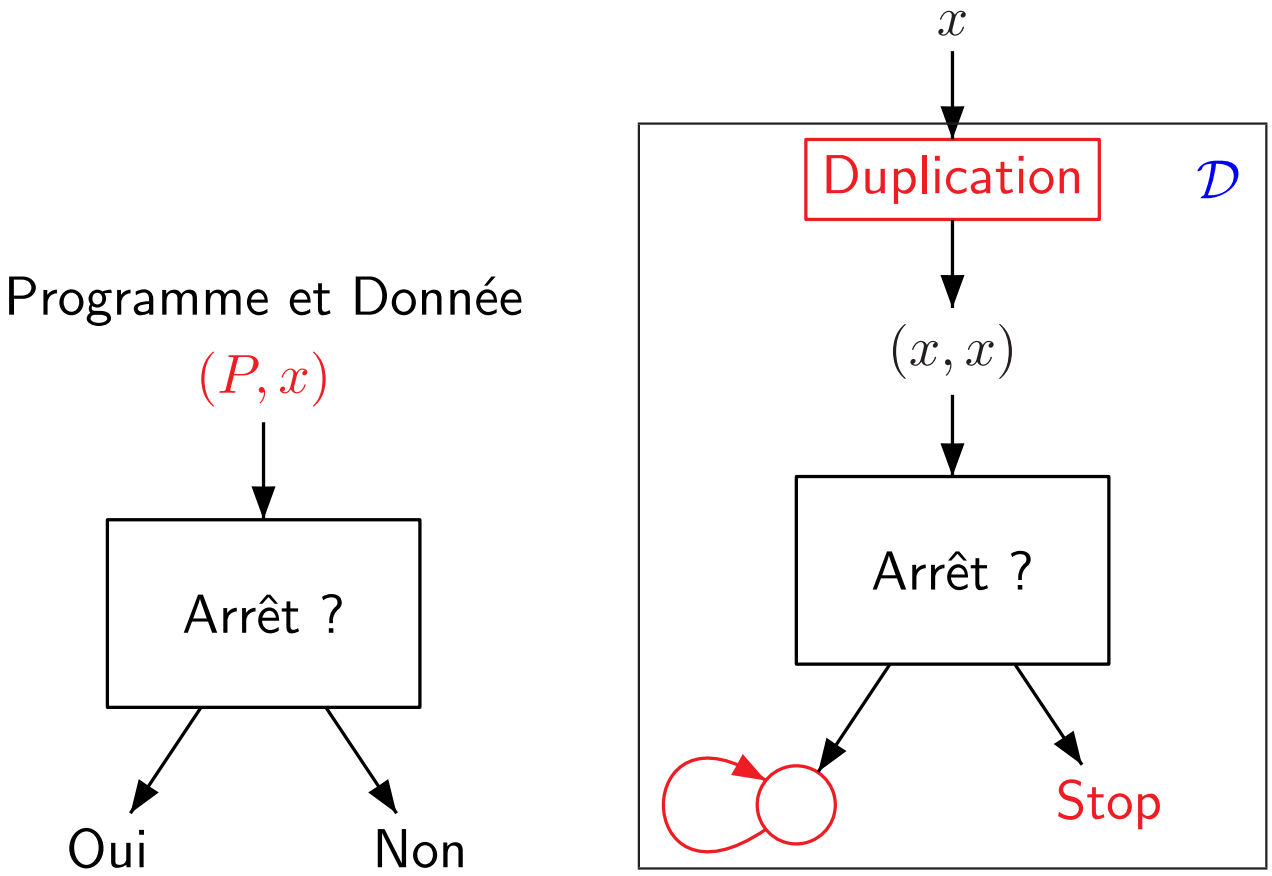
```
...
// Y a-t-il un risque de division par 0
z <- x / y
...
```

Plan

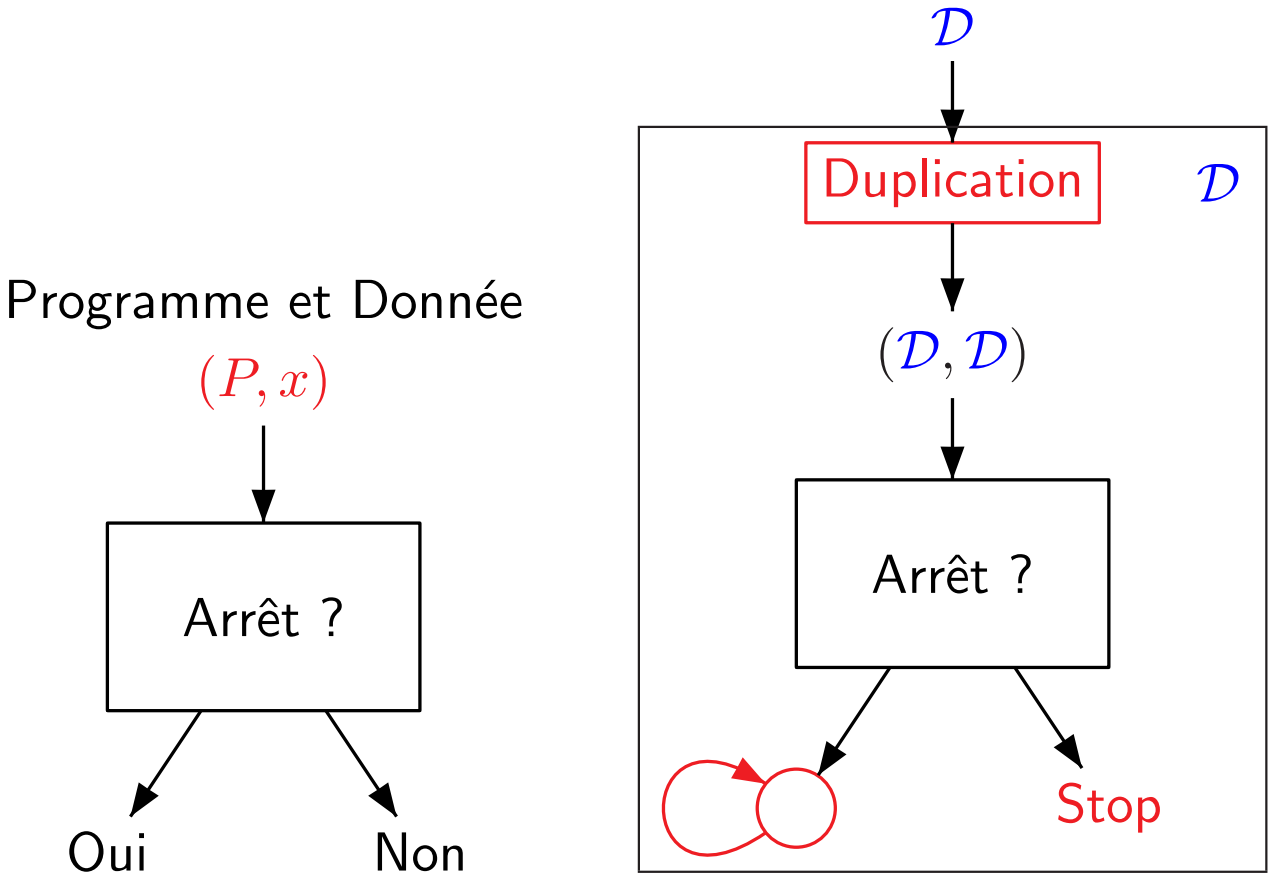
Diagonalisation



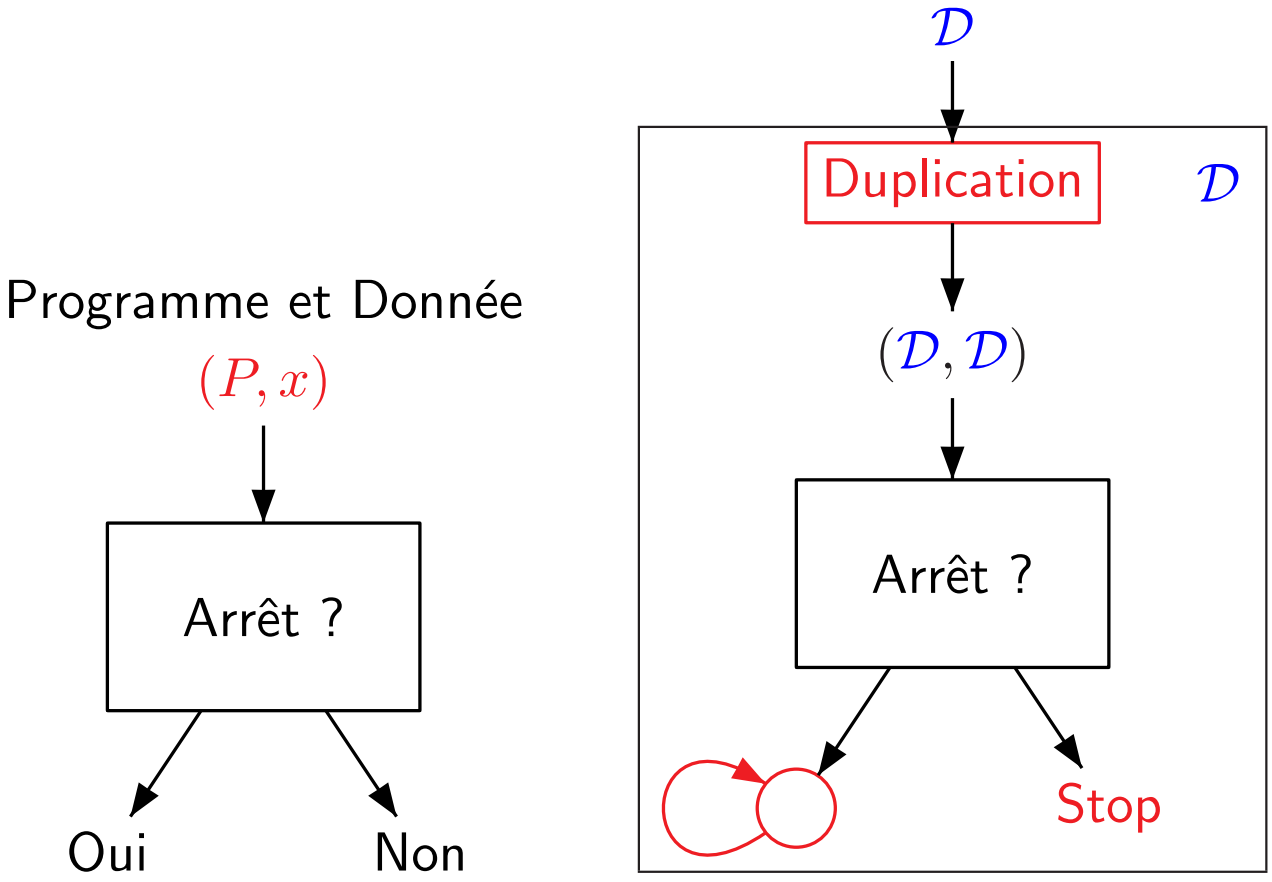
Diagonalisation



Diagonalisation

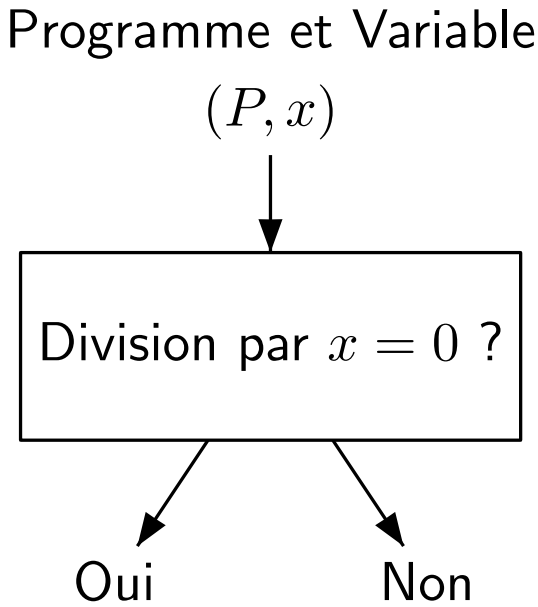


Diagonalisation

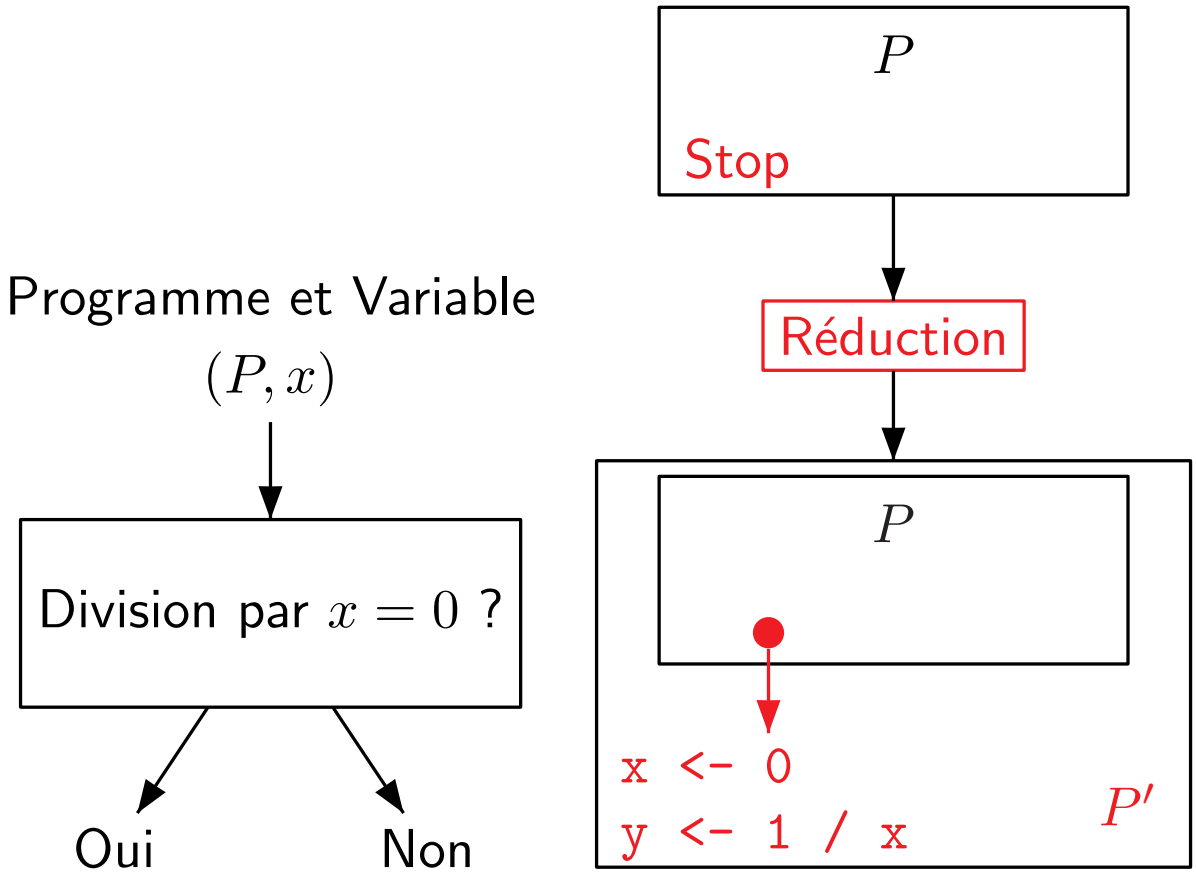


Il n'y a pas de programme pour tester l'arrêt !

Réduction

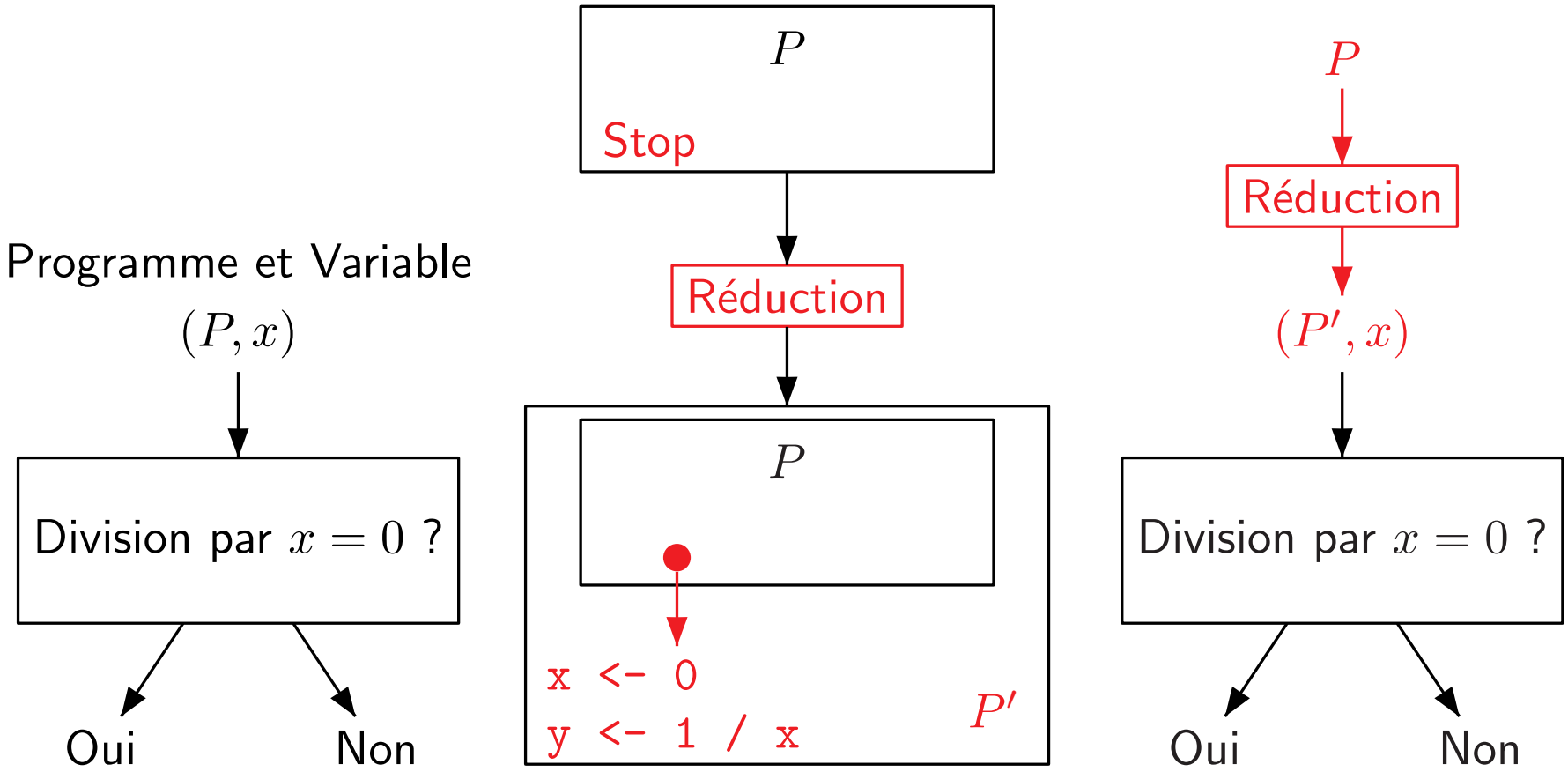


Réduction



P s'arrête si et seulement si P' divise par $x = 0$

Réduction



P s'arrête si et seulement si P' divise par $x = 0$

Plan

Hilbert et les formalistes

David Hilbert, 1862 – 1943

Comment formaliser les preuves, par exemple sur les entiers naturels.

Hilbert et les formalistes

David Hilbert, 1862 – 1943

Comment formaliser les preuves, par exemple sur les entiers naturels.

Définir un système de preuves

- Des axiomes

$$x = x + 0 \qquad 0 \neq x + 1 \qquad (x + y) + 1 = x + (y + 1)$$

Hilbert et les formalistes

David Hilbert, 1862 – 1943

Comment formaliser les preuves, par exemple sur les entiers naturels.

Définir un système de preuves

- Des axiomes

$$x = x + 0 \qquad 0 \neq x + 1 \qquad (x + y) + 1 = x + (y + 1)$$

- Des règles de déduction : modus ponens et généralisation

$$\frac{\varphi \qquad \varphi \longrightarrow \psi}{\psi} \qquad \frac{\varphi}{\forall x,\ \varphi}$$

Hilbert et les formalistes

David Hilbert, 1862 – 1943

Comment formaliser les preuves, par exemple sur les entiers naturels.

Définir un système de preuves

- Des axiomes

$$x = x + 0 \qquad 0 \neq x + 1 \qquad (x + y) + 1 = x + (y + 1)$$

- Des règles de déduction : modus ponens et généralisation

$$\frac{\varphi \qquad \varphi \longrightarrow \psi}{\psi} \qquad \frac{\varphi}{\forall x,\ \varphi}$$

- Une preuve est une suite de déductions à partir des axiomes

Hilbert et les formalistes

David Hilbert, 1862 – 1943

Comment formaliser les preuves, par exemple sur les entiers naturels.

Définir un système de preuves

- Des axiomes

$$x = x + 0 \qquad 0 \neq x + 1 \qquad (x + y) + 1 = x + (y + 1)$$

- Des règles de déduction : modus ponens et généralisation

$$\frac{\varphi \qquad \varphi \longrightarrow \psi}{\psi} \qquad \frac{\varphi}{\forall x,\ \varphi}$$

- Une preuve est une suite de déductions à partir des axiomes

Propriétés d'un système de preuves :

Cohérence Si un énoncé est prouvable alors il est vrai.
On ne peut pas prouver un énoncé et aussi prouver son contraire.

Complétude Si un énoncé est vrai alors il est prouvable.

Post et le calcul des propositions

Emil Post, 1897 – 1954

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

Théorème : 1921

Le calcul des propositions est cohérent et complet.

Post et le calcul des propositions

Emil Post, 1897 – 1954

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

Théorème : 1921

Le calcul des propositions est cohérent et complet.

Mais il n'est pas très puissant et ne permet pas de formaliser des énoncés simples sur les entiers comme

Pour tout entier n, il existe un entier $p \geq n$ qui est premier.

Pour cela on utilise le calcul des prédicats (logique du premier ordre).

Gödel et l'incomplétude

Kurt Gödel, 1906 – 1978

Théorème : 1931

Aucun système de preuves n'est complet pour le calcul des prédicats sur les entiers

Théorème : 1931

Il n'est pas possible de prouver la cohérence d'un système de preuves (suffisamment expressif) avec ce même système de preuves.

Mais quel rapport avec la calculabilité ?

Gödel et l'incomplétude

Kurt Gödel, 1906 – 1978

Théorème : 1931

Aucun système de preuves n'est complet pour le calcul des prédicats sur les entiers

Théorème : 1931

Il n'est pas possible de prouver la cohérence d'un système de preuves (suffisamment expressif) avec ce même système de preuves.

Mais quel rapport avec la calculabilité ?

- Pour sa preuve, Gödel a besoin de représenter les formules par des entiers.
- On doit pouvoir effectivement calculer le nombre qui représente une formule.
- On doit pouvoir effectivement retrouver une formule à partir du nombre qui la représente.

Gödel avait donc besoin de définir une classe de fonctions effectivement calculables.

Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{rccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline \end{array}$$

Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{cccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline & 6 & 2 & 7 & 0 & 2 \end{array}$$

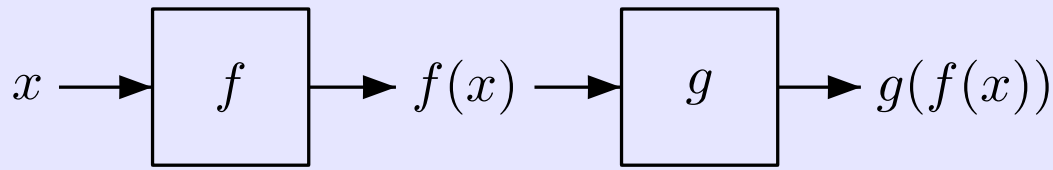
Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{cccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline & 6 & 2 & 7 & 0 & 2 \end{array}$$

- La composition de fonctions



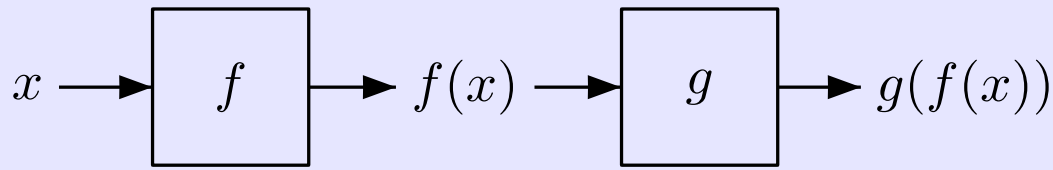
Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{cccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline & 6 & 2 & 7 & 0 & 2 \end{array}$$

- La composition de fonctions



- L'itération de fonctions : par exemple la multiplication

$$\mathrm{prod}(n,m) = \begin{cases} 0 & \text{si } m = 0 \\ n + \mathrm{prod}(n, m-1) & \text{sinon.} \end{cases}$$

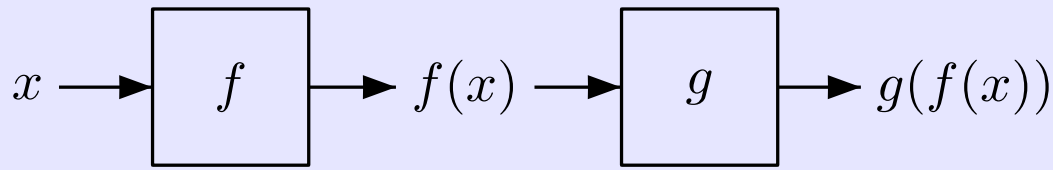
Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{cccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline & 6 & 2 & 7 & 0 & 2 \end{array}$$

- La composition de fonctions



- L'itération de fonctions : par exemple la multiplication

$$\mathrm{prod}(n,m) = \begin{cases} 0 & \text{si } m = 0 \\ n + \mathrm{prod}(n, m-1) & \text{sinon.} \end{cases}$$

```
x <- 0; Pour i de 1 à m faire x <- x + n fpour; Retourner x
```

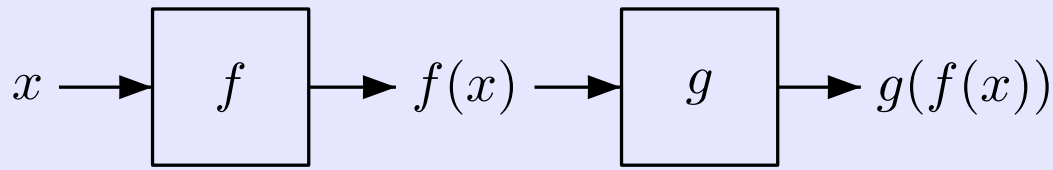
Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{cccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline & 6 & 2 & 7 & 0 & 2 \end{array}$$

- La composition de fonctions



- L'itération de fonctions : par exemple la multiplication

$$\mathrm{prod}(n,m) = \begin{cases} 0 & \text{si } m = 0 \\ n + \mathrm{prod}(n, m-1) & \text{sinon.} \end{cases}$$

```
x <- 0; Pour i de 1 à m faire x <- x + n fpour; Retourner x
```

Y a-t-il des fonctions calculables qui ne sont pas primitives récursives ?

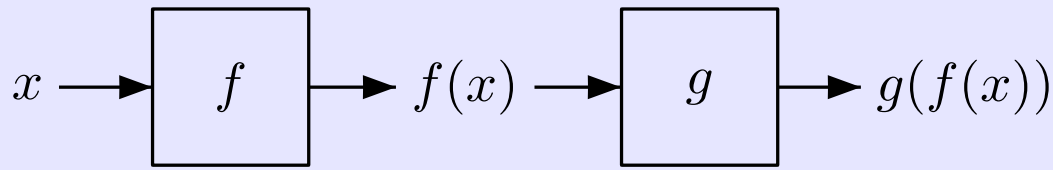
Gödel et les fonctions calculables

Les fonctions primitives récursives (1931 – 1934)

- L'addition (ou simplement l'incrémentation)

$$\begin{array}{cccccc} & 5 & 8 & 4 & 5 & 7 \\ + & & 4 & 2 & 4 & 5 \\ \hline & 6 & 2 & 7 & 0 & 2 \end{array}$$

- La composition de fonctions



- L'itération de fonctions : par exemple la multiplication

$$\mathrm{prod}(n,m) = \begin{cases} 0 & \text{si } m = 0 \\ n + \mathrm{prod}(n, m-1) & \text{sinon.} \end{cases}$$

```
x <- 0; Pour i de 1 à m faire x <- x + n fpour; Retourner x
```

Y a-t-il des fonctions calculables qui ne sont pas primitives récursives ?
Oui : la fonction d'Ackermann (Wilhelm, 1896 – 1962)

Church et le λ-calcul

Alonzo Church, 1903 – 1995

1936

- Church introduit le λ-calcul, ancêtre de la programmation fonctionnelle.
- Il montre que toutes les fonctions primitives récursives sont définissables dans le λ-calcul.

Kleene et la minimisation

Stephen C. Kleene, 1909 – 1994

Les fonctions μ-récursives (1936)

À la composition et à l'itération des fonctions,
Kleene ajoute la définition par minimisation (non bornée):

$$g(x) = \min\{y \mid f(x,y) = 1 \qquad \}$$

Remarque : $\mathrm{dom}(g) \subseteq \mathbb{N}$ mais la fonction g n'est pas forcément totale.

Kleene et la minimisation

Stephen C. Kleene, 1909 – 1994

Les fonctions μ-récursives (1936)

À la composition et à l'itération des fonctions,
Kleene ajoute la définition par minimisation (non bornée):

$$g(x) = \min\{y \mid f(x,y) = 1 \qquad \}$$

Remarque : $\mathrm{dom}(g) \subseteq \mathbb{N}$ mais la fonction g n'est pas forcément totale.

Calcul effectif de $g(x)$ pour $x \in \mathrm{dom}(g)$:

```
y <- 0; z <- f(x,y)
Tant que z ≠ 1 faire y <- y+1; z <- f(x,y) FinTantQue
Retourner y
```

Kleene et la minimisation

Stephen C. Kleene, 1909 – 1994

Les fonctions μ-récursives (1936)

À la composition et à l'itération des fonctions,
Kleene ajoute la définition par minimisation (non bornée):

$$g(x) = \min\{y \mid f(x,y) = 1 \text{ et } [0 \cdot\cdot y] \subseteq \mathrm{dom}(f(x,-))\}$$

Remarque : $\mathrm{dom}(g) \subseteq \mathbb{N}$ mais la fonction g n'est pas forcément totale.

Calcul effectif de $g(x)$ pour $x \in \mathrm{dom}(g)$:

```
y <- 0; z <- f(x,y)
Tant que z ≠ 1 faire y <- y+1; z <- f(x,y) FinTantQue
Retourner y
```

Kleene et la minimisation

Stephen C. Kleene, 1909 – 1994

Les fonctions μ-récursives (1936)

À la composition et à l'itération des fonctions,
Kleene ajoute la définition par minimisation (non bornée):

$$g(x) = \min\{y \mid f(x,y) = 1 \text{ et } [0 \cdot\cdot y] \subseteq \mathrm{dom}(f(x,-))\}$$

Remarque : $\mathrm{dom}(g) \subseteq \mathbb{N}$ mais la fonction g n'est pas forcément totale.

Calcul effectif de $g(x)$ pour $x \in \mathrm{dom}(g)$:

```
y <- 0; z <- f(x,y)
Tant que z ≠ 1 faire y <- y+1; z <- f(x,y) FinTantQue
Retourner y
```

Théorème : Kleene 1936

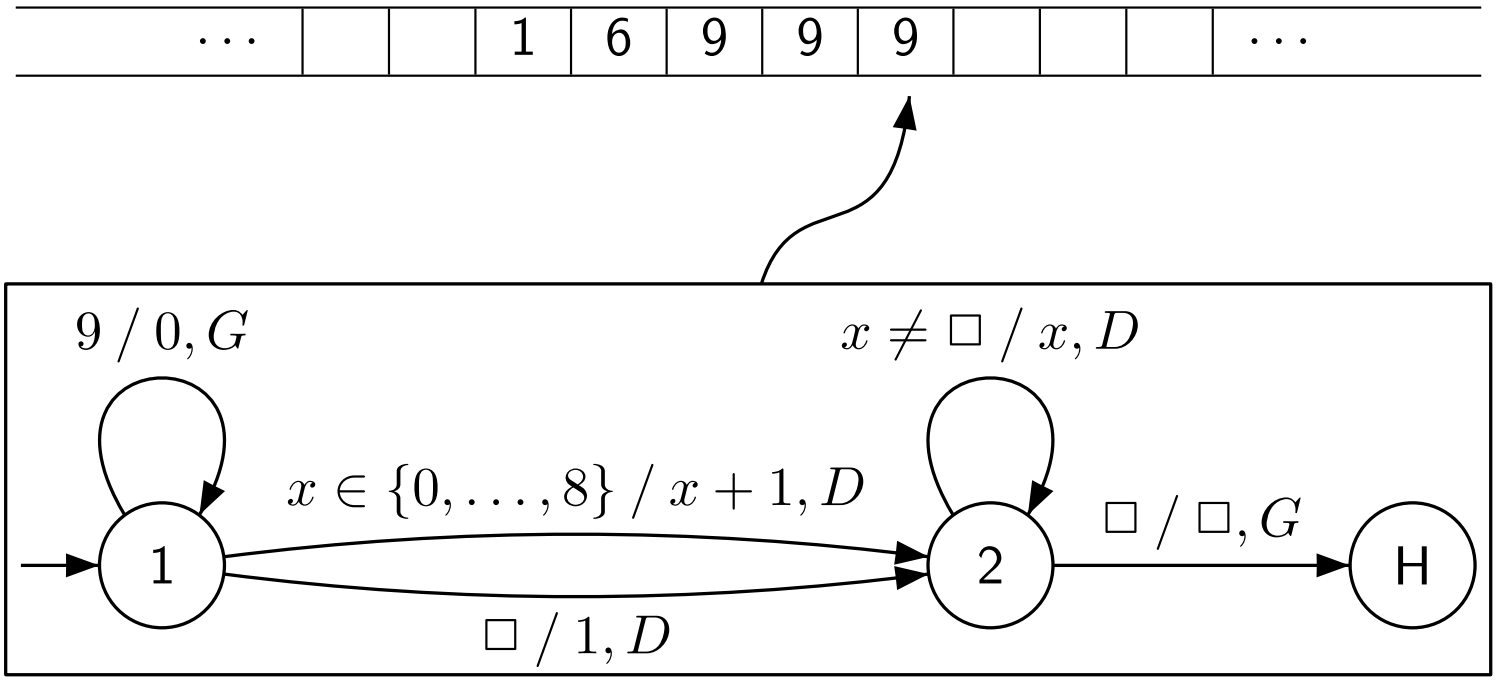
Une fonction est μ-récursive si et seulement si elle est λ-définissable.

Turing et les machines

Alan Turing, 1912 – 1954

http://math.hws.edu/TMCM/java/xTuringMachine/

http://www.labri.fr/perso/betrema/MC/TuringJNLP.html

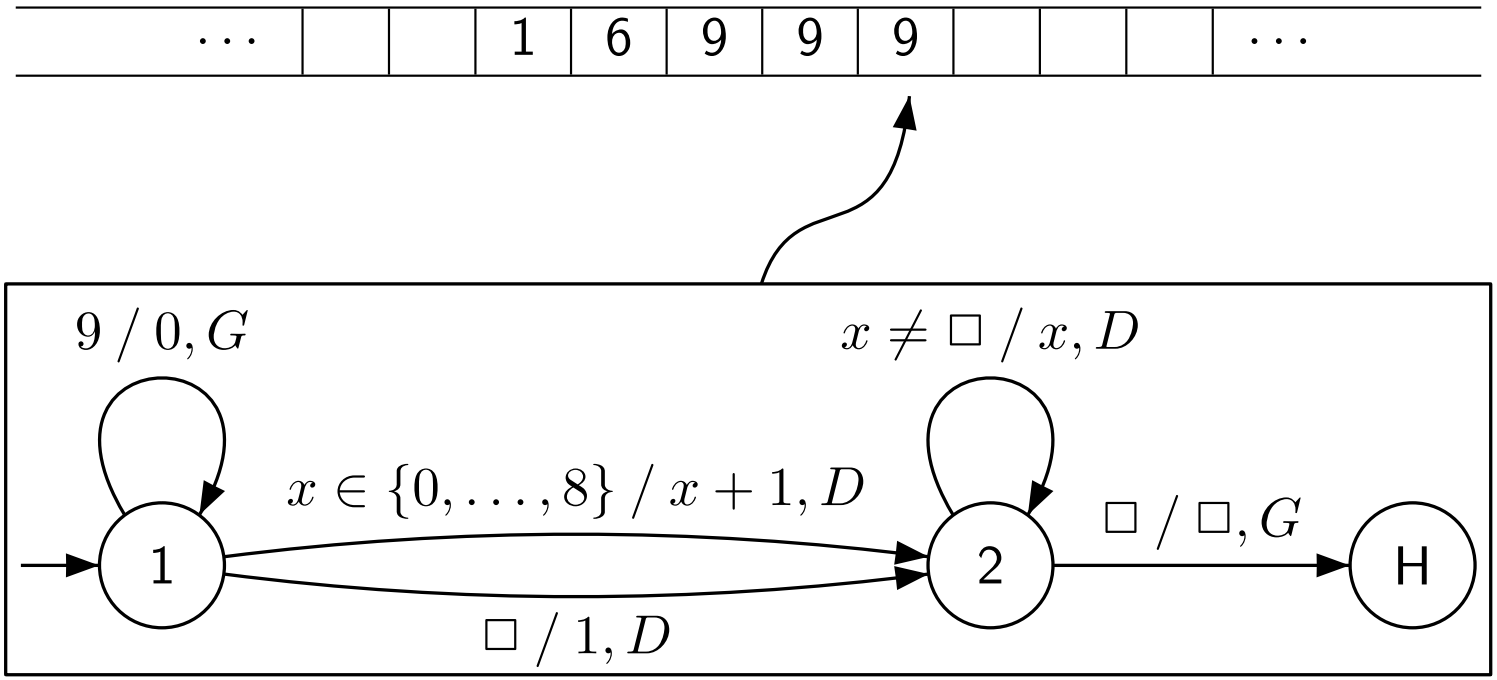


Turing et les machines

Alan Turing, 1912 – 1954

http://math.hws.edu/TMCM/java/xTuringMachine/

http://www.labri.fr/perso/betrema/MC/TuringJNLP.html



Théorème : Turing 1936

Une fonction est μ-récursive si et seulement si elle est calculable par une machine de Turing.

La thèse de Church

Thèse de Church

La notion de effectivement calculable est capturée par les formalismes équivalents suivants :

- Les machines de Turing
- Les fonctions μ-récursives
- Le λ-calcul
- Les processus combinatoires finis (Post, 1936)
- Les algorithmes de Markov (1954)
- Les machines RAM (Shepherdson and Sturgis, 1963)
- Les ordinateurs modernes

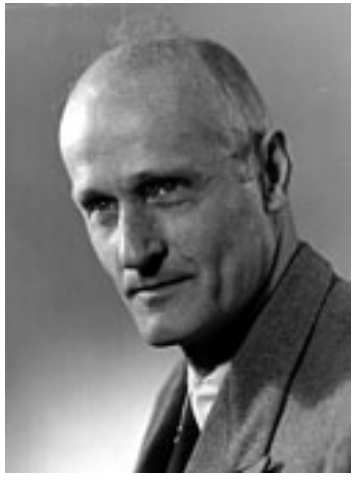
Quelques portraits

David Hilbert
1862 – 1943

Emil Post
1897 – 1954

Kurt Gödel
1906 – 1978

Alonzo Church
1903 – 1995



Stephen C. Kleene
1909 – 1994

Alan Turing
1912 – 1954

Plan

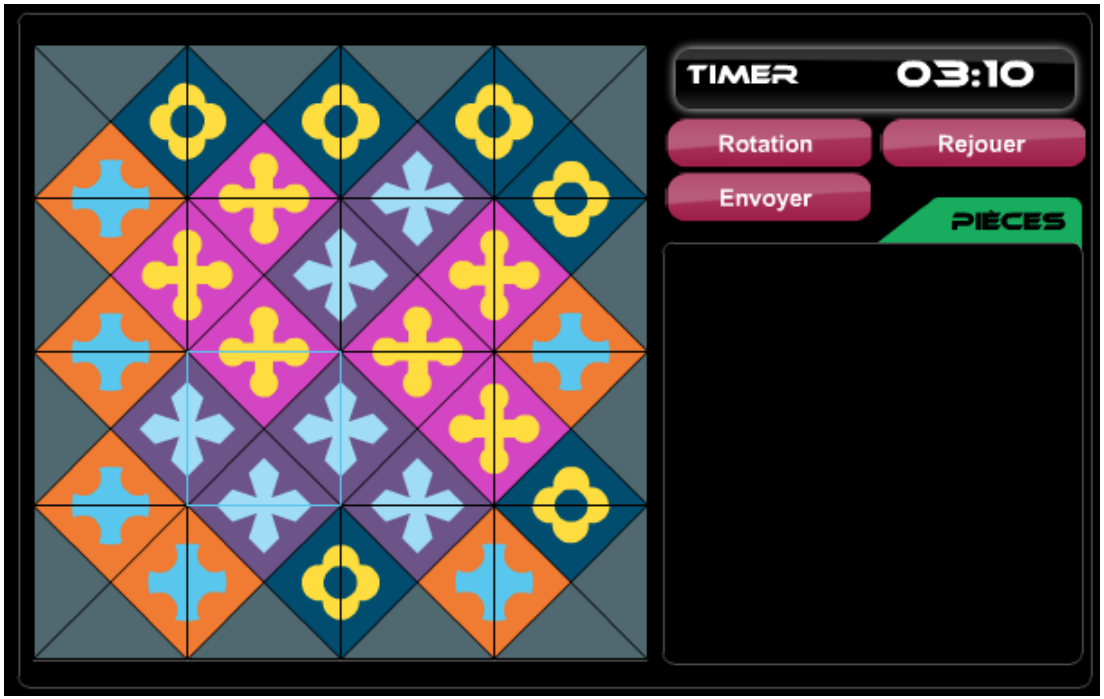
Objectifs

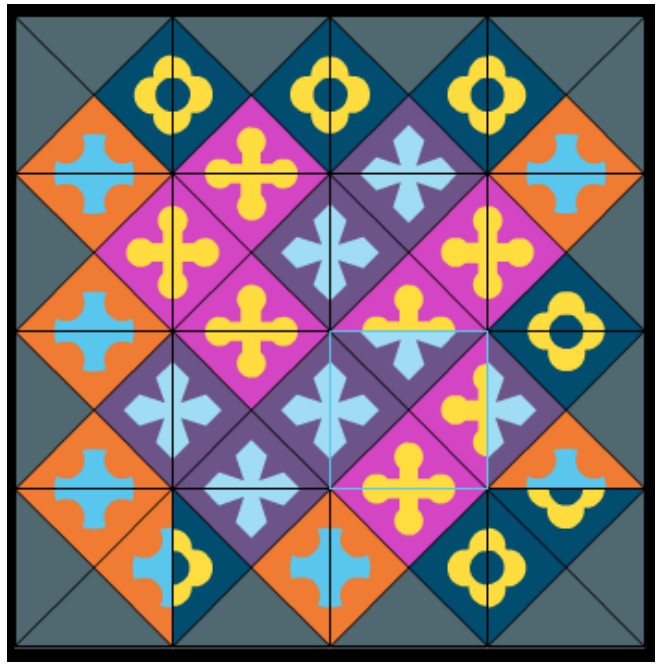
Complexité des problèmes

- Combien de temps faut-il pour résoudre un problème ?
- Trouvera-t-on un jour des algorithmes radicalement plus efficaces ?
- Est-ce que facile à vérifier équivaut à facile à résoudre ?
- ...

Facile à vérifier : Ethernity

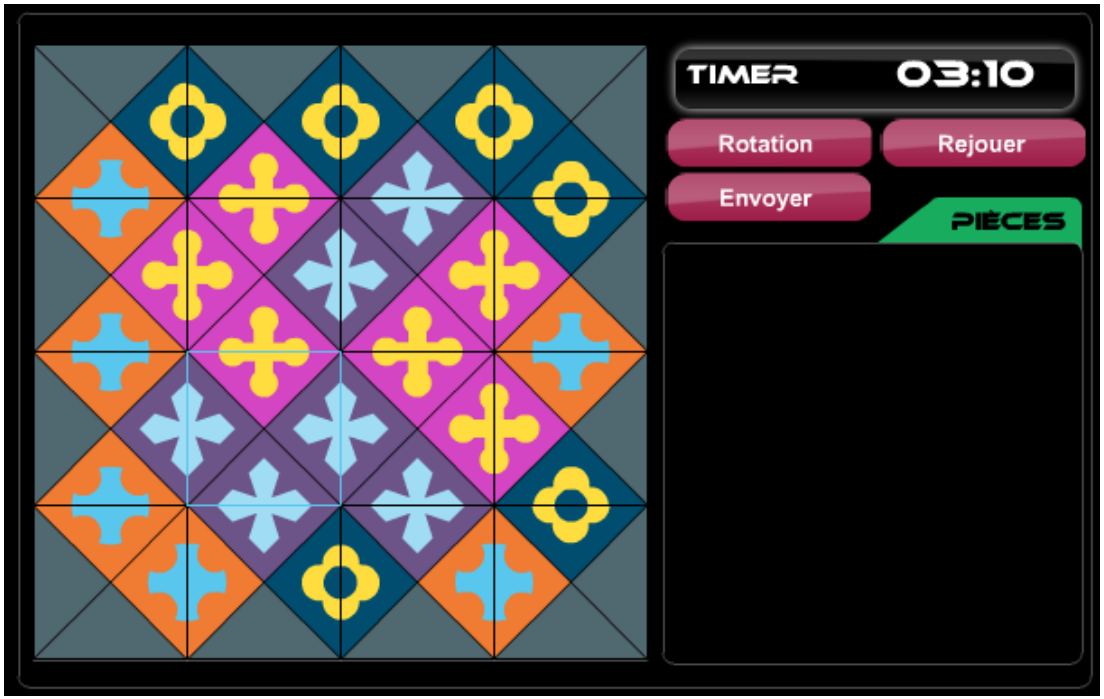
http://fr.eternityii.com/

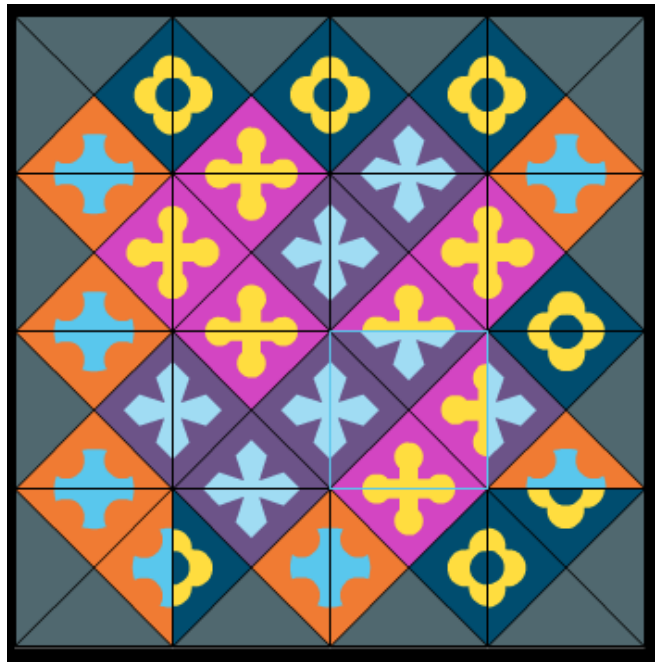




Facile à vérifier : Ethernity

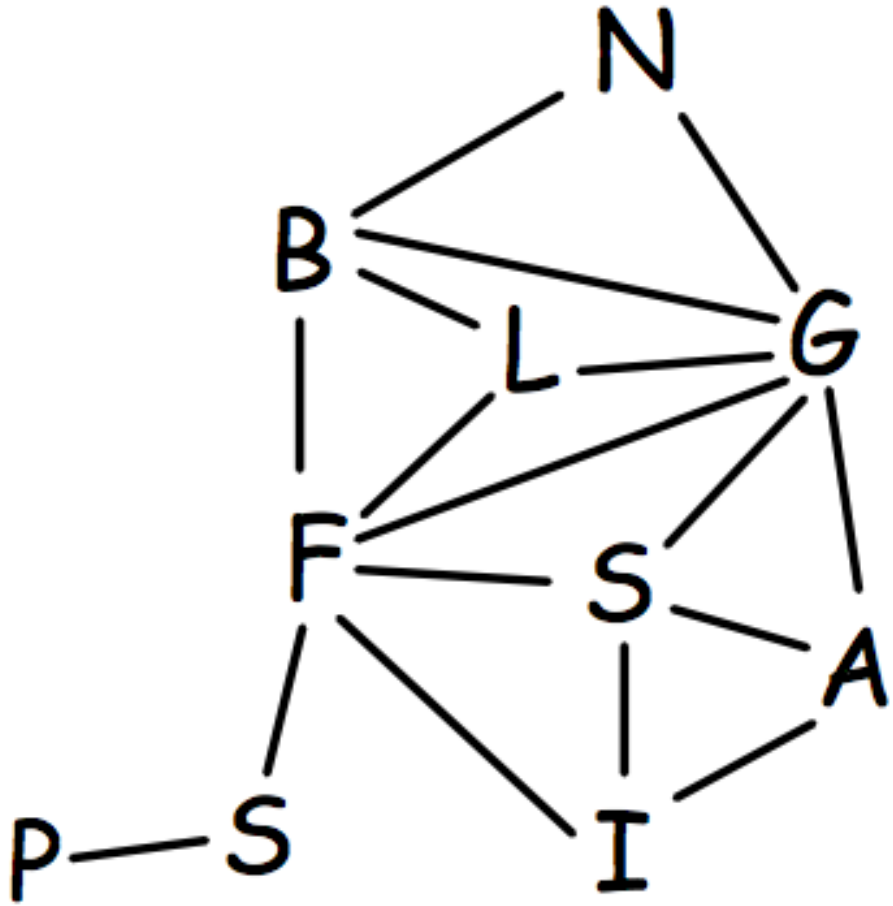
http://fr.eternityii.com/



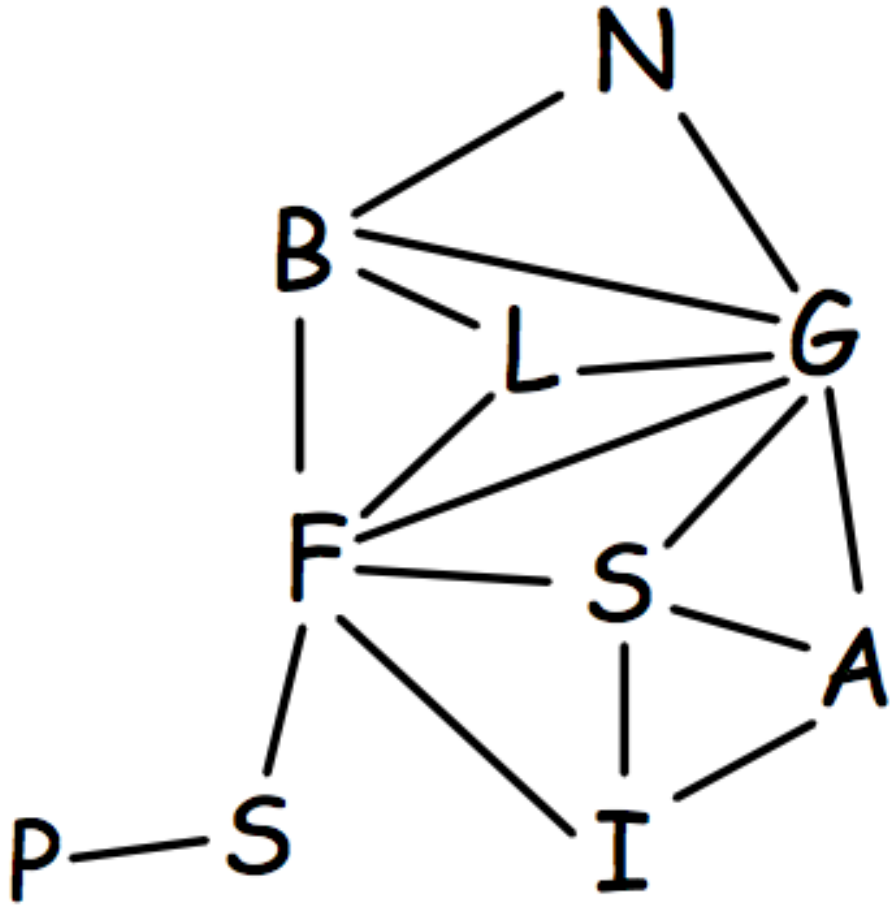


Facile = en temps polynomial

Facile à trouver : 2-coloriage



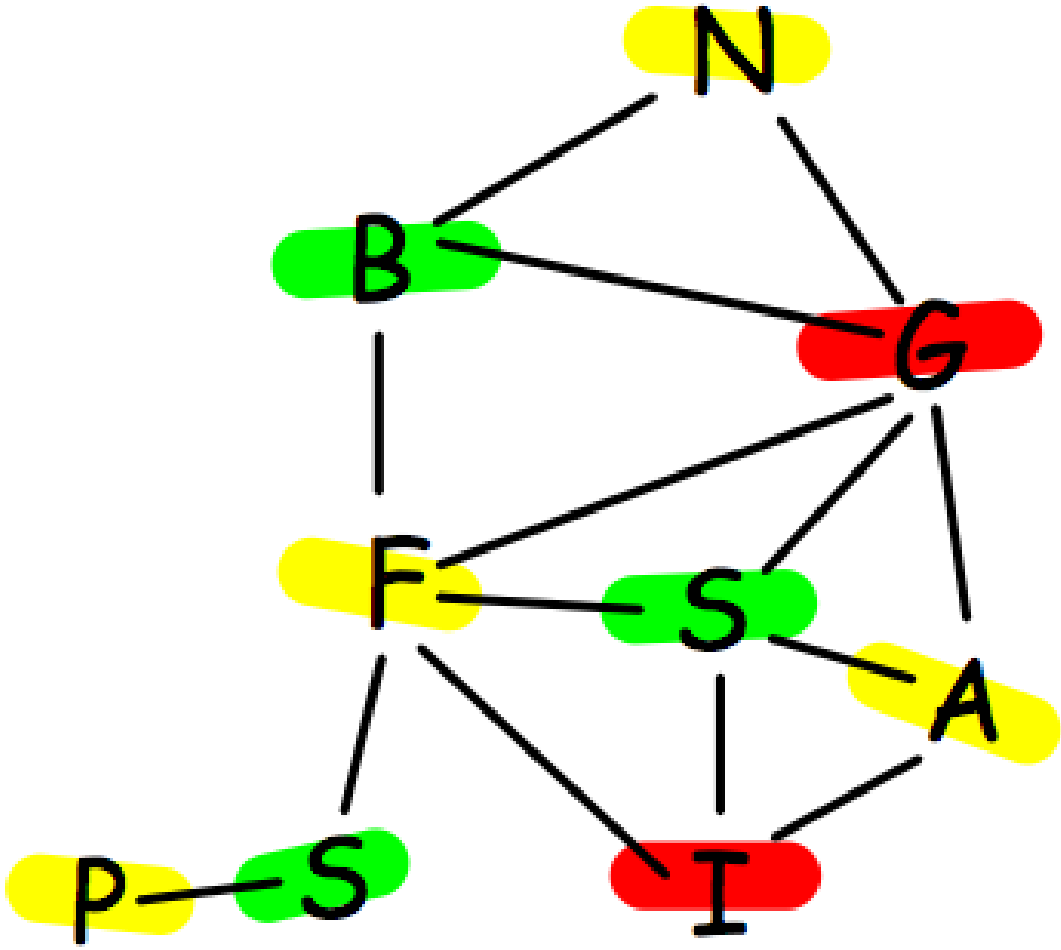
Facile à trouver : 2-coloriage



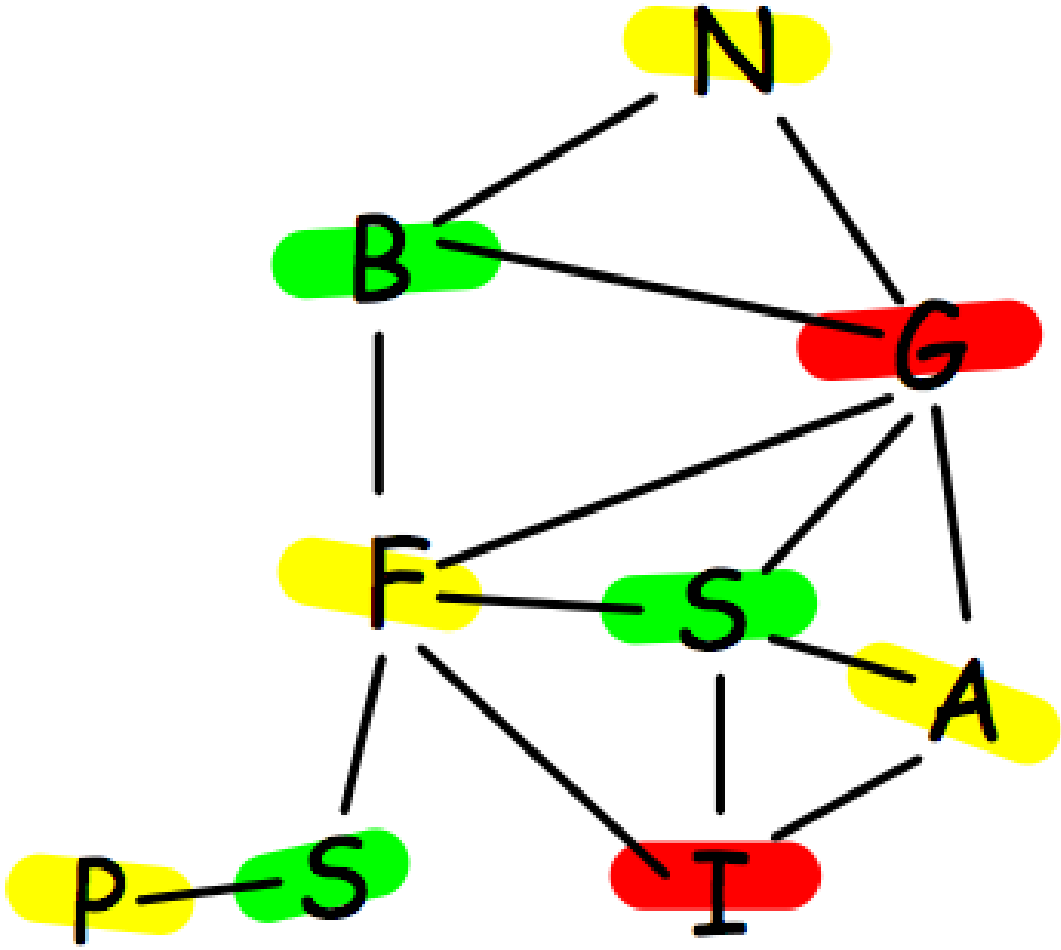
Théorème :

Un graphe est 2-coloriable si et seulement si il n'a pas de cycle de longueur impaire.

Facile à vérifier : 3-coloriage



Facile à vérifier : 3-coloriage



Pourra-t-on trouver un jour un algorithme
pour résoudre facilement le problème du 3-coloriage ?

Facile à vérifier : somme d'entiers

Donnée : une suite d'entier $n_1, n_2, \ldots, n_k$ et un entier S.

Problème : Trouver une sous suite de somme S.

Facile à vérifier : Le problème SAT

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

$(x_1$ ou $x_2)$ et $(x_1$ ou non x_2 ou $x_3)$ et (non x_1 ou x_2 ou $x_4)$ et (non x_3 ou non $x_4)$

x_1	Vrai
x_2	Faux
x_3	Faux
x_4	Vrai

Facile à vérifier : Le problème SAT

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

$(x_1$ ou $x_2)$ et $(x_1$ ou non x_2 ou $x_3)$ et (non x_1 ou x_2 ou $x_4)$ et (non x_3 ou non $x_4)$

x_1	Vrai
x_2	Faux
x_3	Faux
x_4	Vrai

Pourra-t-on trouver un jour un algorithme
pour résoudre facilement la satisfaisabilité ?

Réduction

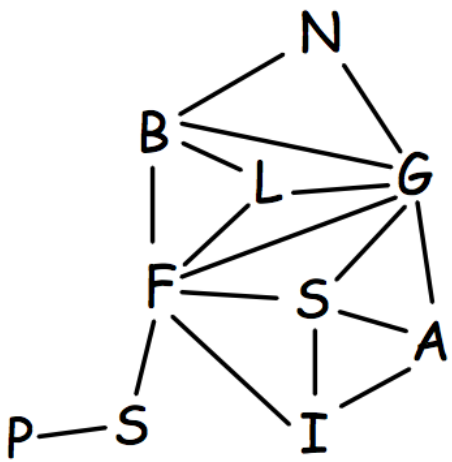
Définition : Réduction

Une réduction d'un problème A à un problème B est un programme R qui transforme très facilement une donnée x de A en une donnée $R(x)$ de B de telle sorte que

x est solution de A si et seulement si $R(x)$ est solution de B

Exemple :

Le problème du 3-coloriage se réduit au problème de satisfaisabilité.



$\longrightarrow$ Réduction $\longrightarrow$

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

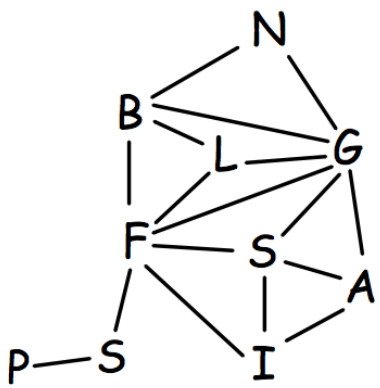
Réduction

Proposition :

Si R est une réduction de A à B et que P est un programme pour résoudre B alors

$$R\,;P$$

est un programme pour résoudre A.



$\longrightarrow$ Réduction $\longrightarrow \varphi_G \longrightarrow$ Programme pour SAT $\longrightarrow$ Oui / Non

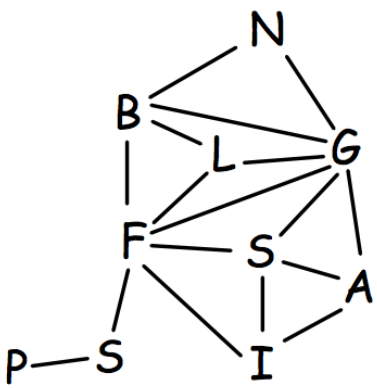
Réduction

Proposition :

Si R est une réduction de A à B et que P est un programme pour résoudre B alors

$$R\,;P$$

est un programme pour résoudre A.



$\longrightarrow$ Réduction $\longrightarrow \varphi_G \longrightarrow$ Programme pour SAT $\longrightarrow$ Oui / Non

Si on trouve un algorithme polynomial pour résoudre SAT
alors on aura un algorithme polynomial pour le 3-coloriage.

Complexité

Remarque :

Si A se réduit à B et B se réduit à A, alors A et B ont la même complexité

Exemple :

Le problème du 3-coloriage est équivalent au problème de satisfaisabilité.

Classes de complexité : P versus NP

Définition : P et NP

- Un problème est dans la classe PTIME (P) si on peut le résoudre en temps polynomial.
- Un problème est dans la classe NPTIME (NP) si on peut deviner une réponse et vérifier en temps polynomial qu'elle est correcte.
  Formellement : non déterminisme.

Classes de complexité : P versus NP

Définition : P et NP

- Un problème est dans la classe PTIME (P) si on peut le résoudre en temps polynomial.
- Un problème est dans la classe NPTIME (NP) si on peut deviner une réponse et vérifier en temps polynomial qu'elle est correcte.
  Formellement : non déterminisme.

Théorème : SAT est NP-complet — Stephen Cook, Leonid Levin, 1971

Le problème SAT est dans la classe NP et
tout problème de la classe NP peut se réduire à SAT (NP-difficile),

Classes de complexité : P versus NP

Définition : P et NP

- Un problème est dans la classe PTIME (P) si on peut le résoudre en temps polynomial.
- Un problème est dans la classe NPTIME (NP) si on peut deviner une réponse et vérifier en temps polynomial qu'elle est correcte.
  Formellement : non déterminisme.

Théorème : SAT est NP-complet — Stephen Cook, Leonid Levin, 1971

Le problème SAT est dans la classe NP et
tout problème de la classe NP peut se réduire à SAT (NP-difficile),

Exemple : Les problèmes suivants sont NP-complets

3-coloriage, Voyageur de commerce, Somme d'entiers, Ethernity, Sudoku, ...

La question à 1 million de dollars : $\mathrm{P} = \mathrm{NP}$ ou $\mathrm{P} \neq \mathrm{NP}$?

`http://www.claymath.org/millennium/P_vs_NP/`

Autres classes de complexité

Différentes notions de “facile”

- Espace logarithmique
- Temps polynomial
- Espace polynomial
- Temps exponentiel

Autres classes de complexité

Différentes notions de "facile"

- Espace logarithmique
- Temps polynomial
- Espace polynomial
- Temps exponentiel

Théorème : Hiérarchie

$$\mathrm{LOGSPACE} \subseteq \mathrm{NLOGSPACE} \subseteq \mathrm{PTIME} \subseteq \mathrm{NPTIME} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE}$$

$$\mathrm{PSPACE} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME} \subseteq \mathrm{EXPSPACE} = \mathrm{NEXPSPACE}$$

Plan

Autres classes de complexité

[1] Jean-Michel Autebert.
Calculabilité et Décidabilité.
Masson, 1992.

[2] Olivier Carton.
Langages formels, calculabilité et complexité.
Vuibert, 2008.

[3] Richard L. Epstein, Walter A. Carnielli.
Computability: Computable Functions, Logic, and the Foundations of Mathematics.
Wadsworth, 2000 (2nd édition).

[4] Dexter C. Kozen.
Automata and Computability.
Springer, 1997.

[5] Christos Papadimitriou.
Computational complexity.
Addison-Wesley, 1995.