2. Variantes des machines de Turing

Bruno Grenet

Université Grenoble Alpes – IM²AG
L3 Informatique
UE Modèles de calcul – Machines de Turing

https://membres-ljk.imag.fr/Bruno.Grenet/MCAL-MT.html

Introduction

Rappels

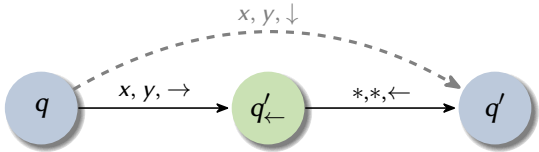
- Une machine de turing est un quadruplet $\mathcal{M} = (Q, \Sigma, q_0, \delta)$ où
  - Q est un ensemble fini d'états *de taille quelconque*
  - Σ est *n'importe quel* alphabet fini, mais doit contenir $\square$
  - δ est une *fonction partielle* : $\delta(q, x)$ peut être indéfini
- Ruban unidimensionnel, non borné ni vers la gauche ni vers la droite

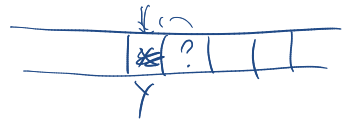
- Est-ce que ces choix dans la définition sont importants ?
- Qu'est-ce qui compte vraiment ? qu'est-ce qu'on peut modifier ?
- Y a-t-il d'autres possibilités pour définir une machine ?

Premières remarques

Déplacements

- À chaque étape, la tête doit aller à gauche ou à droite
- On peut rajouter l'option « rester sur place » ($\downarrow$)
- Définition *équivalente* car on peut *simuler* « rester sur place » :





$* =$ n'importe quel symbole

- On peut aussi séparer les instructions en *écriture* et *déplacement*

Fonction *partielle* ?

- On peut rajouter un état « arrêt »
- Pour chaque état q et chaque symbole x, $\delta(q, x)$ est défini
- La machine s'arrête dès qu'elle atteint l'état « arrêt »

Contents

1. Taille de l'alphabet

2. Mémoire bornée ou non bornée ?

3. Machines à plusieurs rubans

4. Machines non déterministes

Contents

1. Taille de l'alphabet

2. Mémoire bornée ou non bornée ?

3. Machines à plusieurs rubans

4. Machines non déterministes

Encodage des données

Exemples de programmes en C

- Manipulation de `floats` (32 bits) ou `doubles` (64 bits)
- Manipulation d'`ints`, `unsigned int`, `longs`, …
- Opérations sur des octets
- Opérations bit-à-bit
- …

$\rightarrow$ on ne manipule pas forcément directement les bits mais on pourrait !

Et sur une machine de Turing ?

- Définition générale : alphabet Σ quelconque
- *Besoin* d'un symbole spécial $\square$ car ruban non borné
- On peut *encoder* n'importe quel alphabet en binaire
  - un *encodage* $\langle\cdot\rangle : \Sigma \setminus \square \to \{0,1\}^\ell$ est une fonction *injective* : $\langle u\rangle = \langle v\rangle \iff u = v$
  - Extension aux mots : $\langle w_{[0]} w_{[1]} \cdots w_{[n-1]}\rangle = \langle w_{[0]}\rangle\langle w_{[1]}\rangle \cdots \langle w_{[n-1]}\rangle$

Restriction à un alphabet binaire

Théorème

Soit $\mathcal{M} = (Q, \Sigma, q_0, \delta)$ une machine de Turing. Il existe une machine $\mathcal{M}' = (Q', \{0, 1, \square\}, q_0, \delta')$ et un *encodage* $\langle\cdot\rangle : \Sigma \setminus \{\square\} \to \{0,1\}^\ell$ tels que

- $Q \subset Q'$ — Q' *contient* Q
- si $\mathcal{M}(w) = (q, w', i)$, alors $\mathcal{M}'(\langle w \rangle) = (q, \langle w' \rangle, i \times \ell)$ — *même* configuration finale
- si $\mathcal{M}(w) \uparrow$, alors $\mathcal{M}'(\langle w \rangle) \uparrow$

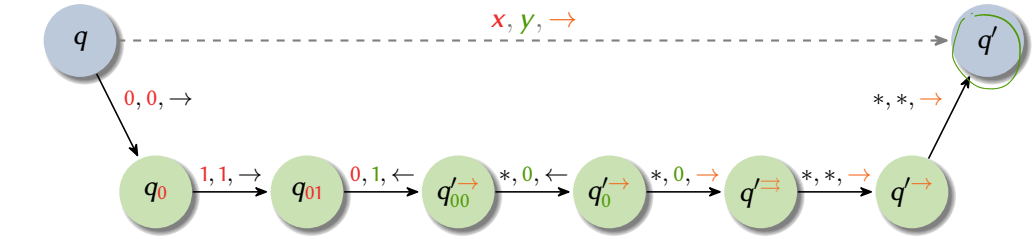
Idée de la preuve

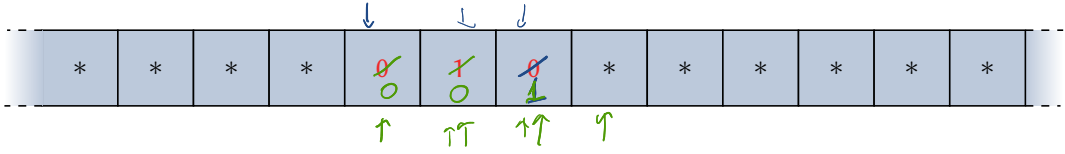
- Pour chaque symbole $x \in \Sigma \setminus \square$: code *unique* $\langle x \rangle$ sur ℓ bits — $2^\ell \geq \#\Sigma - 1$
- Pour chaque transition, suite de transitions pour :
  - lire ℓ bits à la suite
  - écrire ℓ bits à la suite

  (nouveaux états *intermédiaires*)

Idée de la preuve : 1er dessin

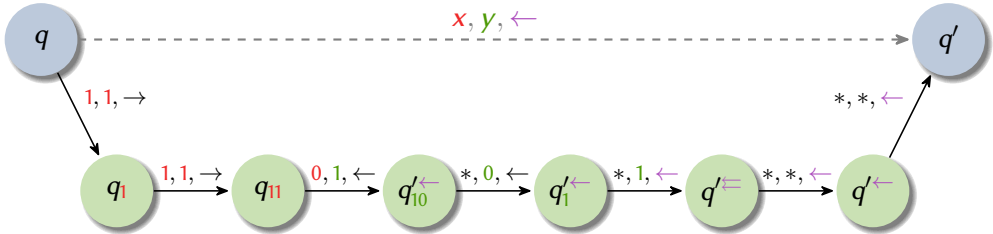
$\langle x \rangle = 010$
$\langle y \rangle = 001$
$* = 0$ ou 1





Idée de la preuve : 2ème dessin

$\langle x \rangle = 110$
$\langle y \rangle = 101$
$* = 0$ ou 1



Bilan sur la taille d'alphabet

Résumé

- Toute machine de Turing peut être *simulée* par une machine d'alphabet $\{0, 1, \square\}$
- On peut même se restreindre à $\{1, \square\}$
- Remarque : cela se fait au prix d'une augmentation du nombre d'états

Inversement

- On peut *grouper* ℓ cases ensemble
  - Nouvel alphabet Σ^ℓ
  - Il suffit d'adapter la table de transition pour traiter ℓ symboles d'un coup

Morale

- On peut choisir la taille d'alphabet qui nous arrange
  - pour décrire une machine, on peut utiliser beaucoup de symboles
  - pour raisonner abstraitement sur une machine, on peut supposer $\Sigma = \{0, 1, \square\}$

$\rightarrow$ la taille d'alphabet n'a aucune importance !

Et le nombre d'états ?

Diminuer le nombre d'états

- Ajout de symboles du type $x^q \rightarrow$ état et position de la tête
- Nombre fixé d'états pour simuler une transition

$\rightarrow$ Toute machine de Turing peut être *simulée* par un machine ayant (par exemple) 5 états

Morale

- On peut choisir le nombre d'états
  - en utiliser beaucoup si ça nous arrange
  - supposer qu'elle en a peu si c'est plus simple

$\rightarrow$ le nombre d'états n'a aucune importance !

Interprétation

- Deux mémoires : ruban (non bornée) ; états (bornée)
- Compromis : transfert d'une quantité *bornée* de mémoire entre ruban et états
- Si besoin, aucun souci à utiliser beaucoup d'états **et** beaucoup de symboles !

Contents

1. Taille de l'alphabet

2. Mémoire bornée ou non bornée ?

3. Machines à plusieurs rubans

4. Machines non déterministes

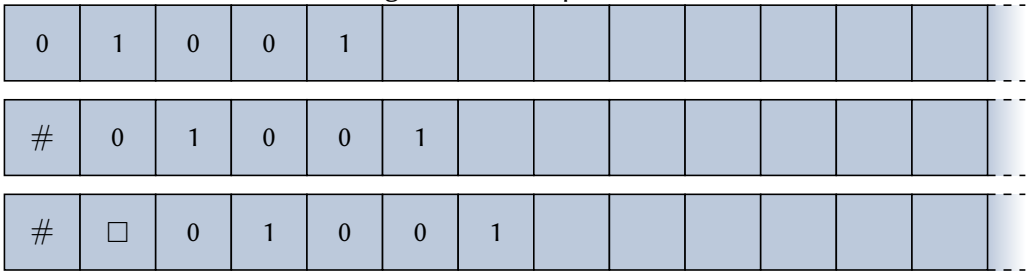
Ruban semi-borné

Théorème

Tout langage $L(\mathcal{M})$ reconnu par une machine de Turing $\mathcal{M}$ peut être reconnu par une machine $\mathcal{M}'$ dont le ruban est *borné à gauche* : $\mathcal{M}'$ n'utilise que des cases du ruban situées à droite de sa case de départ.

Idée de la preuve

- Initialement, *décaler w* pour libérer une case à gauche et écrire # dans cette case
- Quand on lit #, recommencer l'opération de décalage
- Les autres transitions sont gardées telles qu'elles



Ruban borné

Théorème

Soit $k \in \mathbb{N}$ et $\mathcal{M}$ une machine de Turing qui, sur toute entrée w, utilise $\leq k$ cases de son ruban. Alors $L(\mathcal{M})$ est régulier (reconnaissable par un automate fini)

Preuve

- $\mathcal{M} = (Q, \Sigma, q_0, \delta)$ avec $\#\Sigma = s$
- $\#\text{configurations} = \#Q \times s^k \times k$
- On crée un automate fini $\mathcal{A}$
  - états = configurations de $\mathcal{M}$
  - $(q, w, i) \xrightarrow{a} (q', w', i')$ ssi $\begin{cases} (q, w, i) \vdash (q', w', i') \\ w_{[i]} = a \end{cases}$

Bilan sur la mémoire bornée

Mémoire nécessaire

- Mémoire bornée insuffisante : automates finis
- La *forme* de la mémoire non bornée n'a pas d'importance
  - autre exemple : ruban bidimensionnel ou multidimensionnel

Et les automates à pile ?

- Rappel : automate avec une pile…
- Moins *puissant* que les machines de Turing : par ex. reconnaître $a^n b^n c^n$
- Mais avec deux piles : pareil qu'une machine de Turing

Contents

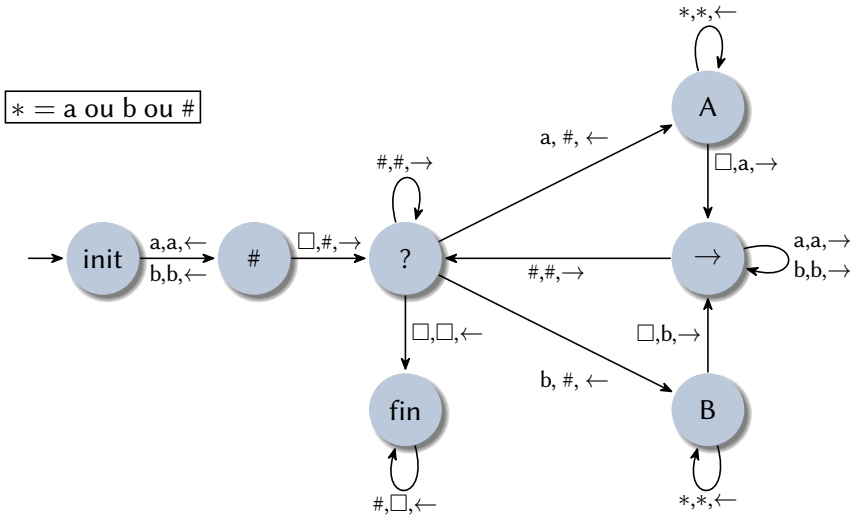
1. Taille de l'alphabet

2. Mémoire bornée ou non bornée ?

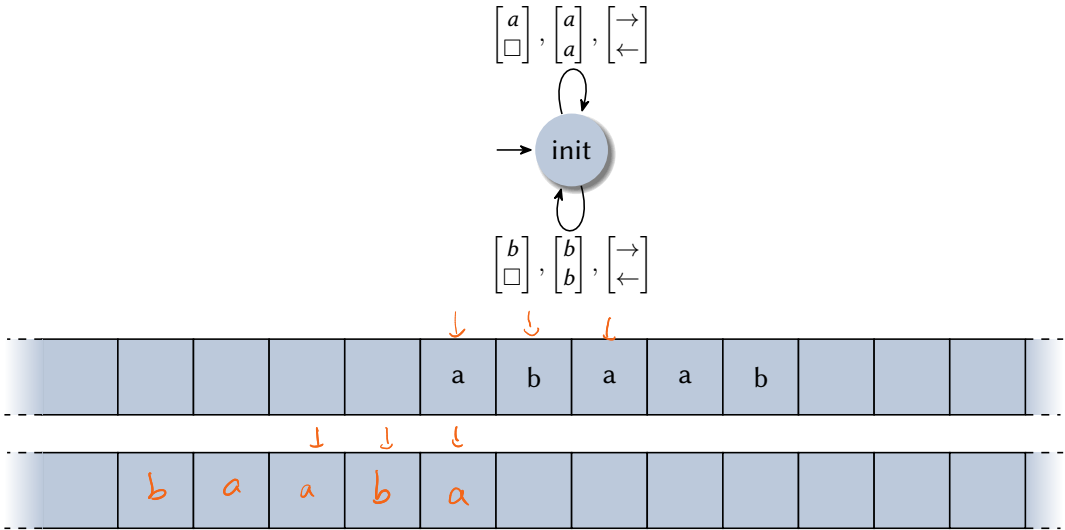
3. Machines à plusieurs rubans

4. Machines non déterministes

Exemple : calcul du miroir



Calcul du miroir à 2 rubans



Définition d'une machine à k rubans

Définition

Une machine de Turing à k rubans est un quadruplet $\mathcal{M} = (Q, \Sigma, q_0, \delta)$ où

- Q est un ensemble fini d'*états*
- Σ est l'*alphabet* contenant le blanc $\square$
- $q_0 \in Q$ est l'*état initial*
- $\delta : Q \times \Sigma^k \to Q \times \Sigma^k \times \{\rightarrow, \leftarrow\}^k$ est la *table de transition*

Remarque

- Même définition, sauf pour δ qui gère les k rubans à la fois
- Mêmes règles de fonctionnement (adaptées)
- Configuration initiale :
  - Mot w sur le 1er ruban, autres rubans vides
  - Fonction à t entrées : entrées sur les rubans 1 à t
- Calcul de fonction : *choix* du ou des rubans de sortie

Équivalence des deux modèles

Théorème

Une machine de Turing $\mathcal{M}^{(k)}$ à k rubans peut être *simulée* par une machine de Turing *classique* $\mathcal{M}$:

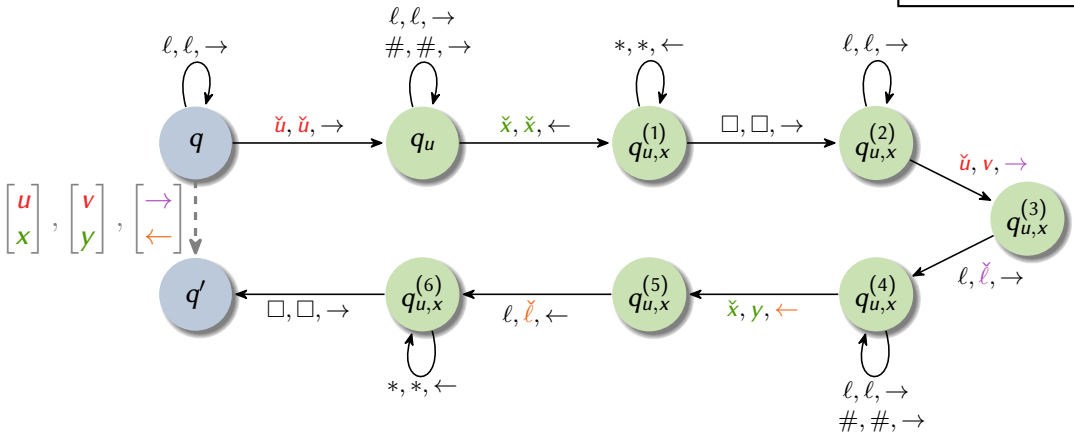
- si $\mathcal{M}^{(k)}$ *reconnaît* un langage L, on peut trouver $\mathcal{M}$ t.q. $L = L(\mathcal{M})$;
- si $\mathcal{M}^{(k)}$ *calcule* une fonction f, on peut trouver $\mathcal{M}$ t.q. $f(w) = f_{\mathcal{M}}(w)$ pour tout w

Idée de la preuve

- *Encodage* de k rubans sur un seul — ex.: à la suite, séparés par #
- Nouveaux symboles $\check{x}$ pour chaque $x \in \Sigma$ — position des têtes de lecture
- Simulation d'une transition :
  - Parcourir le ruban et retenir les lettres lues — nom des états
  - Re-parcourir le ruban pour appliquer la transition

Idée de la preuve : un dessin simplifié (!)

$\ell \in \Sigma$
$* \in \Sigma \sqcup \check{\Sigma} \sqcup \{\#\}$



Bilan sur les machines à k rubans

Vers une preuve complète (cas $k = 2$)

- Définir exactement la machine $\mathcal{M}$:
  - Nouvel alphabet : $\Sigma \sqcup \check{\Sigma} \sqcup \{\#\}$
  - États : pour $q \in Q$, $x_1, x_2 \in \Sigma \to q$, q_{x_1}, $q^{(1)}_{x_1,x_2}$, …, $q^{(6)}_{x_1,x_2}$ + états pour les décalages
  - Transitions : chaque transition donne lieu à $O(\#\Sigma)$ transitions
- Preuve (très) technique mais idée générale assez simple
  - cas de k rubans similaire, mais encore plus technique !

Morale

- Les machines à 1, 2, ou 1000 rubans sont équivalentes
- On peut choisir le *modèle* qui nous arrange
  - pour décrire une machine, on peut s'autoriser plusieurs rubans
  - pour raisonner abstraitement sur une machine, on peut supposer qu'elle n'en a qu'un

Contents

1. Taille de l'alphabet

2. Mémoire bornée ou non bornée ?

3. Machines à plusieurs rubans

4. Machines non déterministes

Machine non déterministe

Définition

- Pour chaque état $q \in Q$ et lettre $x \in \Sigma$, $\delta(q, x)$ est un ensemble de triplets $(q', y, \leftrightarrow)$
- À chaque étape de calcul, la machine *choisit de manière non-déterministe* un triplet
- Une machine de Turing non déterministe $\mathcal{N}$ reconnaît un langage L si
  - $\forall w \in L$, il existe des choix tels que $\mathcal{N}$ termine dans l'état « oui »
  - $\forall w \notin L$, aucun choix ne permet à $\mathcal{N}$ de terminer dans l'état « oui »

Théorème

Tout langage reconnu par une machine de Turing non déterministe $\mathcal{N}$ peut être reconnu par une machine de Turing classique $\mathcal{D}$ (déterministe)

Idée de la preuve

- Simuler *toutes* les exécutions possibles *tous les choix possibles*

Machine non déterministe

Déf

- ▶
- ▶
- ▶

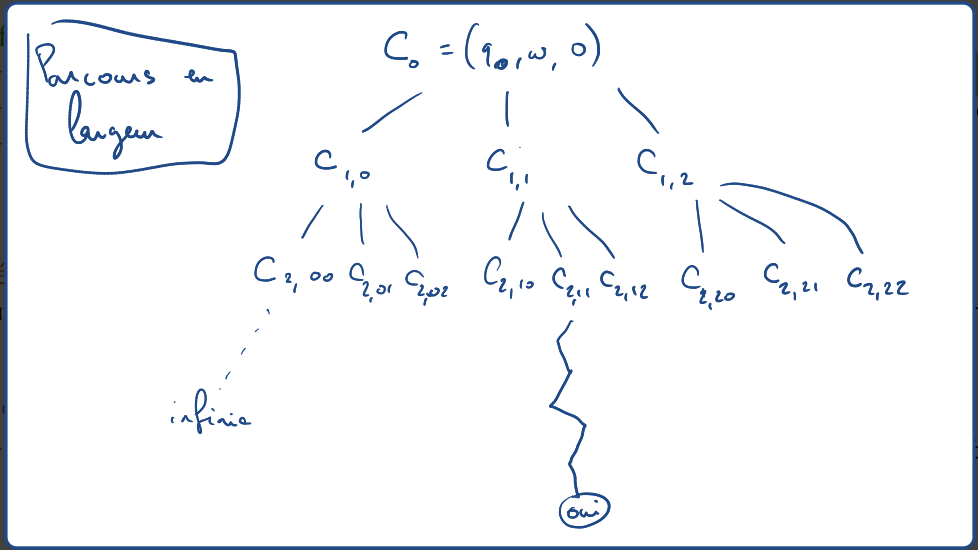
Thé

Tou

par

Idé

- ▶



Machine non déterministe

Définition

- Pour chaque état $q \in Q$ et lettre $x \in \Sigma$, $\delta(q, x)$ est un ensemble de triplets $(q', y, \leftrightarrow)$
- À chaque étape de calcul, la machine *choisit de manière non-déterministe* un triplet
- Une machine de Turing non déterministe $\mathcal{N}$ reconnaît un langage L si
  - $\forall w \in L$, il existe des choix tels que $\mathcal{N}$ termine dans l'état « oui »
  - $\forall w \notin L$, aucun choix ne permet à $\mathcal{N}$ de terminer dans l'état « oui »

Théorème

Tout langage reconnu par une machine de Turing non déterministe $\mathcal{N}$ peut être reconnu par une machine de Turing classique $\mathcal{D}$ (déterministe)

Idée de la preuve

- Simuler *toutes* les exécutions possibles — *tous les choix possibles*
- Machine $\mathcal{D}$ à trois rubans :
  1. Lecture seule : w, jamais modifié
  2. Chemin vers le nœud courant de l'arbre — incrément en base b
  3. Ruban de $\mathcal{N}$ — copie de w puis simulation

Conclusion

- Les machines de Turing sont un modèle de calcul très **robuste** :
  - les détails ne changent rien au modèle…
  - … tant qu'on garde les caractéristiques importantes

Ce qui est important

- Avoir une mémoire non bornée
- Pouvoir l'exploiter complètement

mémoire bornée = automate fini
ex. des automates à pile

Ce qui ne change rien

- L'alphabet utilisé (≥ 2 symboles)
- Le nombre d'états
- La *forme* de la mémoire non bornée
  - Uni- ou multidimensionnelle
  - Bornée à une extrémité ou non
  - Un ou plusieurs rubans
- Les opérations élémentaires exactes
  - Déplacement de la tête
  - Nombre de symboles lus/écrits
  - Séparation écriture / déplacement
- Les conditions d'arrêt de la machine
  - État particulier
  - Absence de transition possible