UCLouvain

École polytechnique de Louvain

LINFO1123

Calculabilité, Logique et Complexité

Yves Deville

Dylan Goffinet

2021-2022

Table des matières

1 | Concepts

1.1 Ensembles, langages, relations et fonctions

1.1.1 Ensembles

Un ensemble est une collection d'objets, sans répétition, appelés les éléments de l'ensemble.
Exemples : ensemble vide $= \emptyset$, ensemble fini $= \{1, 2, 3\}$, ensemble infini $= \{1, 2, 3, ...\}$, complément de $A = \bar{A}$.

1.1.2 Langages

Un langage est un ensemble de mots constitués de symboles d'un alphabet donné.
Exemple : palindromes sur $\{a, b\}$: $\epsilon, a, b, aa, aaa, baab, babab, ...$

Un alphabet Σ est un ensemble de symboles.
Exemples : $\Sigma = \{0, 1\}, \Sigma = \{a, b, c\}$

Une chaîne de caractères, ou mot, est une séquence finie de symboles juxtaposés.
Exemples : $010110, abccbcabcb, \epsilon$ (= chaîne de caratères vide)
L'ensemble de tous les mots de l'alphabet Σ s'écrit Σ^*
Exemples : $\Sigma = \{\} \rightarrow \Sigma^* = \{\epsilon\}, \Sigma = \{a\} \rightarrow \Sigma^* = \{\epsilon, a, aa, aaa, aaaa, ...\}$

1.1.3 Relations et fonctions

Une relation R sur A, B est un sous ensemble de AxB. En pratique, on dit que c'est une fonction.

Une fonction f (de A dans B) est une relation t.q. pour $a \in A$, il existe au plus un $b \in B$ t.q. $< a, b > \in f$.

Propriétés des fonctions

Soit $f : A \rightarrow B$:

- Domaine de f : $\text{dom}(f) = \{a \in A \mid f(a) \neq \perp\}$
- Image de f = $\text{im}(f) = \{b \in B \mid \exists a \in A : b = f(a)\}$
- f est fonction totale ssi $\text{dom}(f) = A$ ($\nexists a \in A : f(a) = \perp$)
- f est fonction partielle ssi $\text{dom}(f) \subseteq A$ (f totale est partielle mais f partielle n'est pas forcément totale)
- f est surjective ssi $\text{im}(f) = B$ (tout y a au moins un x)
- f est injective ssi $\forall a, a' \in A : a \neq a' \rightarrow f(a) \neq f(a')$ (tout x a un y différent)
- f est bijective ssi f est totale, injective et surjective (tout y a un et un seul x)

1.2 Ensembles énumérables

Un ensemble est énumérable s'il est soit fini ou s'il a le même cardinal [1] que $\mathbb{N}$ [2] (si on peut le mettre en bijection avec $\mathbb{N}$). En informatique, un programme est une chaîne finie de caractères $\rightarrow$ énumérable.

Propriétés

- Tout sous-ensemble d'un ensemble énumérable est énumérable
- L'union et l'intersection de deux ensembles énumérables est énumérable
- L'union d'une infinité énumérable d'ensemble énumérables est énumérable

1. Deux Ensembles A et B ont le même cardinal s'il existe une bijection entre A et B.
2. Pour rappel, l'ensemble $\mathbb{N}$ est l'ensemble des entiers positifs $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

1.2.1 Diagonalisation de Cantor

Soit $E = \{x \in \mathbb{R} \mid 0 < x \leq 1\}$. E est non énumérable [3]

Preuve :

1. Supposons E énumérable. Il existe donc une énumération des éléments de E : $x_0, ..., x_k, ...$

	1 digit	2 digit	3 digit	$\cdots$	$k+1$ digit	$\cdots$
x_0	x_{00}	x_{01}	x_{02}	$\cdots$	x_{0k}	$\cdots$
x_1	x_{10}	x_{11}	x_{12}	$\cdots$	x_{1k}	$\cdots$
x_2	x_{20}	x_{21}	x_{22}	$\cdots$	x_{2k}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\cdots$
x_k	x_{k0}	x_{k1}	x_{k2}	$\cdots$	x_{kk}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

FIGURE 1.1: Construire une table t.q. le nombre $x_k = 0, x_{k0}x_{k1}x_{k2}\ldots x_{kk}\ldots$

2. Prendre la diagonale ($d = 0, x_{00}x_{11}x_{22}\ldots x_{kk}\ldots$)
3. Modifier la diagonale t.q.
$$x'_{ii} = \begin{cases} 5 & \text{si } x_{ii} \neq 5 \\ 6 & \text{sinon} \end{cases}$$
Ce qui donne $d' = 0, x'_{00}x'_{11}x'_{22}\ldots x'_{kk}\ldots$ ($d' \in E$)
4. Contradiction : Comme E est énumérable, et que $d' \in E$, alors d' doit être dans l'énumération. Or, si $d' = x_p$, on a :
$$\begin{aligned} d' &= 0, x_{p0}x_{p1}x_{p2}\ldots x_{pp}\ldots \\ &= 0, x'_{p0}x'_{p1}x'_{p2}\ldots x'_{pp}\ldots \end{aligned}$$
5. Conclusion : E n'est pas énumérable

Avec ceci, on remarque qu'il existe des ensembles avec un cardinal plus grand que $\mathbb{N}$. Quelques un des ces ensembles sont : $\mathcal{P}(A)$, fonctions de $\mathbb{N} \to \mathbb{N}$, chaînes infinies sur un alphabet fini, etc. Le fait que les fonctions de $\mathbb{N} \to \mathbb{N}$ soient non énumérables (cfr. section 6.1) nous dit que l'ensemble des problèmes a donc un plus grand cardinal que l'ensemble des programmes (ayant le même cardinal que $\mathbb{N}$).

3. Il n'existe pas de bijection entre E et $\mathbb{N}$ c.à.d. qu'il y a beaucoup plus de réels entre 0 et 1 qu'il n'y a d'entiers positifs.

2 | Résultats fondamentaux

2.1 Programmes et fonctions

2.1.1 Algorithme

Un algorithme est une procédure qui peut être appliquée à n'importe quelle donnée (appartenant à une classe de données symboliques) et qui a pour effet de produire un résultat (appartenant à une classe de données symboliques). C'est un ensemble fini d'instructions (programme) calculant une fonction (algorithme $\neq$ fonction)

2.1.2 Univers des programmes

On se base sur un langage de programmation. Ex : les programmes Python (si une fonction est calculable, elle doit être calculable par une programme Python).

2.1.3 Univers des fonctions

On se limite aux fonctions de $\mathbb{N} \to \mathbb{N}$, et plus généralement de $\mathbb{N}^n \to \mathbb{N}$.

2.2 Fonction calculable

Une fonction $f : \mathbb{N} \to \mathbb{N}$ est calculable ssi il existe un programme Python qui, recevant comme données n'importe quel nombre naturel x, fourni comme résultat $f(x)$ s'il est défini, sinon $\perp$ (s'il ne se termine pas ou erreur).

Remarques

- Existence $\neq$ savoir l'écrire
- Si non calculable $\to$ aucun programme ne peut la calculer
- Généralisable pour $\mathbb{N}^n \to \mathbb{N}$

2.2.1 Ensemble récursif

On dit que l'ensemble A est récursif ssi il existe un programme qui prend en input x et qui renvoi (c.à.d un ensemble récursif est un ensemble pour lequel on est capable de dire si un élément y appartient) :

$$\begin{cases} 1 & \text{si } x \in A \\ 0 & \text{si } x \notin A \end{cases}$$

Le programme calcule donc une fonction totale. Exemple : $\{x \in \mathbb{N} \mid x \text{ pair}\}$.

2.2.2 Ensemble récursivement énumérable

On dit que l'ensemble A est récursivement énumérable ssi il existe un programme qui prend en input x et qui renvoi (tôt ou tard) :

$$\begin{cases} 1 & \text{si } x \in A \\ \text{Un autre résultat, ou ne se termine pas} & \text{si } x \notin A \end{cases}$$

2.2.3 Propriétés

- A récursif $\Rightarrow$ A récursivement énumérable
- A récursif $\Leftrightarrow$ $\bar{A}$ récursif
- A récursivement énumérable et $\bar{A}$ récursivement énumérable $\Leftrightarrow$ A récursif
- A fini ou $\bar{A}$ fini $\Rightarrow$ A et $\bar{A}$ récursif

2.3 Thèse de Church-Turing

1. Aucun modèle de la notion de fonction calculable n'est plus puissant que les Machines de Turing
2. Toute fonction calculable (au sens intuitif) est calculable par une machine de Turing
3. *Toutes les définitions formelles de la calculabilité connues à ce jour sont équivalentes* (Python = Oz= C en calculabilité)
4. Toutes les formulations de la calculabilité établies par la suite seront équivalentes aux définitions connues
5. *Une fonction est calculable s'il existe un programme d'ordinateur qui calcule cette fonction*

2.4 Numérotation

Soit P l'ensemble des programmes syntaxiquement corrects qui reçoivent une donnée entière et retournent un résultat entier.

— P est énumérable récursif (mais plus petit que l'ensemble des fonctions)
— $P = P_0, P_1, \ldots$ (sans répétition)
— P_k est le programme numéro k dans P
— φ_k est la fonction numéro k calculée par le programme P_k ($\varphi : \mathbb{N} \to \mathbb{N}$)

2.5 Calculabilité

2.5.1 Problème de l'arrêt

Soit la fonction $halt : P \times \mathbb{N} \to \mathbb{N}$ t.q. $halt(n, x) = \begin{cases} 1 & \text{si } \varphi_n(x) \neq \bot \\ 0 & \text{sinon} \end{cases}$

halt n'est pas calculable.

Preuve

Supposons *halt* calculable.

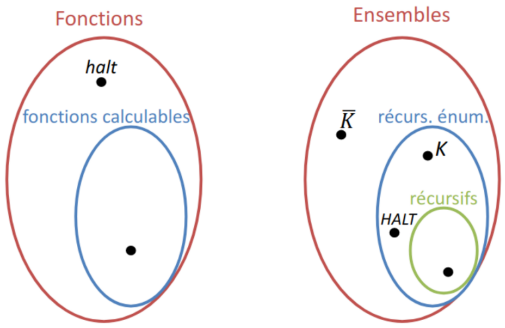
1. Construire la table

	0	1	2	$\cdots$	k	$\cdots$
P_0	$halt(0,0)$	$halt(0,1)$	$halt(0,2)$	$\cdots$	$halt(0,k)$	$\cdots$
P_1	$halt(1,0)$	$halt(1,1)$	$halt(1,2)$	$\cdots$	$halt(1,k)$	$\cdots$
P_2	$halt(2,0)$	$halt(2,1)$	$halt(2,2)$	$\cdots$	$halt(2,k)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\cdots$
P_k	$halt(k,0)$	$halt(k,1)$	$halt(k,2)$	$\cdots$	$halt(k,k)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

FIGURE 2.1: Table des valeurs de la fonction *halt*

2. Prendre la diagonale : $d(n) = halt(n, n)$
3. Modifier la diagonale :
$$d'(n) = \begin{cases} 1 & \text{si } halt(n,n) = 0 \\ \bot & \text{si } halt(n,n) = 1 \end{cases}$$
Si *halt* est calculable, alors d' est calculable. Soit P_d le programme qui calcule cette fonction.
4. Contradiction :
$$d'(d) = \begin{cases} 1 & \to halt(d,d) = 0 \to P_d \text{ ne se termine pas OR } d'(d) = 1 \\ \bot & \to halt(d,d) = 1 \to P_d \text{ se termine OR } d'(d) = \bot \end{cases}$$
5. Conclusion : d' n'est pas calculable, donc *halt* n'est pas calculable.

Soit deux ensembles non récursifs : $HALT = \{(nx)|halt(n,x) = 1\}$ et $K = \{n|(n,n) \in HALT\}$. K et $HALT$ sont récursivement énumérable, donc $\bar{K}$ et $H\bar{A}LT$ ne sont pas récursivement énumérable. Ainsi, on peut faire la différence entre les fonctions et les ensembles :



2.5.2 Hoare-Allison

Soit un langage Q (non trivial) qui a des programmes Q_k et qui ne calcule que des fonctions totales :

- La fonction φ'_k est calculée par le programme Q_k
- L'interpréteur $interpret(n,x)$ de ce langage Q est calculable (p.ex. en Python)
- La fonction $halt(n,x)$ pour ce langage Q est calculable (fonction constante qui vaut 1)
- $interpret(n,x)$ n'est pas calculable *dans* Q

Preuve

Supposons $interpret$ calculable dans Q.

1. Construire la table

	0	$\cdots$	k	$\cdots$
Q_0	$interpret(0,0)$	$\cdots$	$interpret(0,k)$	$\cdots$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\cdots$
Q_k	$interpret(k,0)$	$\cdots$	$interpret(k,k)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

FIGURE 2.2: Table des valeurs de la fonction $interpret$

2. Prendre la diagonale : $d(n) = interpret(n,n)$
3. Modifier la diagonale : $d'(n) = interpret(n,n) + 1$ (calculable dans Q si $interpret$ calculable dans Q)
4. Contradiction : $d'(d) = interpret(d,d) + 1$ OR $d'(d) = \varphi'(d) = interpret(d,d)$
5. Conclusion : $interpret(n,x)$ n'est pas calculable dans Q

Interprétation

Si un langage de programmation (non trivial) ne permet de calculer que des fonctions totales, alors :

- L'interpréteur de ce langage n'est pas calculable dans ce langage
- Il existe des fonctions totales non programmables dans ce langage
- Ce langage est dit restrictif

Conséquences

L'ensemble $\{n|\varphi_n \text{est totale}\}$ n'est pas récursif.

> Tout formalisme utilisé pour étudier la calculabilité doit permettre de définir son propre interpréteur.
>
> $$\exists\ z\ \forall n,x : \varphi_z(n,x) = \varphi_n(x)$$
>
> Avec φ_z la fonction calculée par l'interpréteur P_z

2.5.3 Rice

Deux formulations :

1. Soit $A \subseteq \mathbb{N}$, si A récursif, $A \neq \varnothing$ et $A \neq \mathbb{N}$ alors $\exists i \in A$ et $\exists j \in \bar{A}$ t.q. $\varphi_i = \varphi_j$
2. Si $\forall i \in A$ et $\forall j \in \bar{A}$: $\varphi_i \neq \varphi_j$ alors A non récursif ou $A = \varnothing$ ou $A = \mathbb{N}$

→ Aucun programme ne peut dire si une fonction respecte des spécifications

Si A $\neq \varnothing$ et $A \neq \mathbb{N}$, $\forall i \in A, \forall j \in \bar{A}$: $\varphi_i \neq \varphi_j$. Alors A non récursif.

Preuve

1. On suppose A récursif.
   On pose $P_k(x) \equiv$ while(True) (c.à.d. $\varphi_k = \perp$).
   Si $k \in \bar{A}$, comme $A \neq \varnothing \Rightarrow \exists m \in A$ et $\varphi_k \neq \varphi_m$
2. Construire *halt* :

$$halt(n,x) \equiv \begin{cases} \text{Construire le programme (sans l'exécuter) } P(z) \equiv P_n(x); P_m(z) \\ d = \text{ numéro du programme } P(z) \\ \text{if } d \in A \text{ then } print(1) \text{ Si } P_n \text{ se termine, } \varphi_d = \varphi_m \rightarrow d \in A \\ \text{else } print(0) \text{ Si } P_n \text{ ne se termine pas, } \varphi_d = \varphi_k \rightarrow d \in \bar{A} \end{cases}$$

3. *halt* n'est pas calculable, donc A est non récursif

Exemple d'application

Soit $A = \{i \mid \varphi_i \text{ est totale}\}$

— $A \neq \emptyset$: on peut écrire que $P_k \equiv$ print("Hello there")
$k \in A \rightarrow A \neq \emptyset$

— $A \neq \mathbb{N}$: on peut écrire que $P_d \equiv$ while $True$ {} doSomething() $d \in A \rightarrow A \neq \mathbb{N}$

— $\forall i \in A, \forall j \in \bar{A}$: φ_i est total tandis que φ_j ne l'est pas → pas égaux

Donc, A n'est pas récursif.

2.5.4 Paramétrisation

Si un programme $P(a,b)$ existe, alors il existe un programme $P'_b(a)$ (où b est fixé) t.q. $P'_b(a) \equiv Exec\ P(a,b)$

Forme S-1-1

Il existe une fonction totale calculable $S_1^1 : \mathbb{N}^2 \rightarrow \mathbb{N}$ t.q. $\forall k : \varphi_k(x_1, x_2) = \varphi_{S_1^1(k,x_2)}(x_1)$

Forme S-m-n

$\forall m, n \geq 0, \exists$ une fonction totale calculable $S_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ t.q. $\forall k : \varphi_k(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}) = \varphi_{S_1^1(k,x_{n+1},\ldots,x_{n+m})}(x_1, \ldots, x_n)$

2.5.5 Point fixe

Soit f une fonction totale calculable. Il existe k t.q. $\varphi_k = \varphi_f(k)$.

Preuve

On pose :

$$h(u,x) = \begin{cases} \varphi_{\varphi_u(u)}(c) & \text{si } \varphi_u(u) \neq \perp \\ \perp & \text{sinon} \end{cases} \tag{2.1}$$

Où $h(u,x)$ est calculable

$$h(u,x) = \varphi_{S(u)}(x) \tag{2.2}$$

Par application de $S-1-1$

$$g(x) = f(S(x)) \tag{2.3}$$

Où g est totale calculable (car f et S le sont),
et f est donné par k' : $\varphi_{k'}(x) = g(x) = f(S(x))$.

On a que k' est une constante par l'équation 2.2 :

$$h(k',x) = \varphi_{S(k')}(x)$$

Par l'équation 2.1 et comme $g = \varphi_{k'}$:

$$h(k',x) = \varphi_{k'(k')}(x)$$

Par l'équation 2.3, on a que $\varphi_{k'} = g(x) = f(S(x))$, donc :

$$h(k',x) = \varphi_{f(S(x))}(x)$$

Si on pose que $S(k') = k$, on obtient :

$$\varphi_k(x) = \varphi_{f(k)}(x)$$

Analyse

Pour tout transformateur de programme T, il existe deux programmes P_k et P_j t.q. :

- P_j est la transformation de P_k via T
- P_j et P_k calculent la même chose.

Exemple

$K = \{n \mid \varphi_n(n) \neq \perp\}$. Soient $f_n(x) = \perp\ \forall x$ et $f_m(x) = x\ \forall_x$

Définition de $f(x) = \begin{cases} n & \text{si } x \in K \\ m & \text{sinon} \end{cases}$ f est totale calculable si K récursif.

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

- Si $k \in K$
  - Par définition de f : $f(k) = n$
  - Par point fixe : $\varphi_k(k) = \varphi_n(k) \rightarrow$ Contradiction
- Si $k \notin K$
  - Par définition de f : $f(k) = m$
  - Par point fixe : $\varphi_k(k) = \varphi_m(k) \rightarrow$ Contradiction

3 | Modèles

3.1 Familles de modèles

1. Modèles basées sur le calcul :
   - Déterministes (une exécution)
   - Non-Déterministes (plusieurs exécutions possibles)
2. Modèles basés sur le langage

3.2 Langages de programmation

3.2.1 Définition

- Syntaxe
- Sémantique
- Convention de représentation d'une fonction par un programme

3.2.2 Langages complets

Langages tel que : Oz, Python, JS, C, etc. sont (en terme de calculabilité) équivalent. Ils permettent de calculer les mêmes fonctions.

3.2.3 BLOOP (Bounded Loop)

Un programme BLOOP est un programme Python tel que :

- Pas de boucle while
- Boucle for limité (pour que la boucle se termine quoi qu'il arrive)
- Pas de méthode récursive

Propriétés

- Tous les programmes BLOOP se terminent
- BLOOP ne calcule que des fonctions totales (mais ne les calcule pas toutes les Hoare-Allison)
- Il existe un compilateur des programmes BLOOP
- BLOOP n'est pas un modèle complet de la calculabilité

3.2.4 ND-Java

C'est un sous-ensemble de Java ("Non-Deterministic Java") On y ajoute fonction $choose(n)$ renvoyant un entier aléatoire entre 0 et n. Cette fonction est non-déterministe car à un même input elle ne renvoie pas toujours le même output.

3.2.5 ND-Récursif

A est ND-Récursif si $\exists$ un programme ND-Java t.q. s'il reçoit un input $x \in \mathbb{N}$:

- $x \in A$ alors $\exists$ une exécution qui retourne 1
- $x \notin A$ alors pour toute exécution le résultat est 0

ND-Récursif énumérable

Comme ND-Récursif sauf que le cas $x \notin A$ ne se fini pas forcément.

Propriétés

- Récursif $\Rightarrow$ ND-Récursif
- Récursif énumérable $\Rightarrow$ ND-Récursif énumérable

3.3 Automates Finis (FA)

Objectif : décider si un mot appartient ou non à un langage.

— Nombre fini d'états
— Lecture d'une donnée (mot : string)
— Chaque symbole lu *une fois*
— Transitions entre états en fonction du symbole
— État final = après avoir tout lu
— Pas de mémoire

3.3.1 Composition

— Σ : ensemble (fini) de symboles
— S : ensemble (fini) d'états
— $S_0 \in S$: état initial
— $A \subseteq S$: ensemble des états acceptants
— $\delta : Sx\Sigma \to S$: fonction de transition

3.3.2 Exemple

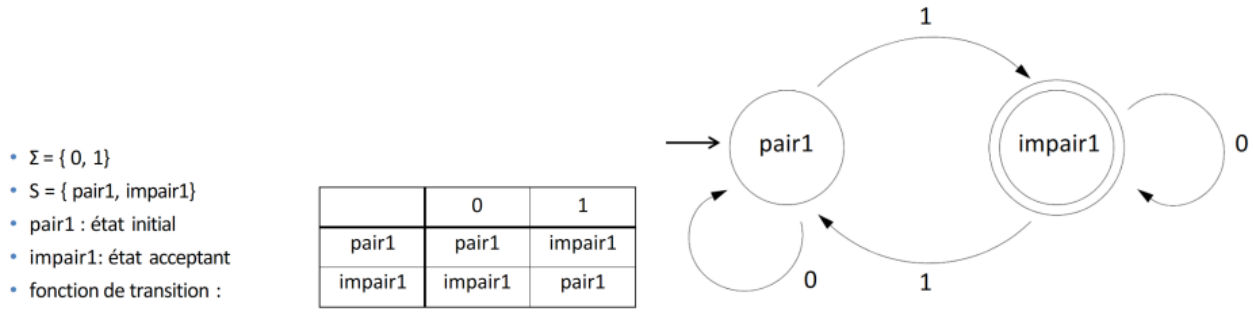


FIGURE 3.1: Exemple d'un automate fini

Un automate détermine un ensemble récursif $\{m \mid m$ accepté par le FA$\}$. En contrepartie, il ne peut pas programmer son interpréteur.

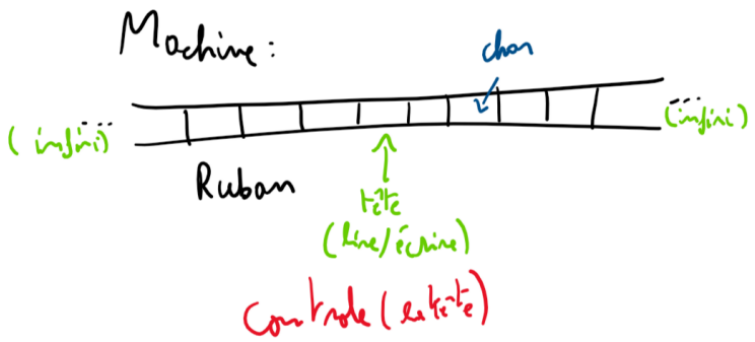
3.3.3 ND-FA

Non déterministe $\to \{m \mid$ s'il existe une exécution où m est accepté par le FA$\}$

ϵ-NDFA

Il peut y avoir des transitions vides.

3.4 Machines de Turing



- **Ruban** : infini des deux côtés (mais à un moment n de l'exécution, il y a un nombre fini de cases allouées)
- Tête : sur une seule case à la fois. Elle peut lire et écrire
- Contrôle : dirige la tête sur le ruban

3.4.1 États

- Début
- Arrêt (le ruban = le résultat)
- Autres (pendant l'exécution)

3.4.2 Déplacement

$$< q, c > \rightarrow < new\_q, Mouv, new\_c >$$

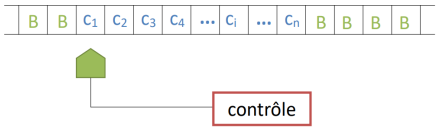
Avec :

- q : l'état courant ($\neq$ état marchant)
- c : symbole sous la tête de lecture
- new_c : symbole à écrire sous la tête de lecture (avant le mouvement)
- $Mouv$: mouvement (Droite ou Gauche) à faire (1 case à la fois)
- new_q : état suivant

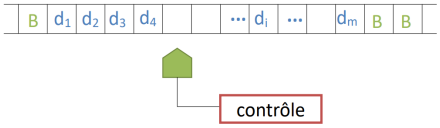
3.4.3 Modélisation

- Σ : ensemble fini de symboles d'entrée
- Γ : ensemble fini de symboles du ruban (ce qui peut être écrit sur le ruban)
  - $\Sigma \subset \Gamma$ (les symboles d'entrée peuvent être écrit sur le ruban)
  - $B \in \Gamma, B \notin \Sigma$ (symbole blanc = par défaut sur les cases vides)
- S : ensemble fini d'états
- $S_0 \in S$: état initial
- $stop \in \S$: état d'arrêt
- $\delta : Sx\Gamma \rightarrow Sx\{G, D\}x\Gamma$: fonction de transition (fini)

3.4.4 Exécution



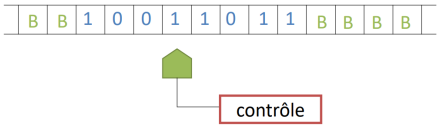
— Données sur le ruban $(c_1, c_2, c_3, ..., c_n)$
— Autres cases contiennent le symbole blanc (B)
— Tête de lecture initialement sur la première case de la donnée
— Tant que des instructions sont applicables, le contrôle applique les instructions
— L'exécution s'arrête dès que l'état devient *stop*
— S'il n'y a pas d'instruction applicable, il y a un arrêt
— Résultat : contenu du ruban à l'état *stop* $(d_1, d_2, d_3, ..., d_m)$



3.4.5 Exemple

Machine de Turing calculant la fonction $f(x) = x + 1$.

— Représentation des entier sous forme binaire
— $\Sigma = \{0, 1\}$
— $\Gamma = \{0, 1, B\}$



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

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

Positionner la 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 | Classes de complexité

4.1 Réduction et ensemble complet

But : on a un problème P,et on va le *déduire* à partir d'un problème P'.

4.2 Réduction algorithmique (calculabilité)

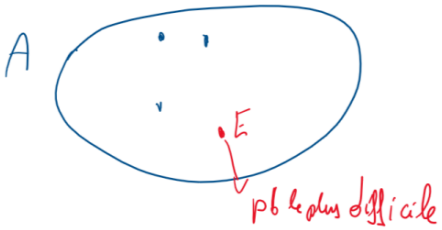
Un ensemble A est algorithmiquement réductible à un ensemble B ($A \leq_a B$) si en supposant B récursif, A est récursif.

Exemple :

Soit $P = \{n \mid \varphi_n$ renvoi un nombre pair $\}$: $HALT \leq_a P$ (si P énumérable, $HALT$ énumérable)

4.2.1 Ensemble complet

Étant donné une classe A de problèmes dont E est le problème le plus difficile :



On dit que le problème E est A-complet par rapport à une relation de réduction ($\leq_a$) ssi :

1. $E \in A$
2. $\forall B \in A : B \leq_a E$

On dit que le problème E est A-difficile par rapport à une relation de réduction ($\leq_a$) ssi :

1. $\forall B \in A : B \leq_a E$

4.2.2 Propriétés

- Si $A \leq_a B$ et B récursif, alors A récursif
- Si $A \leq_a B$ et B récursivement énumérable, alors A pas nécessairement énumérable
- Si $A \leq_a B$ et A non récursif, alors B non récursif
- $A \leq_a \bar{A}$
- $A \leq_a B \Leftrightarrow \bar{A} \leq_a B$
- Si A récursif, alors pour tout B, $A \leq_a B$

4.3 Réduction fonctionnelle (complexité)

Un ensemble A est fonctionnellement réductible à un ensemble B ($A \leq_r B$) ssi il existe une fonction totale calculable f t.q. :

$$a \in A \Leftrightarrow f(a) \in B$$

Pour décider si un élément appartient à A, il suffit de :

1. Calculer $f(a)$
2. Tester si $f(a) \in B$

4.3.1 Propriétés

- Si $A \leq_f B$ et B récursif, alors A récursif
- Si $A \leq_f B$ et B récursif énumérable, alors A récursif énumérable
- Si $A \leq_f B$ et A non récursif, alors B non récursif
- $A \leq_f B \leftrightarrow \bar{A} \leq_f \bar{B}$
- Si A récursif, alors pour tout B, $A \leq_f B$
- Pas nécessairement $A \leq_f \bar{A}$
- Si $A \leq_f B$, alors $A \leq_b B$ (l'inverse n'est pas nécessairement vrai)

4.4 Modèle de calcul

Tout les modèles de complexité ont entre eux des complexité spatiales et temporelles reliées de façon polynomiale.

4.4.1 Conséquence

Si un programme est faisable dans un modèle (non déterministe), alors il est pratiquement faisable dans tous les modèles (non déterministes). Concrètement, entre une MT et un programme Python, il y aura des différences de complexité mais elles resteront polynomiales.

4.5 Classes de complexité

- $DTIME(f)$: Ensemble récursif décidé par un programme Python en complexité temporelle $\mathcal{O}(f)$
- $DSPACE(f)$: Ensemble récursif décidé par un programme Python en complexité spatiale $\mathcal{O}(f)$
- $NTIME(f)$: Ensemble ND-récursif décidé par un programme Python en complexité temporelle $\mathcal{O}(f)$ (sur toutes les branches)
- $NSPACE(f)$: Ensemble ND-récursif décidé par un programme Python en complexité spatiale $\mathcal{O}(f)$ (sur toutes les branches)
- Classe P (Polynomiale) : $P = \bigcup_{i \geq 0} DTIME(n^i)$
- Classe NP (Non-Polynomiale) : $P = \bigcup_{i \geq 0} NTIME(n^i)$

4.6 Relations entre classes de complexité

4.6.1 Déterministe VS non déterministe

- Si $A \in NTIME(f)$ alors $A \in DTIME(c^f)$
- Si $A \in NSPACE(f)$ alors $A \in DSPACE(f^2)$
- $NPACE(f) = DSPACE(f)$

4.6.2 Time VS Space

- Si $A \in NTIME(f)$ alors $A \in NSPACE(f)$
- Si $A \in DTIME(f)$ alors $A \in DSPACE(f)$
- Si $A \in NSPACE(f)$ alors $A \in NTIME(c^f)$
- Si $A \in DSPACE(f)$ alors $A \in DTIME(c^f)$

4.7 NP-complétude

$$\text{Si } A \in P \ \rightarrow \ A \in NP \ ?$$

4.7.1 Réduction polynomiale

Un ensemble A est polynomialement réductible à un ensemble B ($A \leq_p B$) ssi il existe une fonction totale calculable f de complexité temporelle polynomiale t.q. :

$$a \in A \Leftrightarrow f(a) \in B$$

Si $A \leq_p B$ et $B \in P$ alors $A \in P$.

4.7.2 NP-complétude

Un problème E est NP-complet ($\leq_p$) ssi :

1. $E \in NP$
2. $\forall B \in NP : B \leq_p E$

Un problème E est NP-difficile ($\leq_p$) ssi :

1. $\forall B \in NP : B \leq_p E$

Propriétés

Soit E un ensemble NP-complet

- $E \in P$ ssi $P = NP$
- $E \notin P$ ssi $P \neq NP$
- Si $E \leq_p B$ et $B \in NP$, alors B est NP-complet

5 | Analyse et Perspectives

5.1 Fondement de la thèse de Church-Turing

Forme originale :

- Première partie : Toute fonction calculable par une machine de Turing est effectivement calculable
- Deuxième partie : Tout fonction effectivement calculable est calculable par une machine de Turing

Ici, en parlant de Machine de Turing, on parle d'un modèle de calculabilité (pourrait être Python par exemple).

5.2 Formalismes de calculabilité

Soit D un nouveau formalisme de calculabilité :

5.2.1 Caractéristiques de formalisme

- SD (Soudness des Descriptions) : toute fonction D-calculable est calculable
- CD (Complétude des Définitions) : toute fonction calculable est D-calculable
- SA (Soudness Algorithmique) : l'interpréteur de D est calculable
- CA (Complétude Algorithmique) : si $p \in P$ (P est par exemple Java), que $p' \in D$ et que $p \equiv p'$ (calculent la même fonction), alors équivalence des formalismes
- U (Description Universelle) : l'interpréteur de D est D-calculable
- S ($S-m-n$ Affaiblie) : $\forall d \in D\ \exists S : d(x,y) = [S(x)](y)$

5.2.2 Propriétés

- $SA \Rightarrow SD$
- $CA \Rightarrow CD$
- SD et $U \Rightarrow SA$
- CD et $S \Rightarrow CA$
- SA et $CD \Rightarrow U$
- CA et $SD \Rightarrow S$
- S et $U \Rightarrow S-m-n$
- SA et $CA \Leftrightarrow SD$ et CD et U et S
- SA et CD et $S \Leftrightarrow CA$ et SD et U

Exemple : BLOOP est SD, SA, et S mais pas le reste.

6 | Preuve supplémentaire

6.1 L'ensemble des fonctions totales n'est pas énumérable

Soit F l'ensemble des fonctions totales telles que $f : \mathbb{N} \to \mathbb{N}$.

F est non énumérable.

Preuve

Supposons F énumérable. Il existe donc une énumération des éléments de F : $f_0(0), f_1(0), \ldots$.

1. Construire la table

	1	1	2	$\cdots$	k	$\cdots$
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$\cdots$	$f_0(k)$	$\cdots$
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$\cdots$	$f_1(k)$	$\cdots$
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$\cdots$	$f_2(k)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\cdots$
f_k	$f_k(0)$	$f_k(1)$	$f_k(2)$	$\cdots$	$f_k(k)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

FIGURE 6.1: Table des résultats de la fonction f

2. Prendre la diagonale d qui est aussi une fonction de $\mathbb{N} \to \mathbb{N}$ ($d \in F$)
3. Modifier la diagonale pour obtenir d' t.q. :

$$f'_i(j) = \begin{cases} 5 & \text{si } f_i(j) \neq 5 \\ 6 & \text{sinon} \end{cases}$$

   Où $f_i(j)$ est le résultat de la fonction f avec le numéro i pour la donnée j.
4. Contradiction :
   Comme F est énumérable et que $d' \in F$, alors d' doit être dans l'énumération. Or si d' a le numéro p on a :

$$\begin{aligned} d' &= f_p(0), f_p(1), f_p(2), \ldots, f_p(p), \ldots \\ &= f'_p(0), f'_p(1), f'_p(2), \ldots, f'_p(p), \ldots \end{aligned}$$

   Si $f_p(0)$ vaut 5, on a $(f_p(0) = 5) \neq (f'_p(0) = 6)$.
5. Conclusion :
   F n'est pas énumérable.

7 | Exemples de QCM/Vrai ou Faux

- Soit A un problème appartenant à $DTIME(n^2)$:
  - $A \in P$
  - $A \in DSPACE(n^2)$
  - Il peut exister un programme Java de complexité $\mathcal{O}(n)$ qui décide A
  - $A \in DTIME(n^3)$
- Si une fonction f est calculable, alors toute fonction g dont f est une extension est calculable : Faux
- Si le domaine d'une fonction est fini, alors cette fonction est calculable : Vrai
- Si SAT $\leq_a A$ et $A \in P$, alors SAT $\in P$: Faux
- L'ensemble des sous-ensembles récursivement énumérables de $\mathbb{N}$ est énumérable : Vrai
- Tout ensemble de paires d'entiers est récursif : Faux
- L'ensemble des programmes Java calculant une fonction f telle que $f(10) = 10$ est un ensemble récursif : Faux
- Un sous-ensemble infini d'un ensemble récursivement énumérable est récursivement énumérable : Faux
- Soit D un nouveau modèle de calculabilité. Si toute fonction calculable est calculable dans D et si toute fonction calculable dans D est effectivement calculable, alors D est un modèle complet de la calculabilité : Faux
- Un algorithme donné ne calcule qu'une et une seule fonction : Vrai
- Un sous-ensemble d'un ensemble récursivement énumérable est récursivement énumérable : Faux
- Un problème qui peut être résolu par un algorithme (de complexité) exponentiel est pratiquement infaisable (intrinsèquement complexe) : Faux
- Le complément d'un ensemble récursivement énumérable est récursivement énumérable : Faux
- Soient les programmes $P_{32}(n)$ dont le code est « print(1) » et $P_{57}(n)$ dont le code est « print(0) » :
  - Les fonctions phi_{32} et phi_{57} sont des fonctions constantes
- Les propriétés SA, CD et S sont suffisantes pour que D soit un modèle complet de la calculabilité : Vrai
- Un ensemble est f-réductible (fonctionnellement réductible) à son complément : Faux
- Si A peut être décidé par un algorithme polynomial et si B est f-réductible à A, alors B peut être décidé par un algorithme polynomial : Faux
- Tout ensemble récursivement énumérable est a-réductible à HALT : Vrai
- Un ensemble énumérable est récursif : Faux
- Si A est un sous-ensemble (strict et non vide) récursif de programmes Java, alors toute fonction calculée par un programme de A est aussi calculée par un programme du complément de A : Faux
- Soit A est un ensemble (infini) récursivement énumérable. Si B $\subseteq$ A, alors B est aussi récursivement énumérable : Faux
- Il existe un ensemble infini de chaînes finies de caractères (A-Z) qui est non récursivement énumérable : Vrai
- Une extension d'une fonction partielle calculable est toujours calculable : Faux
- Le modèle de calcul BLOOP possède les propriétés SD et SA : Vrai
- L'union d'une infinité énumérable d'ensembles récursivement énumérable est récursivement énumérable : Faux
- Pour déterminer si un problème A est NP-complet, il suffit de déterminer que A est polynomialement réductible un problème NP-complet connu (e.g. SAT) : Faux
- Il existe des ensembles récursifs qui ne sont pas récursivement énumérables : Faux
- Un sous-ensemble infini d'un ensemble récursif est récursif : Faux