Calculabilité, logique et complexité

Chapitre 4

Modèles de calculabilité

UCLouvain

Chapitre 4: Modèles de calculabilité

1. Familles de modèles
2. Langages de programmation
3. Automates finis
4. Machines de Turing
5. Fonctions récursives → pas à l'exam

Acquis d'apprentissage

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

- Distinguer les modèles de calculabilité basés sur le calcul de ceux basé sur des langages (ensemble de mots
- Comprendre et expliquer les différences entre un modèle déterministe et non déterministe
- Comprendre, expliquer et justifier les restrictions du langage BLOOP
- Comprendre, expliquer et illustrer le langage ND-Java
- Comprendre, expliquer et appliquer les notions d'ensemble ND-récursif et d'ensemble ND-récursivement énumérable ainsi que leurs propriétés
- Comprendre, expliquer et illustrer le modèle des automates finis, ainsi que ses limitations, ses extensions et ses applications
- Comprendre, expliquer et illustrer le modèle des machines de Turing
- Comprendre, expliquer et appliquer les notions de fonction T-calculable, d'ensemble T-récursif et NDT-récursif, et d'ensembles T-récursivement énumérable et NDT-récursivement énumérable, ainsi que leurs propriétés
- Décrire les différentes extensions des machines de Turing et les analyser du point de vue de leur puissance et de leur efficacité

1. Familles de modèles

Deux familles de modèles

- Modèles basés sur le **calcul**
- Modèles basés sur les **langages** (ensemble de chaines de caractères)

1. Familles de modèles

Modèles basés sur le calcul

Objectif de modéliser le concept de fonctions calculables, processus de calcul, algorithme effectif

Certains modèles sont volontairement limitatifs

Exemples

- Automate fini
- Automate à pile
- Machine de Turing
- Langages de programmation
- Fonctions récursives
- Lambda calcul
- Modèles logico-mathématique
- …

1. Familles de modèles

Modèles basés sur les langages

Rappel: un *langage* est un ensemble de mots constitués de symboles d'un alphabet donné

Une *grammaire formelle* est une définition d'un langage

Objectif d'une grammaire formelle: modéliser une classe de langages

On souhaite bien entendu pouvoir associer à chaque grammaire une méthode permettant de reconnaître les mots du langage défini par la grammaire

Le langage est alors un ensemble

- **récursif**
- ou **récursivement énumérable**

1. Familles de modèles

Déterminisme vs nondéterminisme

Parmi les modèles de calcul, on peut distinguer

- modèles **déterministes** :
  - une seule exécution possible
- modèles **non déterministes** :
  - existence de plusieurs exécutions possibles

2. Langages de programmation

Langage de programmation: modèle possible de la calculabilité, basé sur le calcul

Utilisation du concept de “programme”

- pour formaliser le concept de “procédure” ou “algorithme” effectif
- pour établir les résultats fondamentaux de la calculabilité (Ch. 3).

Langage de programmation comme modèle

Définition :

- syntaxe du langage
- sémantique du langage
- convention de représentation d’une fonction par un programme

2. Langages de programmation

Equivalence des langages de programmation

Existe-t-il des langages de programmation plus puissants que d'autres ?

Les langages tels que

- Algol, APL, Basic, C, C++, C#, Cobol, Fortran, Haskel, Java, Javascript, Lisp, Logo, Miranda, ML, Modula, Pascal, Perl, PHP, Prolog, Python, Ruby, …

sont tous ***équivalents***

Ils permettent de calculer les mêmes fonctions. Ce sont des langages de programmation ***complets***

En pratique, certains langages de programmation sont mieux adaptés à certaines *classes de problèmes*

2. Langages de programmation

Langage BLOOP

Définition d'un langage "Bounded Loop" (boucle bornée)

Sous-ensemble de Java

Définition du langage

Restrictions du langage Java

Un programme BLOOP est un programme Java tel que :

- pas de boucle while
- dans le corps d'une boucle for, pas de modification de la variable compteur
- pas de méthodes récursives ni mutuellement récursives

2. Langages de programmation

Propriétés

- Tous les programmes BLOOP se terminent
- BLOOP ne calcule que des fonctions totales
- BLOOP ne calcule pas toutes les fonctions totales (Pourquoi ?)
- Il existe un compilateur des programmes BLOOP
- L'interpréteur de BLOOP est une fonction totale non programmable en BLOOP (Pourquoi ?)
- Le langage BLOOP n'est pas un modèle complet de la calculabilité

2. Langages de programmation

Langage de programmation non déterministe

Extension du langage Java en une version non déterministe:

ND-Java

Ajout de la fonction prédéfinie ***choose(n)***

- Renvoie un entier compris entre 0 et n
- Fonction non déterministe; les différents résultats sont envisagés

choose(3)

0 1 2 3

Exemple

Problème du voyageur de commerce

Précondition :

- n : nombre de villes à visiter
- dist[0..n-1,0..n-1] : tableau des distances entre les villes
- B : un entier positif

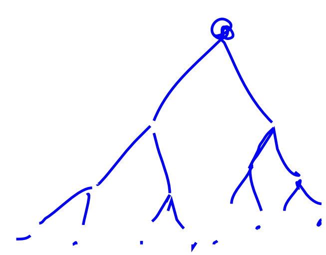
Postcondition:

- YES : si il existe un circuit reliant les n villes et de longueur ≤ B
- NO : sinon

2. Langages de programmation

```
int n = ... ;                      // nombre de villes
int[][] dist = { ... } ;           // dist[i,j] distance entre ville i et j
int[] circuit = new int[n];        // circuit[i] i-ème ville visitée
int long, i, c;

// Génération d'un circuit : une permutation de 0..n-1
for (int i = 0; i < circuit.length; i++) {
    circuit[i] = i;
}
for (int i = 0; i < circuit.length-1; i++) {
    c = i + choose(n-i-1); -> donne un arbre
    swap(circuit[i],circuit[c]);
}
```



2. Langages de programmation

```
// Longueur du circuit
long= 0 ;
for (int i = 0; i < circuit.length-1; i++) {
    long = long + dist[ circuit[i], circuit[i+1] ] ;
}
long= long + dist[ circuit[n-1], circuit[0] ];


// Test de la longueur
if (long <= B) {
   System.out.println("YES");
} else {
   System.out.println("NO");
}
```

à chaque feuille de l'arbre
↳ soit YES
  " NO

2. Langages de programmation

Sémantique

- La fonction choose(n) est non déterministe
- Renvoie une valeur quelconque (entre 0 et n)
- Plusieurs exécutions possibles

On considère *toutes les exécutions possibles*

Comment interpréter le (les) résultat(s) d'un programme non déterministe (ND-programme) ?

- existence d'exécutions infinies et finies
- possibilité de résultats différents dans différentes exécutions

Différentes approches possibles

1. Un programme ND calcule une relation plutôt qu'une fonction
2. Voir un programme ND comme un moyen de décider si un élément appartient à un ensemble

Approche (2) en calculabilité

2. Langages de programmation

Ensemble ND-récursif et ND-récursivement énumérable

Soit $A \subseteq N$

A est ***ND-récursif*** si il existe un programme ND-Java tel que lorsqu'il reçoit comme donnée n'importe quel nombre naturel x,

- si $x \in A$, alors *il existe une exécution* fournissant (tôt ou tard) comme résultat 1
- si $x \notin A$, alors *toutes les exécutions* possibles fournissent (tôt ou tard) comme résultat 0

A est ***ND-récursivement énumérable*** si il existe un programme ND-Java tel que lorsqu'il reçoit comme donnée n'importe quel nombre naturel x, *il existe une exécution* fournissant (tôt ou tard) comme résultat 1 ssi $x \in A$

(si $x \notin A$, les exécutions possibles fournissent un résultat ($\neq 1$) ou *ne se termine pas*)

2. Langages de programmation

Propriétés

Possibilité de simuler les exécutions d'un ND-programme à l'aide d'un programme déterministe : un interpréteur de ND-programmes

- Comment réaliser cet interpréteur ?
- Si, pour une donnée de taille n, il existe une exécution fournissant un résultat avec une complexité temporelle de O(f(n)), alors l'interpréteur de ND-programmes fournira ce résultat avec une complexité temporelle de $O(2^{f(n)})$

Théorème :

Un ensemble est ND-récursif **ssi** il est récursif

Théorème :

Un ensemble est ND-récursivement énumérable **ssi** il est récursivement énumérable

3. Automates finis

Modélisation élémentaire du concept de “calcul”

- nombre fini d’états
- lecture d’une donnée: un mot (chaîne de caractères)
- chaque symbole de la donnée est lu *une et une seule* fois
- transitions entre états en fonction du symbole lu
- état final = état après avoir lu tous les symboles de la donnée
- *pas* de possibilité de *mémorisation*

Objectif d’un automate fini (FA) :

Décider si un mot donné *appartient* ou *non* à un langage (ensemble de mots)

3. Automates finis

Modèle des automates finis

Un automate fini est composé de

- Σ : ensemble (fini) de symboles
- S : ensemble (fini) d'états
- $s_0 \in S$: état initial
- $A \subseteq S$: ensemble des états acceptant
- $\delta: S \times \Sigma \rightarrow S$: fonction de transition

Exemple

- $\Sigma = \{ 0, 1\}$
- S = { pair1, impair1}
- pair1 : état initial
- impair1: état acceptant
- fonction de transition :

	0	1
pair1	pair1	impair1
impair1	impair1	pair1

3. Automates finis

Modèle de calcul

Un automate fini est un modèle de calcul

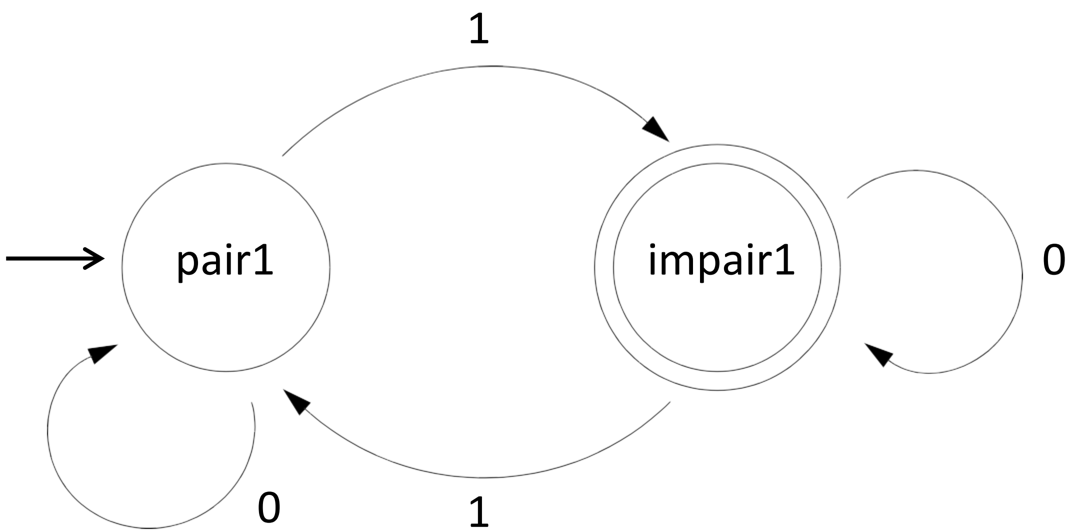
- départ avec état initial
- parcourir les symboles du mot d'entrée, un à un
- à chaque symbole lu, changer d'état (fonction de transition) en fonction de l'état courant et du symbole lu
- état final est l'état après avoir parcouru tous les symboles du mot d'entrée

Ce modèle de calcul peut être simulé par un programme Java :
un interpréteur d'automates finis

3. Automates finis

Exemple

Autre représentation de l’exemple précédant



Si le mot d’entrée contient un nombre impair de 1, alors l’état final sera acceptant (impair1)

Sinon, l’état final ne sera pas acceptant (pair1)

3. Automates finis

Cet automate fini permet donc de reconnaître les chaîne de caractères, composées de 1 et de 0, contenant un nombre impair de 1

L = { m $\in$ {0,1}* | m contient un nombre impair de 1 }

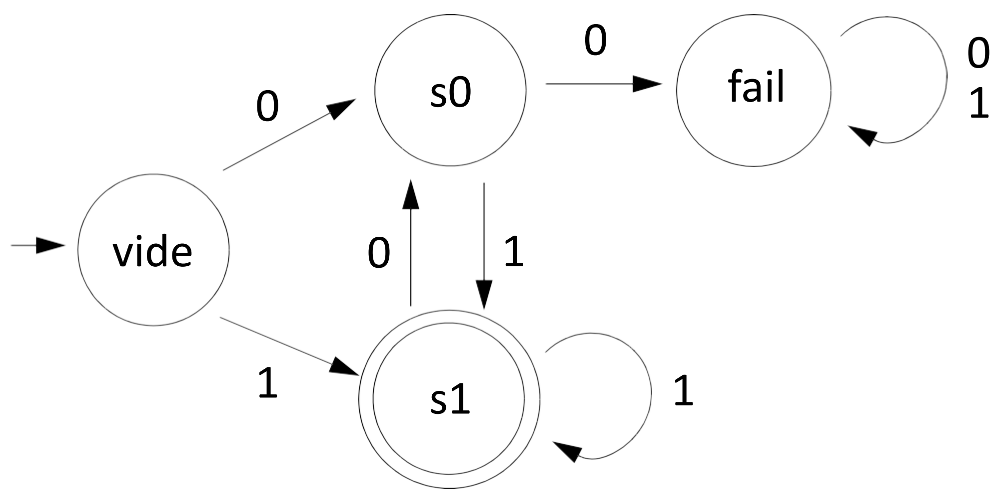
$L = 0^{*}10^{*} (10^{*}10^{*})^{*}$

3. Automates finis

Exemple

- $\Sigma = \{0, 1\}$
- S = { vide, s0, s1, fail}
- vide: état initial
- s1: état acceptant
- fonction de transition :

	0	1
vide	s0	s1
s0	fail	s1
s1	s0	s1
fail	fail	fail



3. Automates finis

Si le mot d'entrée se termine par 1 et ne contient pas d'occurrence de 00, alors l'état final sera acceptant (s1)

Sinon, l'état final ne sera pas acceptant (fail)

L = { m ∈ {0,1}* | *m* se termine par 1 et ne contient pas d'occurrence de 00 }

$L = (1 + 01)^{+}$

Exercice

Construire un automate fini qui accepte les mots de {0,1}* contenant un nombre pair de 0 et un nombre impair de 1

3. Automates finis

Automate fini et ensemble récursif

Etant donné un automate fini (FA),

- un mot m est accepté par FA si après exécution de FA avec m comme donnée, l'état final est *acceptant*
- un mot m n'est pas accepté par FA si après exécution de FA avec m comme donnée, l'état final n'est *pas acceptant*

Un automate fini défini un ***ensemble récursif*** de mots

$$\{ m \mid m \text{ est accepté par FA} \}$$

3. Automates finis

Propriétés des automates finis

- Les automates finis définissent des **ensembles récursifs** (de mots)
- Certains ensembles récursifs ne peuvent être reconnus par un automate fini (Pourquoi ?)

  Exemple: $L = \{ a^n b^n \mid n \geq 0 \}$
- La fonction interpréteur des automates finis est calculable
- L'interpréteur des automates finis ne peut être représenté par un automate fini (Pourquoi ?)
- Le modèle des automates finis n'est pas un modèle complet de la calculabilité

3. Automates finis

Extension des automates finis

Automates finis non déterministes (NDFA)

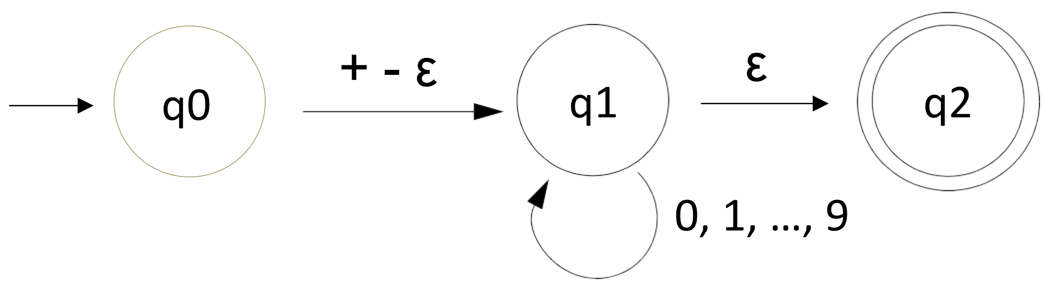
- Plusieurs transitions possibles pour une paire <état,symbole> (éventuellement pas de transition)
- Plusieurs exécutions possibles pour une même donnée
- Un mot m est accepté par NDFA si il existe une exécution de NDFA avec m comme donnée telle que après cette exécution, l'état final est *acceptant*
- Un mot m n'est pas accepté par NDFA si pour toute exécution de NDFA avec m comme donnée, cette exécution ne se termine pas avec un état final *acceptant*
- Un NDFA défini un ensemble récursif de mots
- Si un ensemble récursif est défini par un NDFA, alors cet ensemble peut être défini par un automate fini (déterministe)

3. Automates finis

Extension des automates finis

Automates finis avec transition vide (ε-NDFA)

- Possibilité de transition sans lire de symbole, transition *spontanée* pour une paire <état,ε>
- Extension du principe d'exécution pour tenir compte des transitions vides



- Quel est le langage reconnu par cet automate ?
- Un ε-NDFA défini un ensemble récursif de mots
- Si un ensemble récursif est défini par un ε-NDFA, alors cet ensemble peut être défini par un automate fini (déterministe).
- Possibilité de transformer automatiquement un ε-NDFA en FA équivalent.

3. Automates finis

Applications des automates finis

- Analyse lexicale dans un compilateur (découpe en tokens)
- Recherche de patterns dans un texte (expressions régulières)
- Editeurs de texte
- Interfaces utilisateurs

4. Machines de Turing

Motivation

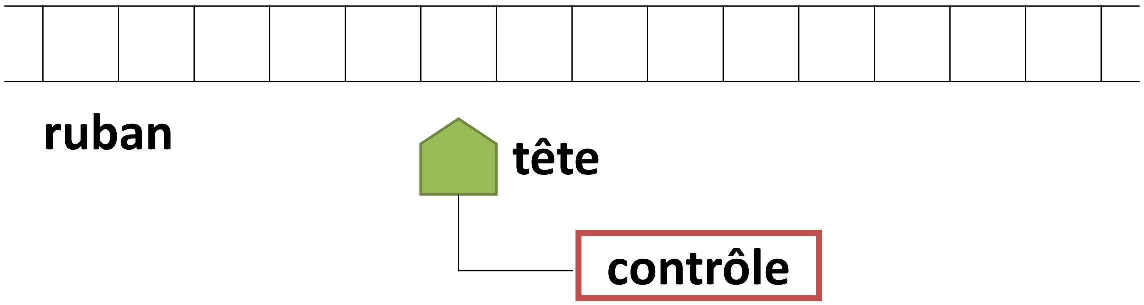
A. Turing, 1936

- Définition, caractérisation précise de la notion de procédure effective, d'algorithme ou de calcul
- Modèle le plus *simple*, le plus *élémentaire*, mais le plus puissant possible

4. Machines de Turing

Organisation d'une machine de Turing

Machine :



Ruban

- potentiellement infini (des deux côtés)
- à chaque moment, le ruban nécessaire est fini

Tête

- une seule tête, sur une case
- peut lire et écrire dans une case

Contrôle

- dirige les actions / opérations

4. Machines de Turing

Contrôle

Ensemble fini d'états parmi lesquels

- un état initial
- un état d'arrêt

Le contrôle contient

- un programme (instructions)
- un mécanisme qui exécute les instructions

Forme d'une instruction

< q , c > → < new_q , Mouv , new_c>

- q : état courant
- c : symbole sous la tête de lecture
- new_c : symbole à écrire sous la tête de lecture
- Mouv : mouvement (G ou D) de la tête de lecture à effectuer (aller à gauche ou aller à droite d'une case)
- new_q : état suivant (après exécution de cette instruction)

4. Machines de Turing

Modélisation

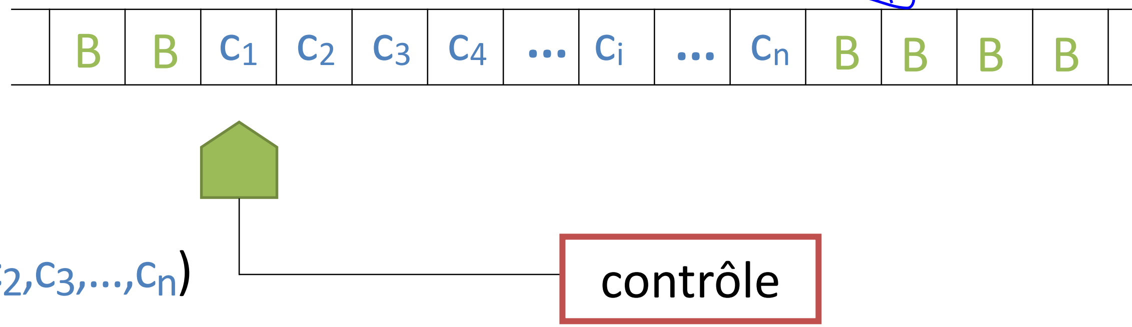
Une machine de Turing (MT) est composée de

- Σ : ensemble fini de symboles d'entrée
- Γ : ensemble fini de symboles du ruban
  - $\Sigma \subset \Gamma$
  - $B \in \Gamma$, $B \notin \Sigma$ (symbole blanc)
- S : ensemble fini d'états
- $s_0 \in S$: état initial
- $stop \in S$: état d'arrêt
- $\delta: S \times \Gamma \rightarrow S \times \{G,D\} \times \Gamma$: fonction de transition (fini)

4. Machines de Turing

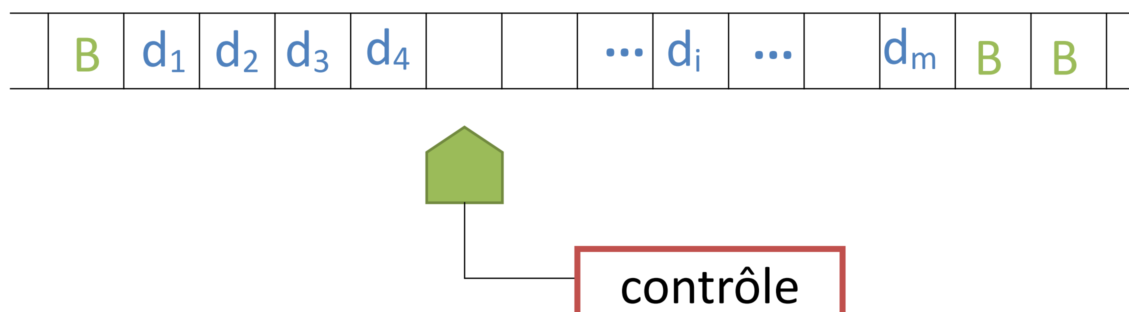
caractère blanc

Exécution



- Donnée sur le ruban ($c_1,c_2,c_3,\dots,c_n$)
- Autres cases contiennent le symbole blanc (B)
- Tête de lecture initialement sur première case de la donnée
- Tant que des instructions sont applicables, le contrôle applique ces instructions
- L'exécution s'arrête dès que l'état devient *stop*
- S'il n'y a pas d'instruction applicable, il y a un arrêt
- Résultat : contenu du ruban à l'état *stop*

  ($d_1, d_2, d_3, \dots, d_m$)



4. Machines de Turing

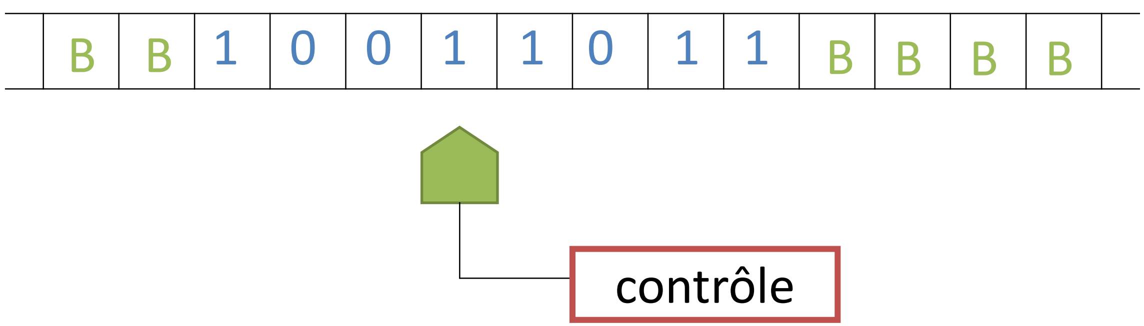
Exemple

Machine de Turing calculant la fonction

$$f(x) = x + 1$$

Représentation des entiers

- sous forme binaire
- $\Sigma = \{ 0,1 \}$
- $\Gamma = \{ 0,1, B\}$ pas d'autres symboles (à part B)



1. Positionner la tête de lecture sur le bit de poids le plus faible
2. Réaliser l'addition et les reports nécessaires
3. Report final et fin

4. Machines de Turing

Positionner tête de lecture à droite

état	symb.	état	mouv.	symb.
début	0	début	D	0
début	1	début	D	1
début	B	report	G	B

Addition

état	symb.	état	mouv.	symb.
report	0	stop	G	1
report	1	report	G	0
report	B	stop	G	1

4. Machines de Turing

état	symb.	état	mouv.	symb.
début	0	début	D	0
début	1	début	D	1
début	B	report	G	B

état	symb.	état	mouv.	symb.
report	0	stop	G	1
report	1	report	G	0
report	B	stop	G	1

Exemple d'exécution

état	gauche	tête	droite
début		1	0011011
début	1	0	011011
début	10	0	11011
…	…	…	…
début	1001101	1	
début	10011011		
report	1001101	1	
report	100110	1	0
report	10011	0	00
stop	1001	1	100

4. Machines de Turing

Exemple

état	symb.	état	mouv.	symb.
début	0	début	D	0
début	1	début	D	1
début	B	début	D	B

Quelle sera l'exécution de cette machine de Turing sur une donnée quelconque ?

Exercice

Construire une machine de Turing qui, recevant une représentation décimale d'un entier n, fourni comme résultat la représentation décimale de 2n

4. Machines de Turing

Fonctions T-calculables

Conventions :

- représentation des entiers sous forme décimale
- $\Sigma = \{ 0,1, 2, ..., 9 \}$
- $\Gamma = \{ 0,1, 2, ..., 9, B\}$
- résultat : représentation décimale (on enlève les blancs éventuel entre chiffres) des symboles du ruban lorsque état *stop*
- pas de résultat si pas d'arrêt avec état *stop*
- fonctions $N \to N$

Une fonction f est ***T-calculable*** ssi il existe une machine de Turing 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 arrêt état différent de *stop*) si $f(x) = \perp$

Extension aisée aux fonctions $N^k \to N$

4. Machines de Turing

Ensembles T-récursifs

Soit A $\subseteq$ *N*

A est ***T-récursif*** si il existe une machine de Turing 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

A est ***T-récursivement énumérable*** si il existe une machine de Turing 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

- si x $\notin$A, machine de Turing fourni un résultat ($\neq$ 1), s'arrête avec un état différent de *stop* ou ne se termine pas

4. Machines de Turing

Thèse de Church-Turing

1. Toute fonction T-calculable est calculable
2. Toute fonction calculable est T-calculable
3. Tout ensemble T-récursif est récursif
4. Tout ensemble récursif est T-récursif
5. Tout ensemble T-récursivement énumérable est récursivement énumérable
6. Tout ensemble récursivement énumérable est T-récursivement énumérable

(1), (3) et (5) sont des théorèmes

(2), (4) et (6) sont des thèses

4. Machines de Turing

Extension du modèle

Possibilité de modifier le modèle de base des machines de Turing

Questions à se poser :

- le nouveau modèle est-il plus ***puissant*** ? (permettant de calculer plus de fonctions)
- le nouveau modèle est-il plus ***efficace*** ? (moins d'étapes pour obtenir le résultat)

1. Changement des conventions
2. Autres types de ruban
3. Plusieurs têtes de lecture
4. Machine de Turing non déterministe
5. Machine de Turing avec oracle

4. Machines de Turing

Autres conventions

- Possibilité de se déplacer de plusieurs cases à la fois
- Plusieurs états stop : $stop_1$, ..., $stop_k$

- Même puissance
- Speedup linéaire

4. Machines de Turing

Symboles et états

Possibilité de réduire les symboles :

- $\Sigma = \{ 0, 1\}$
- $\Gamma = \{ 0, 1, B\}$
  - Même puissance
  - Même efficacité (facteur logarithmique)

Si limitation du nombre d'états (avec symboles fixés),
alors seulement un nombre fini de machines de Turing différentes

Modèle moins puissant

(Sauf si existence d'une machine de Turing universelle parmi ces machines)

4. Machines de Turing

Autres rubans

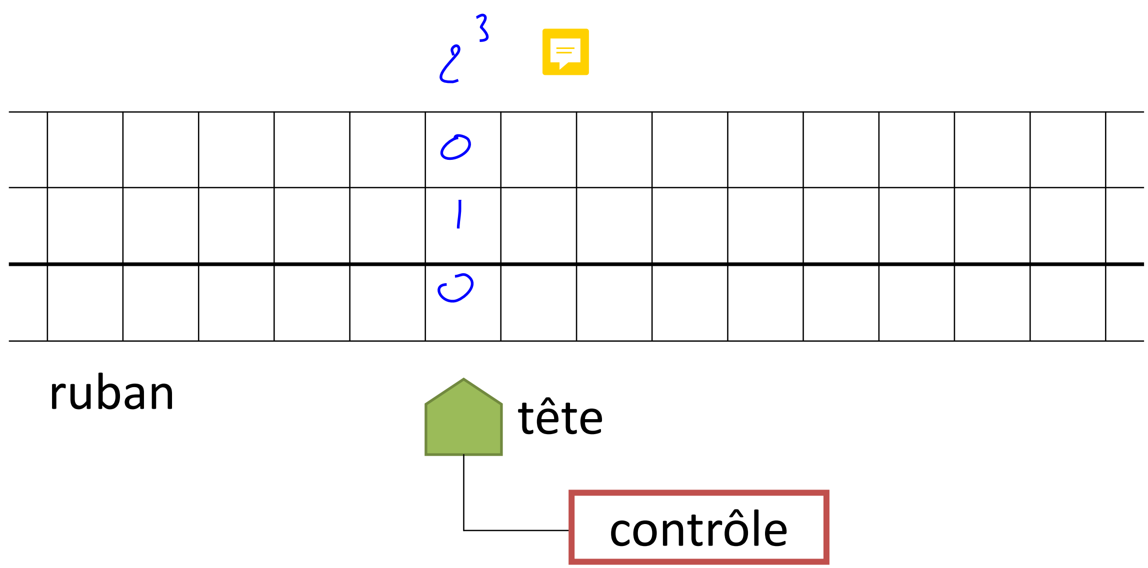
Ruban unidirectionnel

Ruban limité d'un côté (à gauche)

Arrêt si déplacement à gauche de la dernière case

- Même puissance
- Slowdown linéaire

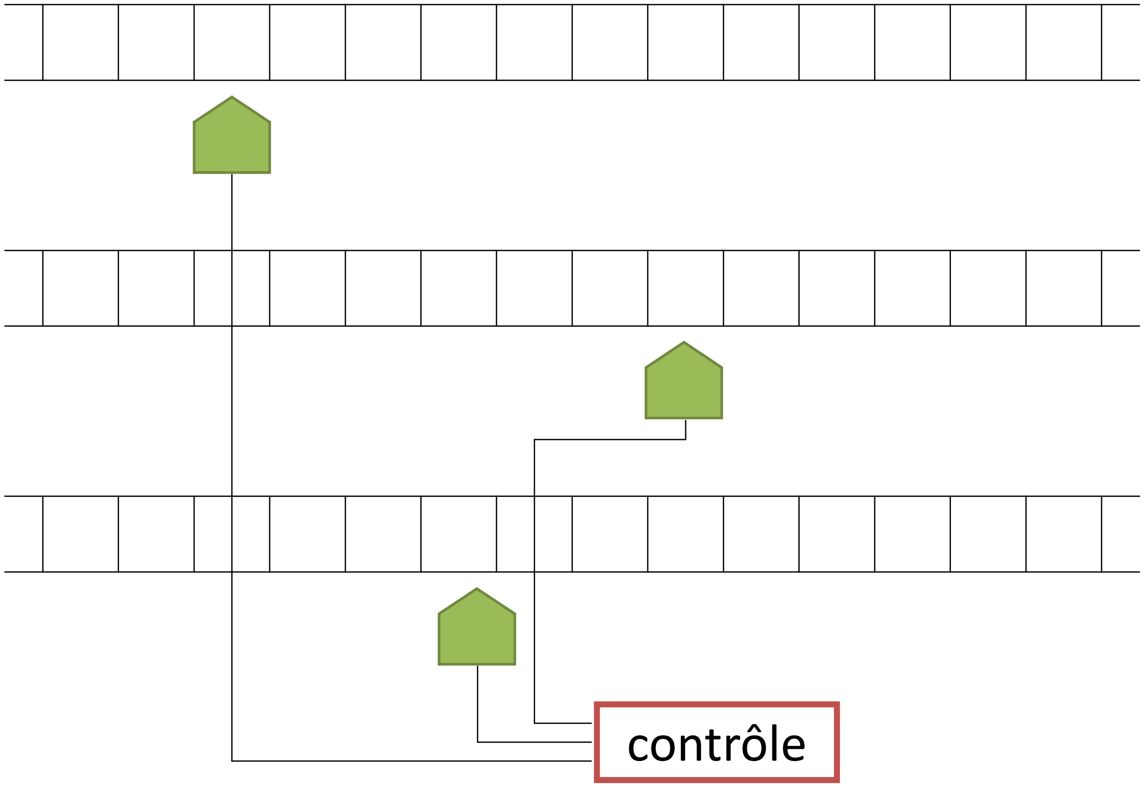
Ruban multicases



- Même puissance
- Même efficacité

4. Machines de Turing

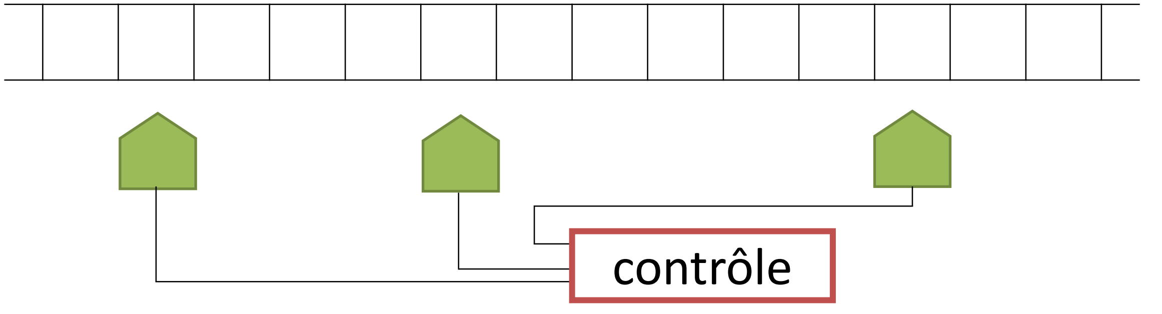
Plusieurs rubans



- Même puissance
- Speedup quadratique

4. Machines de Turing

Plusieurs têtes



- Même puissance
- Speedup quadratique

4. Machines de Turing

Machine de Turing non déterministe

- Plusieurs transitions possibles pour une paire <état,symbole>
  - $\Delta \subseteq S \times \Gamma \times S \times \{G,D\} \times \Gamma$: ***relation*** de transition (finie)
- Plusieurs exécutions possibles pour une même donnée
- Une machine de Turing non déterministe (ND - Machine de Turing) est uniquement utilisée comme un *moyen de décider* si un élément appartient à un ensemble

4. Machines de Turing

Ensemble NDT-récursif et NDT-récursivement énumérable

Soit A $\subseteq N$

A est ***NDT-récursif*** si il existe une ND-machine de Turing telle que lorsqu'elle reçoit comme donnée n'importe quel nombre naturel x,

- si x $\in$A, alors il existe une exécution fournissant (tôt ou tard) comme résultat 1
- si x $\notin$A, alors toutes les exécutions possibles fournissent (tôt ou tard) comme résultat 0

A est ***NDT-récursivement énumérable*** si il existe une ND-machine de Turing telle que lorsqu'elle reçoit comme donnée n'importe quel nombre naturel x, il existe une exécution fournissant (tôt ou tard) comme résultat 1 ssi x $\in$ A

(si x $\notin$ A, les exécutions possibles fournissant un résultat ($\neq$ 1), s'arrête avec un état $\neq$ *stop* ou *ne se termine pas*)

4. Machines de Turing

Puissance et Efficacité

ND-machine de Turing

Même puissance que machine de Turing

A chaque ND-machine de Turing, il est possible de construire un machine de Turing qui a le même effet

Existence d'une machine de Turing interpréteur de ND-machines de Turing

Speedup exponentiel

Une ND-machine de Turing peut réaliser un speedup exponentiel

Mais impossibilité de réaliser ce speedup

Nécessité de simuler l'exécution non déterministe

4. Machines de Turing

Machine de Turing avec oracle

Machine avec oracle $A \subseteq N$

Ajout de 3 états spéciaux :

- oracle_{ask} : demander si l'entier représenté à droite de la tête de lecture appartient à l'ensemble A
- oracle_{yes} : l'entier appartient à A
- oracle_{no} : l'entier n'appartient pas à A

Forme d'une instruction oracle

$$< \text{oracle}_{ask}, c > \rightarrow < , \text{Mouv}, \text{new\_c}>$$

L'état suivant n'est pas spécifié dans l'instruction

L'état sera oracle_{yes} ou oracle_{no} en fonction de la réponse de l'oracle

4. Machines de Turing

Puissance

- Si A est récursif, même puissance que machine de Turing
- Si A n'est pas récursif, modèle plus puissant que les machines de Turing

  Mais impossibilité d'effectivement exécuter un tel programme !

Utilité du modèle avec oracle

Permet d'établir une hiérarchie parmi les problèmes indécidables (ensembles non récursifs)

Exemple:

Si K est récursif, quel problèmes seraient encore indécidables ?

4. Machines de Turing

Machine de Turing universelle

Objectif

Construire une machine de Turing qui soit un interpréteur de machines de Turing

Codage de machines de Turing

Il est possible de représenter, de coder une machine de Turing en utilisant les symboles :

- $\Sigma = \{ 0, 1 \}$

(Donnez un exemple de représentation possible)

Chaque MT va donc être encodée selon une chaîne de caractères (0,1)

Le code de la MT sera l'entier représenté par cet encodage

- Deux MT distinctes ont des encodages distincts
- L'ensemble des codes des machines de Turing est un ensemble (d'entiers) récursifs

4. Machines de Turing

Machine universelle

Il est possible de construire une machine de Turing interpréteur

- 3 rubans
  - Ruban 1: codage de la machine de Turing à interpréter
  - Ruban 2: donnée
  - Ruban 3: résultat intermédiaire de l'interpréteur

Application du théorème de Hoare-Allison

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

$$\exists z \, \forall n, x : \varphi_z(n,x) = \varphi_n(x)$$

- φ_n : fonction calculée par MT de code n
- φ_z : *fonction universelle* calculable
- MT_z : programme universel (interpréteur)

4. Machines de Turing

Problème de l'arrêt

On peut démontrer, en utilisant le modèle des machine de Turing que le problème de l'arrêt est non calculable

Même méthode de démonstration, mais adaptée aux machines de Turing

5. Fonctions récursives

Objectifs

- Modèle de calcul basé sur la définition mathématique de fonction
- fonctions $N^k \to N$
- Introduction du concept de récursivité
- Définition constructive : permet une évaluation, un calcul effectif de la valeur de la fonction en ses différents points
- Modèle de calcul effectif
- Gödel (1931,1934), Church (1935), Kleene (1935)

Classes de fonctions récursives

- Fonctions primitives récursives
  - que des fonctions totales
  - équivalent au langage BLOOP
- Fonctions (générales) récursives
  - toutes les fonctions calculables

5. Fonctions récursives

Fonctions de base

Fonctions constantes

$a: N^0 \rightarrow N$ avec $a() = a$

Nous écrirons simplement a

Exemples: 0, 8

Fonction successeur

$s: N \rightarrow N$ avec $s(n) = n+1$

Exemple:

- s(3) = 4

Fonctions de projection

$p^k : N^k \rightarrow N$ avec $p^{k} (x ,...,x ,...x) = x$

Exemple:

- $p^3{}_2(18, 5, 37) = 5$

5. Fonctions récursives

Composition

$h_1, h_2, \ldots, h_m : N^k \rightarrow N$

$g : N^m \rightarrow N$

$\overline{x} = x_1,\ldots,x_k$

$$f(\overline{x}) = g(\, h_1(\overline{x}), \ldots, h_m(\overline{x})\,)$$

Fonction f définie par composition des fonctions g et $h_1, h_2, \ldots, h_m$

$$f : N^k \rightarrow N$$

5. Fonctions récursives

Récursion primitive

$h : N^{k+2} \rightarrow N$
$g : N^{k} \rightarrow N$
$\bar{\mathbf{x}} = x_1,\ldots,x_k$

$f(\mathbf{x},0) = g(\ \mathbf{x}\ )$
$f(\mathbf{x},n+1) = h(\ \bar{\mathbf{x}},\ n\ ,\ f\ (\mathbf{x},n)\ )$

Fonction f définie par récursivité primitive

$f : N^{k+1} \rightarrow N$

- Si g et h sont calculables, alors f est calculables

Fonctions primitives récursives =

- fonctions de base
- fonctions obtenues par application de composition ou récursion primitive

5. Fonctions récursives

Exemples

Addition

add (m, 0) = m
add (m, n +1) = s (p^3_3 (m, n , add (m, n)))

ou plus simplement

add (m, 0) = m
add (m, n +1) = s (add (m, n))

Multiplication

mult (m, 0) = 0
mult (m, n +1) = add (m, mult (m, n))

Exponentiation

expo (m, 0) = 1
expo (m, n +1) = mult (m, expo(m, n))

5. Fonctions récursives

Exemples

Factorielle

fact (0) = 1
fact (n +1) = mult (s(n), fact(n))

Soustraction

pred (0) = 0
pred (n +1) = n
moins (m, 0) = m
moins (m, n +1) = pred (moins(m, n))

Comparaison

signe (0) = 0
signe (n +1) = 1
plus petit (m, n) = signe(moins(m, n))
egale (m, n) = moins(1, add(signe(moins(m, n), signe(moins(n, m)))

5. Fonctions récursives

Propriétés

Propriétés des fonctions primitives récursives

- Les fonctions primitives récursives sont *calculables*
- Les fonctions primitives récursives sont des fonctions totales
- Les fonctions primitives récursives sont équivalentes aux fonctions calculées par les programmes du langage BLOOP
- Il existe des fonctions totales calculables qui ne sont pas primitives récursives (Pourquoi ?)
- L'interpréteur des fonctions primitives récursives n'est pas une fonction primitive récursive (Pourquoi ?)
- Le modèle des fonctions primitives récursives n'est pas un modèle complet de la calculabilité

5. Fonctions récursives

Fonction d'Ackermann

Un exemple de fonction calculable non primitive récursive

ack (0, m) = m + 1
ack (n +1 , 0) = ack (n, 1)
ack (n +1 , m + 1) = ack (n, ack(n+1, m))

Fonction ayant une croissance plus rapide que n'importe quelle fonction primitiv
récursive.

Exemples

- ack (1, m) = m + 2
- ack (2, m) = 2 m + 3
- ack (3, m) = $2^{m+3} - 3$
- ack (4, 1) ≅ 64 000
- ack (4, 2) ≅ 10^{19200}
- ack (4, 3) ≅ _

5. Fonctions récursives

Minimisation

$h : N^{k+1} \rightarrow N$

$\bar{\mathbf{x}} = x_1, \ldots, x_k$

$f(\bar{\mathbf{x}}) = \mu\, n\ (\ h(\bar{\mathbf{x}}, n) = 0\ )$

- $\mu\, n$: le plus petit n tel que $h(\bar{\mathbf{x}}, n) = 0$
- $f : N^k \rightarrow N$
- Si h est calculable, f est calculable (mais pas nécessairement totale)

Fonctions récursives =

- fonctions de base
- fonctions définies par application de composition ou récursion primitive
- fonctions définies par minimisation

5. Fonctions récursives

Propriétés

Propriétés des fonctions récursives

- Les fonctions récursives sont *calculables*
- Toute fonction calculable est une fonction récursive
- Les fonctions récursives sont un modèle complet de la calculabilité