Éléments d'informatique – Cours 12. Indécidabilité de l'arrêt, théorème de Rice

Pierre Boudes

Paris 13

13 décembre 2011

Contenu du partiel

- Au *partiel*, vous aurez le mêmes types d'exercices qu'au premier partiel avec différents types de données (`int`, `char`, `double`, `struct`) et des fonctions :
  - commenter, compléter un programme (cours)
  - simuler l'exécution d'un programme (trace)
  - if et constantes symboliques (arbre de décision)
  - écriture de boucles (for ou while ?)
  - un exercice porte spécialement sur les structures
  - un exercice supplémentaire : déclarer/définir des fonctions avec $\frac{1}{2}$pt par déclaration juste et $\frac{1}{2}$pt par définition juste

Contenu du partiel

- Au *partiel*, vous aurez le mêmes types d'exercices qu'au premier partiel avec différents types de données (`int`, `char`, `double`, `struct`) et des fonctions :
  - commenter, compléter un programme (cours)
  - simuler l'exécution d'un programme (trace)
  - if et constantes symboliques (arbre de décision)
  - écriture de boucles (for ou while ?)
  - un exercice porte spécialement sur les structures
  - un exercice supplémentaire : déclarer/définir des fonctions avec $\frac{1}{2}$pt par déclaration juste et $\frac{1}{2}$pt par définition juste
  - et une question bonus notée sévèrement
- le programme à modifier/compléter, la trace, l'exercice sur les `struct` sont forcément avec fonctions
- Gérez votre temps, apprenez à lire un sujet et ne rien oublier. Nous testons vos acquis pas votre propension à paniquer !

Structure et contenu d'un programme C

```c
/* Declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */
...

/* Declaration des constantes et types utilisateurs */
...

/* Declaration des fonctions utilisateurs */
...

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}

/* Definitions des fonctions utilisateurs */
...
```

Directives préprocesseur

```
/* Declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */
#include <math.h> /* pow, sqrt */ /* bibliothèque */

/* Declaration des constantes et types utilisateurs */
#define N 5 /* constante symbolique */
#define TRUE 1
#define FALSE 0

/* Declaration des fonctions utilisateurs */
...

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    int donnee[N];
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}
```

Types utilisateurs struct

```
...
/* Declaration des constantes et types utilisateurs */
...
struct paire_s
{
    int g; /* gauche */
    int d; /* droite */
} ;

/* Declaration des fonctions utilisateurs */
...

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    struct paire_s meschaussures = {37, 44};
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}

/* Definitions des fonctions utilisateurs */
```

Fonctions : déclarations (type), appels, définitions

```
...
/* Declaration des fonctions utilisateurs */
int factorielle(int n);                       /*       Z -> Z     */
int pgcd(int a, int b);                       /* Z x Z -> Z     */
double neper(int ordre);                      /*       Z -> R     */
void afficher_paire(struct paire_s x); /* paire -> rien */
int saisie_choix();                          /*  rien -> Z     */

/* Fonction principale */
int main()
{
    ...
    afficher_paire(meschaussures); /* appel */
    ...
}

/* Definitions des fonctions utilisateurs */
double neper(int n) /* définition de la fonction neper */
{
    ...
    somme = somme + 1.0 / factorielle(k); /* appel */
    ...
    return somme;
```

Fonctions récursives

```
...
/* Declaration des fonctions utilisateurs */
int factorielle(int n);                    /*     Z -> Z    */
int pgcd(int a, int b);                    /* Z x Z -> Z    */
double neper(int ordre);                   /*     Z -> R    */
void afficher_paire(struct paire_s x); /* paire -> rien */
int saisie_choix();                       /*  rien -> Z    */

/* Fonction principale */
int main()
{
    ...
    afficher_paire(meschaussures); /* appel */
    ...
}

/* Definitions des fonctions utilisateurs */
double neper(int n) /* définition de la fonction neper */
{
   if (n > 1)
   {
       return 1.0 / factorielle(n)  + neper(n - 1); /* appel récursif
   ...
```

Algorithmes : quels outils pour quels problèmes

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas
   - boucle `while`, conditions booléenn `if`

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas
   - boucle `while`, conditions booléenn `if`

2'. Parcourir/générer : une *ligne* mais au une *surface*, un *volume*...

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas
   - boucle `while`, conditions booléenn `if`

2'. Parcourir/générer : une *ligne* mais au une *surface*, un *volume*...
   - imbriquer les boucles (var. $\neq$)

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas
   - boucle `while`, conditions booléenn `if`

2'. Parcourir/générer : une *ligne* mais au une *surface*, un *volume*...
   - imbriquer les boucles (var. $\neq$)

6. Boucle événementielle
   - boucle while

Algorithmes : quels outils pour quels problèmes

1. Traiter des cas spécifiques
   - `if else` (différencier)
   - `#define` constantes symboliques (nommer)
   - arbre de décision (organiser)
2. Parcourir/générer des cas
   - boucle `for` (rarement while)
   - tableaux
3. Composer des cas
   - boucles (parcourir/générer)
   - accumulateur (à initialiser)

3'. Dénombrer des cas
   - boucles (parcourir/générer)
   - compteur (à initialiser à 0)

4. Sélectionner des cas
   - boucles (parcourir/générer)
   - `if` (sélectionner/traiter)
5. Rechercher un cas
   - boucle `while`, conditions booléenn `if`

2'. Parcourir/générer : une *ligne* mais au une *surface*, un *volume*...
   - imbriquer les boucles (var. $\neq$)

6. Boucle événementielle
   - boucle while
7. Attente active
   - boucle while

Problèmes indécidables, les limites de l'analyse automatique

- Un problème de décision est un énoncé à paramètres qui selon les valeurs des paramètres peut être vrai ou faux. Un programme informatique (un calcul) résout un problème de décision lorsque il prend en entrée les paramètres du problème de décision et détermine si l'énoncé est vrai ou faux pour ces paramètres.

Problèmes indécidables, les limites de l'analyse automatique

- Un problème de décision est un énoncé à paramètres qui selon les valeurs des paramètres peut être vrai ou faux. Un programme informatique (un calcul) résout un problème de décision lorsque il prend en entrée les paramètres du problème de décision et détermine si l'énoncé est vrai ou faux pour ces paramètres.
- Par exemple, déterminer si un entier naturel quelconque est premier est un problème de décision. Ce problème est décidable : il existe un programme (un calcul) qui le résout.

Problèmes indécidables, les limites de l'analyse automatique

- Un problème de décision est un énoncé à paramètres qui selon les valeurs des paramètres peut être vrai ou faux. Un programme informatique (un calcul) résout un problème de décision lorsque il prend en entrée les paramètres du problème de décision et détermine si l'énoncé est vrai ou faux pour ces paramètres.
- Par exemple, déterminer si un entier naturel quelconque est premier est un problème de décision. Ce problème est décidable : il existe un programme (un calcul) qui le résout.
- Certains problèmes sont indécidables c'est à dire qu'il n'existe pas de programme qui les résolve.

Problèmes indécidables, les limites de l'analyse automatique

- Un problème de décision est un énoncé à paramètres qui selon les valeurs des paramètres peut être vrai ou faux. Un programme informatique (un calcul) résout un problème de décision lorsque il prend en entrée les paramètres du problème de décision et détermine si l'énoncé est vrai ou faux pour ces paramètres.
- Par exemple, déterminer si un entier naturel quelconque est premier est un problème de décision. Ce problème est décidable : il existe un programme (un calcul) qui le résout.
- Certains problèmes sont indécidables c'est à dire qu'il n'existe pas de programme qui les résolve.
- Il est important de savoir si un problème est décidable avant de chercher à le résoudre.

Problèmes indécidables, les limites de l'analyse automatique

- Un problème de décision est un énoncé à paramètres qui selon les valeurs des paramètres peut être vrai ou faux. Un programme informatique (un calcul) résout un problème de décision lorsque il prend en entrée les paramètres du problème de décision et détermine si l'énoncé est vrai ou faux pour ces paramètres.
- Par exemple, déterminer si un entier naturel quelconque est premier est un problème de décision. Ce problème est décidable : il existe un programme (un calcul) qui le résout.
- Certains problèmes sont indécidables c'est à dire qu'il n'existe pas de programme qui les résolve.
- Il est important de savoir si un problème est décidable avant de chercher à le résoudre.
- Nous allons voir que de nombreux problèmes sont indécidables. Le plus fameux d'entre eux est le problème de l'arrêt : être capable de déterminer, pour tout programme informatique, s'il s'arrêtera de lui-même ou continuera son exécution éternellement.

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

- On **suppose** l'existence d'une telle bijection $p : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ (une énumération de $\mathcal{P}(\mathbb{N})$).
- Soit la partie $D = \{k \in \mathbb{N} \mid k \notin p(k)\}$.

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

- On **suppose** l'existence d'une telle bijection $p : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ (une énumération de $\mathcal{P}(\mathbb{N})$).
- Soit la partie $D = \{k \in \mathbb{N} \mid k \notin p(k)\}$.
- D doit avoir un numéro dans l'énumération. Soit n tel que $p(n) = D$.

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

- On **suppose** l'existence d'une telle bijection $p : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ (une énumération de $\mathcal{P}(\mathbb{N})$).
- Soit la partie $D = \{k \in \mathbb{N} \mid k \notin p(k)\}$.
- D doit avoir un numéro dans l'énumération. Soit n tel que $p(n) = D$.
- Est-ce que $n \in D$?

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

- On **suppose** l'existence d'une telle bijection $p : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ (une énumération de $\mathcal{P}(\mathbb{N})$).
- Soit la partie $D = \{k \in \mathbb{N} \mid k \notin p(k)\}$.
- D doit avoir un numéro dans l'énumération. Soit n tel que $p(n) = D$.
- Est-ce que $n \in D$? Soit $n \in D$ et par définition de D, $n \notin p(n) = D$ (impossible). Soit $n \notin D = p(n)$ et par définition de D, $n \in D$ (impossible). Contradiction.
- L'hypothèse de départ est donc fausse. □

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

- On **suppose** l'existence d'une telle bijection $p : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ (une énumération de $\mathcal{P}(\mathbb{N})$).
- Soit la partie $D = \{k \in \mathbb{N} \mid k \notin p(k)\}$.
- D doit avoir un numéro dans l'énumération. Soit n tel que $p(n) = D$.
- Est-ce que $n \in D$? Soit $n \in D$ et par définition de D, $n \notin p(n) = D$ (impossible). Soit $n \notin D = p(n)$ et par définition de D, $n \in D$ (impossible). Contradiction.
- L'hypothèse de départ est donc fausse. $\square$

Comme il existe une injection de $\mathbb{N}$ dans $\mathcal{P}(\mathbb{N})$, c'est que $\mathcal{P}(\mathbb{N})$ est strictement plus grand que $\mathbb{N}$.

L'argument de la diagonale inventé par Cantor en 1891

Montrons qu'il n'existe pas de bijection entre $\mathbb{N}$ et les parties de $\mathbb{N}$.

- On **suppose** l'existence d'une telle bijection $p : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ (une énumération de $\mathcal{P}(\mathbb{N})$).
- Soit la partie $D = \{k \in \mathbb{N} \mid k \notin p(k)\}$.
- D doit avoir un numéro dans l'énumération. Soit n tel que $p(n) = D$.
- Est-ce que $n \in D$? Soit $n \in D$ et par définition de D, $n \notin p(n) = D$ (impossible). Soit $n \notin D = p(n)$ et par définition de D, $n \in D$ (impossible). Contradiction.
- L'hypothèse de départ est donc fausse. □

Comme il existe une injection de $\mathbb{N}$ dans $\mathcal{P}(\mathbb{N})$, c'est que $\mathcal{P}(\mathbb{N})$ est strictement plus grand que $\mathbb{N}$.
Sur le même principe on montre (par exemple) que $\mathbb{R}$ ou que l'ensemble des fonctions $\mathbb{N} \to \mathbb{N}$ sont non dénombrables. Ce dernier point montre qu'**il existe beaucoup de fonctions non calculables**.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \rightarrow \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	…
Programme 0	1	1	1	…
Programme 1	1	0	0	…
Programme 2	0	0	1	…
⋮	⋮	⋮	⋮	

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	...
Programme 0	1	1	1	...
Programme 1	1	0	0	...
Programme 2	0	0	1	...
⋮	⋮	⋮	⋮	

- **On suppose** qu'il existe un programme $f(p, q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	…
Programme 0	1	1	1	…
Programme 1	1	0	0	…
Programme 2	0	0	1	…
⋮	⋮	⋮	⋮	

- **On suppose** qu'il existe un programme $f(p, q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p, p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	...
Programme 0	1	1	1	...
Programme 1	1	0	0	...
Programme 2	0	0	1	...
⋮	⋮	⋮	⋮	
Programme n				...
⋮	⋮	⋮	⋮	

- **On suppose** qu'il existe un programme $f(p, q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p, p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	...
Programme 0	1	1	1	...
Programme 1	1	0	0	...
Programme 2	0	0	1	...
⋮	⋮	⋮	⋮	
Programme n	0			...
⋮	⋮	⋮	⋮	

- **On suppose** qu'il existe un programme $f(p, q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p, p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	…
Programme 0	1	1	1	…
Programme 1	1	0	0	…
Programme 2	0	0	1	…
⋮	⋮	⋮	⋮	
Programme n	0	1		…
⋮	⋮	⋮	⋮	

- **On suppose** qu'il existe un programme $f(p,q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p,p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	…
Programme 0	1	1	1	…
Programme 1	1	0	0	…
Programme 2	0	0	1	…
⋮	⋮	⋮	⋮	…
Programme n	0	1	0	…
⋮	⋮	⋮	⋮	

- **On suppose** qu'il existe un programme $f(p, q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p, p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	…	n	…
Programme 0	1	1	1	…		…
Programme 1	1	0	0	…		…
Programme 2	0	0	1	…		…
⋮	⋮	⋮	⋮		⋮	
Programme n	0	1	0	…	?	…
⋮	⋮	⋮	⋮		⋮	

- **On suppose** qu'il existe un programme $f(p, q)$ qui renvoie 1 si le programme n° p s'arrête sur la donnée q et **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p, p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

Est-ce que $g(n)$ s'arrête ? Que vaut $f(n, n)$?

L'indécidabilité de l'arrêt

- On **construit** une énumération des programmes $\mathbb{N} \to \mathbb{N}$.
- Un programme prend un entier en entrée et s'arrête (1) ou boucle éternellement (0).

Donnée :	0	1	2	…	n	…
Programme 0	1	1	1	…		…
Programme 1	1	0	0	…		…
Programme 2	0	0	1	…		…
⋮	⋮	⋮	⋮		⋮	⋮
Programme n	0	1	0	…	?	…
⋮	⋮	⋮	⋮		⋮	⋮

Contradiction

- **On suppose** qu'il existe un program… 1 si le programme n° p s'arrête sur la don… **renvoie** 0 **sinon**.
- On construit un programme g qui prend en entrée un entier p et s'arrête si $f(p, p)$ vaut 0 ou boucle éternellement sinon. Le programme g a lui même un numéro, n.

Est-ce que $g(n)$ s'arrête ? Que vaut $f(n, n)$?

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"`

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"` (indécidable)

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"` (indécidable)
- Le programme contient `printf("bonjour")`

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"` (indécidable)
- Le programme contient `printf("bonjour")` (décidable)

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"` (indécidable)
- Le programme contient `printf("bonjour")` (décidable)
- Le programme effacera mon disque dur

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"` (indécidable)
- Le programme contient `printf("bonjour")` (décidable)
- Le programme effacera mon disque dur (indécidable)

Les limites de l'analyse automatique de programmes

Conclusion. Le problème de l'arrêt est indécidable : il n'est pas possible d'écrire un programme capable d'analyser n'importe quel programme et de déterminer si ce dernier s'arrêtera.

Théorème de Rice

Une généralisation due à Henry Gordon Rice et publiée en 1953, établit que toute propriété des programmes qui s'énonce uniquement sur les entrées-sorties des programmes est soit triviale, soit indécidable.

Exemples :

- Le programme bouclera indéfiniment (indécidable)
- La sortie du programme fera au moins 0 octets (trivial)
- Le programme affichera `"bonjour"` (indécidable)
- Le programme contient `printf("bonjour")` (décidable)
- Le programme effacera mon disque dur (indécidable)

Bonnes révisions !