

MIF15 – Calculabilité & complexité

Sylvain Brandel

2014-2015

sylvain.brandel@univ-lyon1.fr

Fonctionnement

- **Nouveauté 2014 !**
- (tentative de) « classe inversée »
- Supports fournis en avance
 - Supports de cours
 - Ce qui est projeté en CM (dans la mesure du possible)
 - Sujets de TD
- CM
 - Compléments du support fourni
- TD
 - Plus une discussion autour des solutions que vous aurez cherchées avant de venir
- Bref, travail de votre part **AVANT**

Fonctionnement

- Cours condensés sur un demi semestre
 - 1^{ère} moitié : MIF15, de maintenant au 24 octobre
 - 2^{ème} moitié : MIF19, du 27 octobre à la fin du semestre
- Du coup double dose
 - Chaque semaine **deux** CM et **deux** TD
- Emploi du temps
 - CM
 - Jeudi 14H – 15H30
 - Vendredi **8H** – 9H30
 - TD
 - Jeudi **16H** – 17H30
 - Vendredi **10H** – 11H30

Evaluation

- <http://liris.cnrs.fr/sylvain.brandel/teaching/MIF15>
- Pas CCI (contrôle continu intégral)
 - donc CC puis examen (2 sessions)
- Contrôle continu
 - 2 ou 3 contrôles
 - Surprise ou pas
 - Dates précisées au moins une semaine avant
 - Un contrôle tous ensembles qui sera planifié
 - Un contrôle de 15 minutes pèsera 4x moins qu'un contrôle d'une heure
 - Total : 33,3333333333333333%
- Examen
 - Session 1 : **vendredi 24 octobre** 8H
 - Session 2 : en mars, avec tous les autres examens
 - 66,6666666666666666%

Projet

- Pas de TP, pas de projet
- Mais ...
- Jetez un coup d'œil sur JFLAP (cf. Google)
 - Plateforme de test d'un cours
 - Duke University, Trinity, Caroline du Nord, Etats-Unis
 - Pas tout récent
 - Pour tester des machines de Turing, regarder comment elles s'exécutent ...
 - Je vous montrerai

De votre côté

- Travail personnel conséquent
- Se préparer à l'avance
- Ne pas attendre que les réponses viennent toutes seules
- Lisez vos mails ...

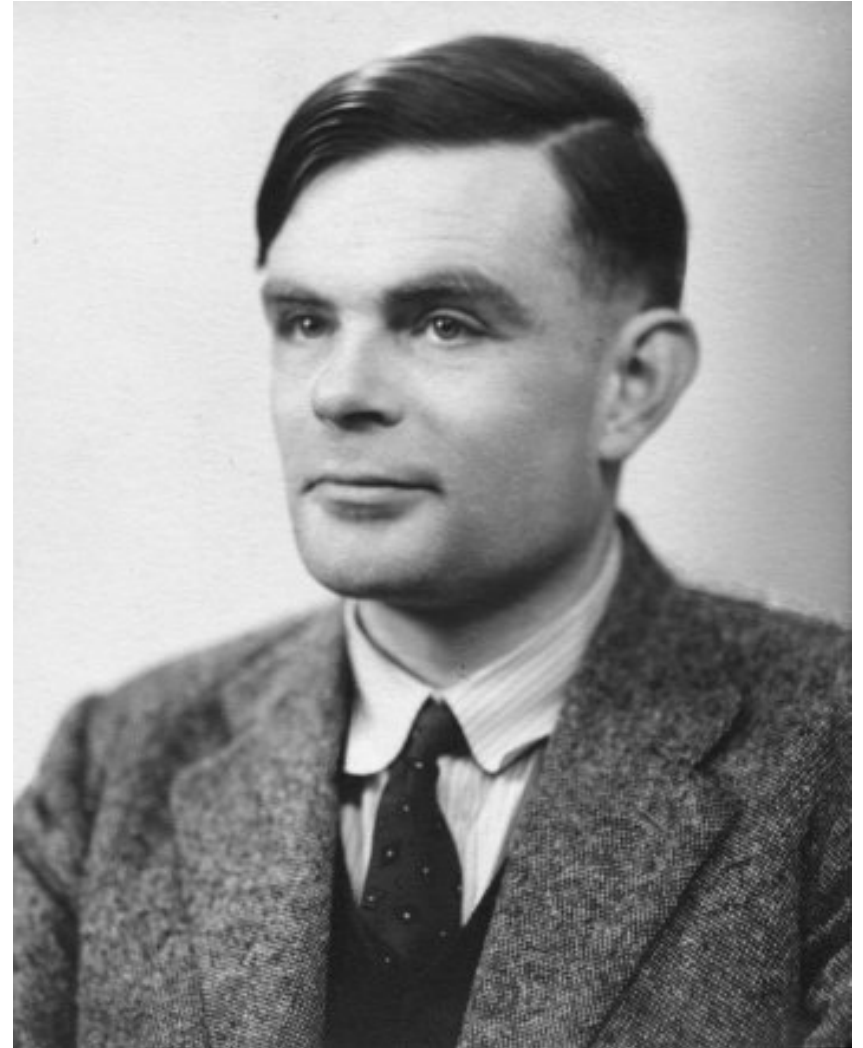
MIF15 – Calculabilité & complexité

Sylvain Brandel

2014 – 2015

sylvain.brandel@univ-lyon1.fr

INTRODUCTION



Motivations

- Informatique fondamentale
- Historiquement
 - Théorie de l'incomplétude
 - Que peut-on calculer avec un algorithme ?
- Lien avec les langages de programmation
 - Ce cours prépare à deux cours de master
 - Calculabilité et complexité
 - Compilation
- Vous intéresser ...
 - Si on sait qu'un problème est indécidable, inutile de chercher un algorithme pour le résoudre

Programme

- Classifier des langages


Exemple d'école	Classe de langage	Reconnu par	Engendré par
a^*b^*	langages rationnels	automates à états finis	grammaire régulière
$\{a^n b^n \mid n \geq 0\}$	langages algébriques	automates à pile	grammaire algébrique
$\{a^n b^n c^n \mid n \geq 0\}$	langages rékursifs	machine de Turing	grammaire (générale)

(les deux premières classes ont été vues en LIF15 – L3)

- La décidabilité et la complexité en découlent

Programme

- Notions mathématiques de base
 - Ensembles
 - Alphabets, langages, expressions régulières
- Automates à états finis
 - Déterministes ou non
 - Liens avec les expressions rationnelles
 - Rationalité
 - Minimisation
- Langages algébriques
 - Grammaires algébriques
 - Automates à pile
 - Algébricité



LIF15 (L3)

Programme (suite)

- Machines de Turing
 - Formalisme de base
 - Langages rékursifs
 - Extensions
 - Machine de Turing Universelle
 - Grammaires
- Indécidabilité
 - Thèse de Church – Turing
 - Problèmes indécidables
- Complexité
 - Classes P, NP ...
 - NP-complétude
 - Théorème de Cook

Littérature

Elements of the Theory of Computation

Harry R. Lewis, Christos H. Papadimitriou

éd. Prentice-Hall

Introduction à la calculabilité

Pierre Wolper

éd. Dunod

Introduction to the Theory of Computation

Michael Sipser, MIT

éd. Thomson Course Technology

Introduction to Theory of Computation

Anil Maheshwari, Michiel Smid, School of Computer Science, Carleton University

free textbook

Gödel Escher Bach, les Brins d'une Guirlande Eternelle

Douglas Hofstadter

éd. Dunod

Logicomix

Apóstolos K. Doxiadis, Christos Papadimitriou, Alecos Papadatos, Annie Di Donna

éd. Vuibert

LIF15 – Théorie des langages formels

Sylvain Brandel

2014 – 2015

sylvain.brandel@univ-lyon1.fr

Chapitre précédent le chapitre 4

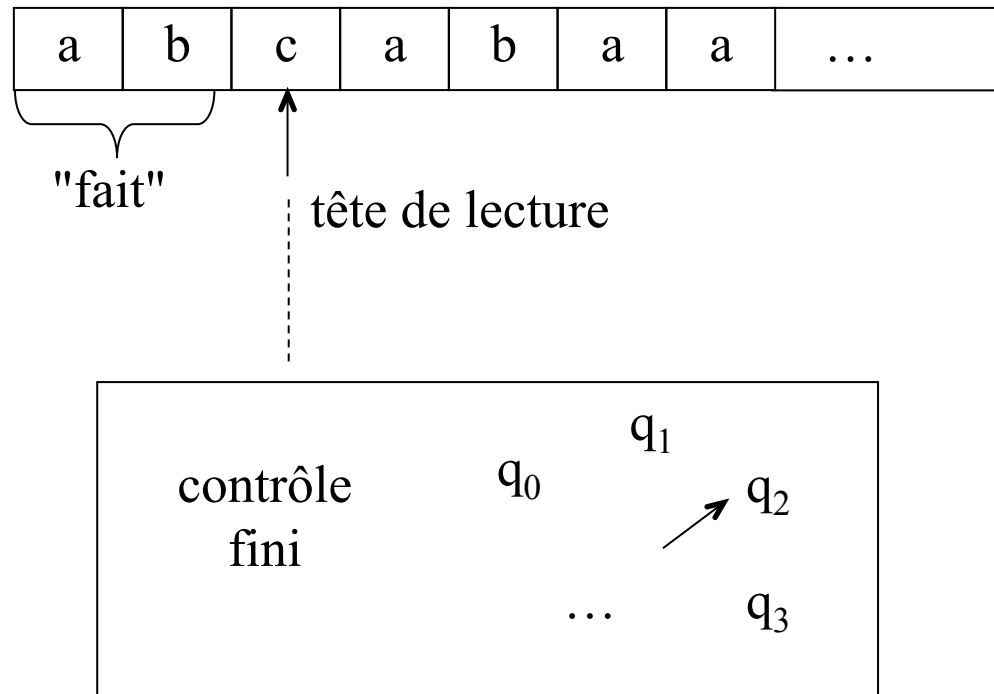
RAPPELS (RAPPELS ?)

Automates finis déterministes

- Simulation d'une machine très simple :
 - mémorisation d'un état
 - programme sous forme de graphe étiqueté indiquant les transitions possibles
- Cette machine lit un mot en entrée.
- Ce mot décrit une suite d'actions et progresse d'état en état
 - jusqu'à la lecture complète du mot.
- Lorsque le dernier état est distingué (état final)
 - on dit que le mot est accepté.

⇒ Un automate permet de reconnaître un langage.

Automates finis déterministes



- Un état dépend uniquement
 - De l'état précédent
 - Du symbole lu

Automates finis déterministes

- Un automate déterministe fini est le quintuplet

$M = (K, \Sigma, \delta, s, F)$ où :

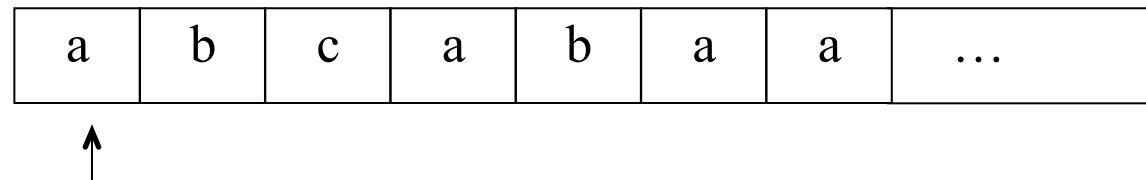
- K : ensemble fini (non vide) d'états
- Σ : alphabet (ensemble non vide de lettres)
- δ : fonction de transition : $K \times \Sigma \rightarrow K$

$\delta(q, \sigma) = q'$ (q' : état de l'automate après avoir lu la lettre σ
dans l'état q)

- s : état initial : $s \in K$
- F : ensemble des états finaux : $F \subset K$

Automates finis déterministes

- Exécution



La machine

- lit a (qui est ensuite oublié),
 - passe dans l'état $\delta(s, a)$ et avance la tête de lecture,
 - répète cette étape jusqu'à ce que tout le mot soit lu.
- La partie déjà lue du mot ne peut pas influencer le comportement à venir de l'automate.
 - d'où la notion de configuration

Automates finis déterministes

- Configuration
 - état dans lequel est l'automate
 - mot qui lui reste à lire (partie droite du mot initial)
- Formellement : une configuration est un élément quelconque de $K \times \Sigma^*$.
- Exemple
 - sur l'exemple précédent, la configuration est $(q_2, cabaa)$.

Automates finis déterministes

- Le fonctionnement d'un automate est décrit par le passage d'une configuration à une autre, cette dernière obtenue
 - en lisant un caractère,
 - et en appliquant la fonction de transition.
- Exemple
 - $(q_2, cabaa) \rightarrow (q_3, abaa)$ si $\delta(q_2, c) = q_3$

Automates finis déterministes

- Un automate M détermine une relation binaire entre configurations qu'on note \vdash_M définie par :
 - $\vdash_M \subset (K \times \Sigma^*)^2$
 - $(q, w) \vdash_M (q', w')$ ssi $\exists a \in \Sigma$ tel que $w = aw'$ et $\delta(q, a) = q'$
- On dit alors que on passe de (q, w) à (q', w') en une étape.

Automates finis déterministes

- On note \vdash_M^* la fermeture transitive réflexive de \vdash_M :
 $(q, w) \vdash_M^* (q', w')$ signifie qu'on passe de (q, w) à (q', w') en zéro, une ou plusieurs étapes
- Un mot w est accepté par M
ssi $(s, w) \vdash_M^* (q, e)$, avec $q \in F$.
- Le langage accepté par M est l'ensemble de tous les mots acceptés par M .
Ce langage est noté $L(M)$.

Automates finis déterministes

- Exemple $M = (K, \Sigma, \delta, s, F)$

- $K = \{q_0, q_1\}$

- $\Sigma = \{a, b\}$

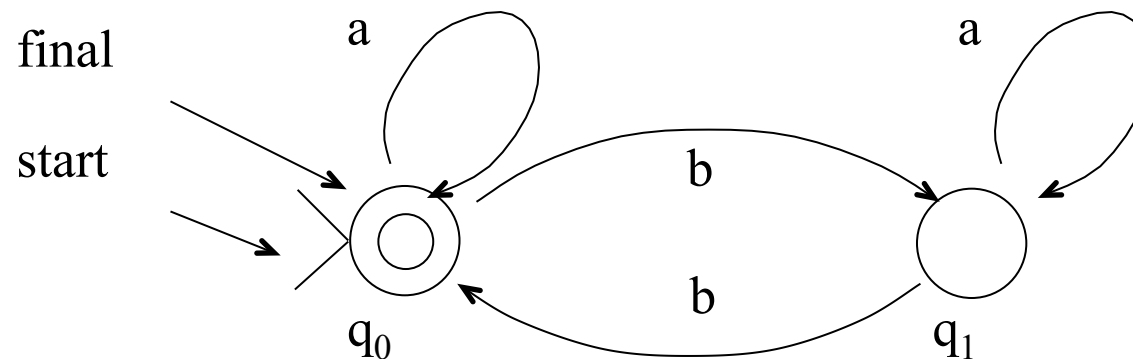
- $s = q_0$

- $F = \{q_0\}$

$\delta :$	q	σ	$\delta(q, \sigma)$
	q_0	a	q_0
	q_0	b	q_1
	q_1	a	q_1
	q_1	b	q_0

\Leftrightarrow

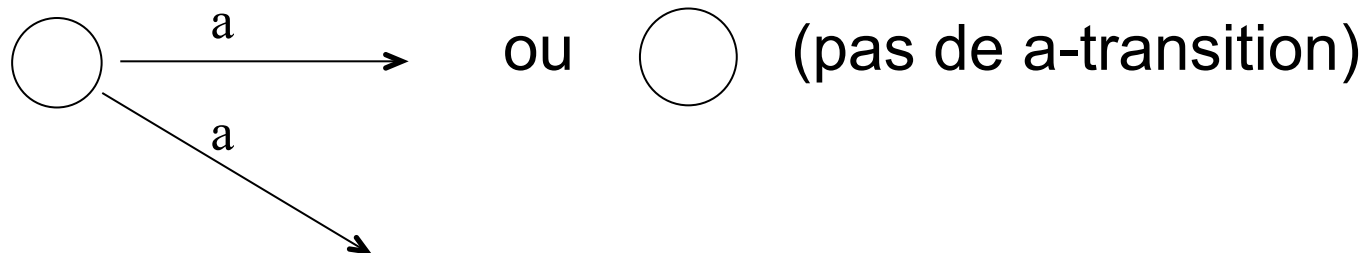
	a	b
q_0	q_0	q_1
q_1	q_1	q_0



Automates finis non déterministes

- Idée : remplacer la fonction \vdash_M (ou δ) par une relation.
- Une relation, c'est beaucoup plus général qu'une fonction.
→ on a ainsi une classe plus large d'automates.

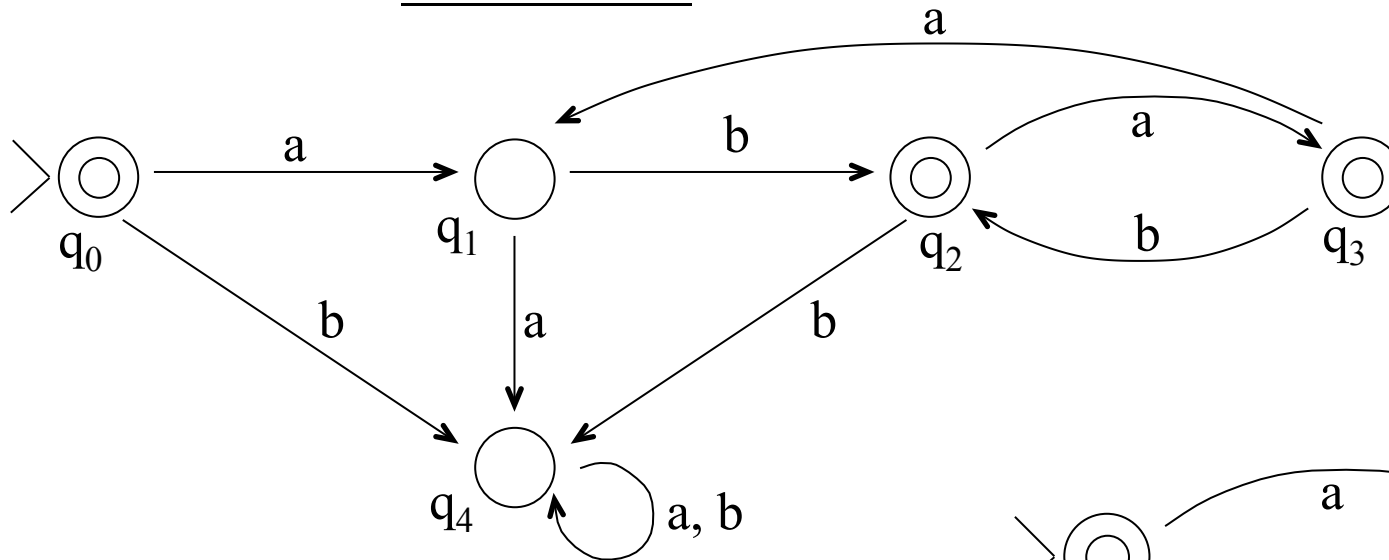
⇒ Dans un état donné, on pourra avoir :



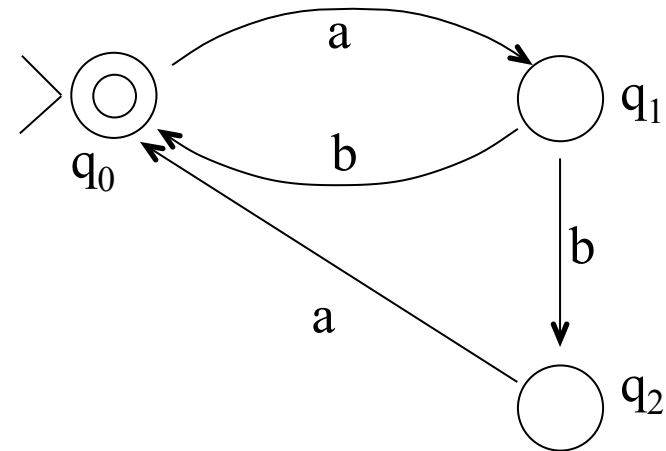
Automates finis non déterministes

- $L = (ab \cup aba)^*$

Automate déterministe :

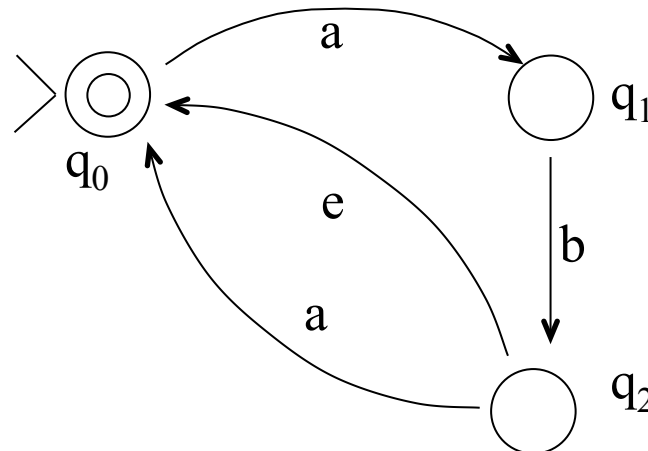


Automate non déterministe :



Automates finis non déterministes

- Dans le cas de l'automate non déterministe, un mot est accepté s'il existe un ou plusieurs chemins (au moins un) pour aller de l'état initial (ici q_0) à l'état final (ici q_0).
- Autre différence par rapport aux automates déterministes :
 - Il est possible d'étiqueter une flèche par le symbole ϵ .
- Autre formulation (encore plus intuitive) de l'automate précédent :



Automates finis non déterministes

- Un automate non déterministe fini est le quintuplet

$$M = (K, \Sigma, \Delta, s, F) \text{ où :}$$

- K : ensemble fini (non vide) d'états
- Σ : alphabet (ensemble non vide de lettres)
- Δ : relation de transition : $K \times (\Sigma \cup \{e\}) \times K$
 $(q, \sigma, p) \in \Delta$: σ -transition ($\sigma \in \Sigma$)
- s : état initial : $s \in K$
- F : ensemble des états finaux : $F \subset K$

(hormis la relation, le reste est identique à la formulation déterministe)

Automates finis non déterministes

- Si $(q, e, p) \in \Delta$: on a une ε -transition (transition spontanée)
 - On passe de q à p sans lire de symbole dans le mot courant.
- \vdash_M est une relation et non plus une fonction (automates déterministes) :
 - (q, e) peut être en relation avec une autre configuration (après une ε -transition)
 - pour une configuration (q, w) , il peut y avoir plusieurs configurations (q', w') (ou aucune) tq $(q, w) \vdash_M (q', w')$

Automates à pile

- Un automate à pile est un sextuplet

$$M = (K, \Sigma, \Gamma, \Delta, s, F) \text{ où :}$$

- K est un ensemble fini d'états,
- Σ est un ensemble fini de symboles d'entrée appelé alphabet,
- Γ est un ensemble fini de symboles de la pile,
- $s \in K$ est l'état initial,
- $F \subseteq K$ est l'ensemble des états finaux,
- Δ est un sous-ensemble fini de

$$(K \times (\Sigma \cup \{e\}) \times (\Gamma \cup \{e\})) \times (K \times (\Gamma \cup \{e\}))$$

appelé fonction de transition.

Automates à pile

- Une transition $((p, a, A), (q, B)) \in \Delta$ où :
 - p est l'état courant,
 - a est le symbole d'entrée courant,
 - A est le symbole sommet de la pile,
 - q est le nouvel état,
 - B est le nouveau symbole en sommet de pile,

a pour effet :

- (1) de passer de l'état p à l'état q ,
- (2) d'avancer la tête de lecture après a ,
- (3) de dépiler A du sommet de la pile,
- (4) d'empiler B sur la pile.

Automates à pile

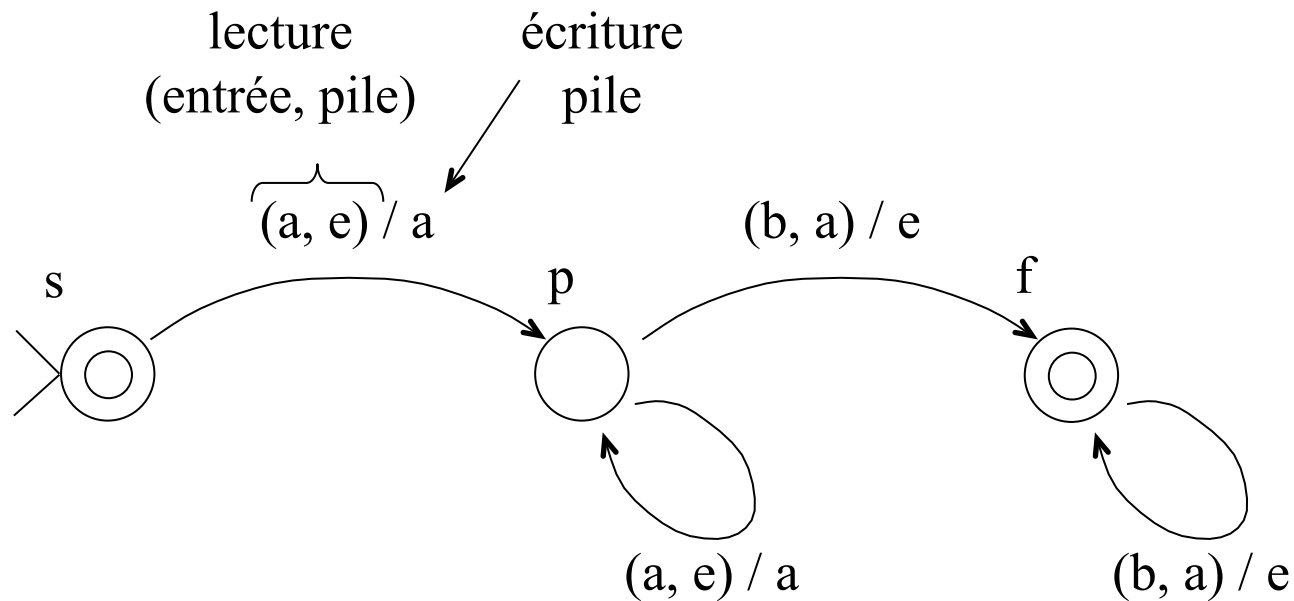
- Soit $M = (K, \Sigma, \Gamma, \Delta, s, F)$ un automate à pile. Une configuration de M est définie par un triplet
$$(q_i, w, \alpha) \in K \times \Sigma^* \times \Gamma^* \text{ où :}$$
 - q_i est l'état courant de M ,
 - w est la partie de la chaîne restant à analyser,
 - α est le contenu de la pile.
- Soient (q_i, u, α) et (q_j, v, β) deux configurations d'un automate à pile $M = (K, \Sigma, \Gamma, \Delta, s, F)$. On dit que (q_i, u, α) conduit à (q_j, v, β) en une étape
$$\text{ssi } \exists \sigma \in (\Sigma \cup \{e\}), \exists A, B \in (\Gamma \cup \{e\}) \text{ tels que :}$$
$$u = \sigma v \text{ et } \alpha = \alpha' A \text{ et } \beta = \beta' B \text{ et } ((q_i, \sigma, A), (q_j, B)) \in \Delta.$$
- On note $(q_i, u, \alpha) \vdash_M (q_j, v, \beta)$.

Automates à pile

- La relation \vdash_M^* est la fermeture réflexive transitive de \vdash_M .
- Soit $M = (K, \Sigma, \Gamma, \Delta, s, F)$ un automate à pile. Une chaîne $w \in \Sigma^*$ est acceptée par M
ssi $(s, w, e) \vdash_M^* (f, e, e)$ avec $f \in F$.
- Le langage accepté par M , noté $L(M)$, est l'ensemble des chaînes acceptées par M .

Automates à pile

- Soit l'automate à pile $M = (K, \Sigma, \Gamma, \Delta, s, F)$ avec :
 - $K = \{s, p, f\}$ $\Delta = \{((s, a, e), (p, a)),$
 - $\Sigma = \{a, b\}$ $((p, a, e), (p, a)),$
 - $\Gamma = \{a, b\}$ $((p, b, a), (f, e)),$
 - $F = \{s, f\}$ $((f, b, a), (f, e)) \}$



Automates à pile

- Un automate à pile est déterministe s'il y a au plus une transition applicable pour tout triplet de la forme
(État courant, symbole d'entrée, sommet de pile).

MIF15 – Calculabilité & complexité

Sylvain Brandel

2014 – 2015

sylvain.brandel@univ-lyon1.fr

Chapitre 4

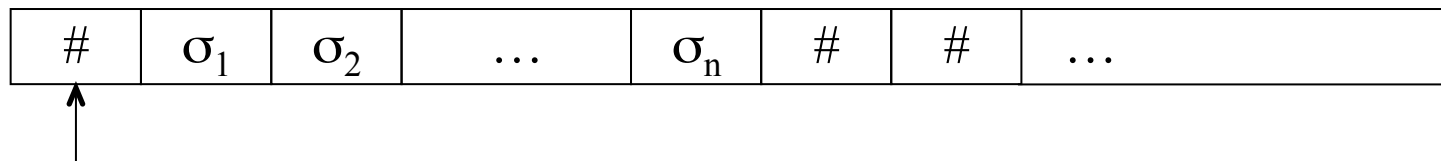
MACHINES DE TURING

Définitions

- Une machine de Turing est constituée :
 - d'un contrôle (ensemble fini d'états et de transitions),
 - d'un ruban infini à droite,
 - d'une tête sur le ruban qui peut lire et écrire, et qui peut se déplacer dans les deux directions d'un caractère à la fois.
- A chaque étape, en fonction de l'état courant et du symbole courant, la machine :
 - change d'état,
 - écrit un symbole à l'emplacement courant,
 - déplace la tête d'une position, à droite ou à gauche.

Définitions

- Initialement la machine est dans un état initial :
 - le mot $w = \sigma_1\sigma_2\ldots\sigma_n$ est dans le ruban, cadré à gauche, avec un blanc devant et une suite infinie de blancs derrière,
 - la tête de lecture / écriture pointe sur l'extrémité gauche du ruban,
 - le contrôle sur l'état initial.
- L'état initial de la machine peut être représenté par le schéma suivant (le symbole # désigne un blanc) :



Définitions

- La machine s'arrête quand elle ne peut plus appliquer de nouvelles transitions.
- Si la machine tente de se déplacer trop à gauche (au-delà de l'extrémité gauche du ruban)
 - le traitement se termine anormalement.

Définitions

- Une machine de Turing standard est un quintuplet

$$M = (K, \Sigma, \Gamma, \delta, q_0) \text{ où :}$$

- K est un ensemble fini d'états,
- Σ est l'alphabet d'entrée,
- Γ est l'alphabet des symboles du ruban,
- δ est la fonction de transition :

fonction partielle de $K \times \Gamma$ dans $K \times \Gamma \times \{G, D\}$,

(les symboles G et D désignent un déplacement
élémentaire à gauche ou à droite)

- $q_0 \in K$ est l'état initial.

Définitions

- Le symbole qui désigne le blanc (#) n'est pas dans Σ , mais appartient à Γ .
- $\Sigma \subset \Gamma$ et Γ peut contenir des symboles utilisés pour écrire sur le ruban.
- Soit la transition $\delta(q_i, a) = (q_j, b, G)$.
 - Cette transition s'applique lorsque :
 - la machine est dans l'état courant q_i ,
 - le symbole courant sur le ruban est a .
 - Après l'application de cette transition :
 - la machine est dans l'état q_j ,
 - le symbole b est écrit sur le ruban à la place de a ,
 - la tête de lecture est déplacée d'une position vers la gauche.

Définitions

- Une configuration associée à une machine de Turing

$M = (K, \Sigma, \Gamma, \delta, q_0)$ est un élément de :

$$K \times \Gamma^* \times \Gamma \times (\Gamma^* (\Gamma - \{\#\}) \cup \epsilon).$$

- Dans une configuration quelconque (q, w_1, a, u_1) :
 - la machine est dans l'état courant q ,
 - w_1 est la partie à gauche de la tête,
 - a est le symbole courant,
 - u_1 est la partie à droite de la tête jusqu'au premier $\#$ (exclu) de la suite infinie de blancs à droite.

Définitions

- Pour simplifier l'écriture des configurations, on introduit une notation abrégée sous la forme :
(état courant, contenu du ruban où le symbole courant est souligné).
- Avec cette notation :
 - la configuration $(q, e, a, bcdf)$ s'écrit $(q, \underline{a}bcdf)$,
 - la configuration $(q, ab, \#, \#f)$ s'écrit $(a, ab\underline{\#}\#f)$.

Définitions

- Soit une machine de Turing $M = (K, \Sigma, \Gamma, \delta, q_0)$ et deux configurations (q_1, w_1, a_1, u_1) et (q_2, w_2, a_2, u_2) . On dit que (q_1, w_1, a_1, u_1) conduit à (q_2, w_2, a_2, u_2) en une étape ssi :

$$\text{— soit } \delta(q_1, a_1) = (q_2, b, D) \text{ et } w_2 = w_1 b \text{ et } \begin{cases} a_2 = \# \text{ et } u_2 = e & \text{si } u_1 = e \\ \text{ou} \\ a_2 u_2 = u_1 & \text{si } u_1 \neq e \end{cases}$$

$$\text{— soit } \delta(q_1, a_1) = (q_2, b, G) \text{ et } w_2 a_2 = w_1 \text{ et } \begin{cases} u_2 = b u_1 & \text{si } b \neq \# \text{ ou } u_1 \neq e \\ \text{ou} \\ u_2 = e & \text{si } b = \# \text{ et } u_1 = e \end{cases}$$

Définitions

- On note cette relation $(q_1, w_1, a_1, u_1) \vdash_M (q_2, w_2, a_2, u_2)$.
- La relation \vdash_M^* est la fermeture réflexive transitive de la relation \vdash_M .

Exemple

- Soit la machine de Turing $M = (K, \Sigma, \Gamma, \delta, q_0)$ où :

- $K = \{q_0, q_1, q_2\}$,

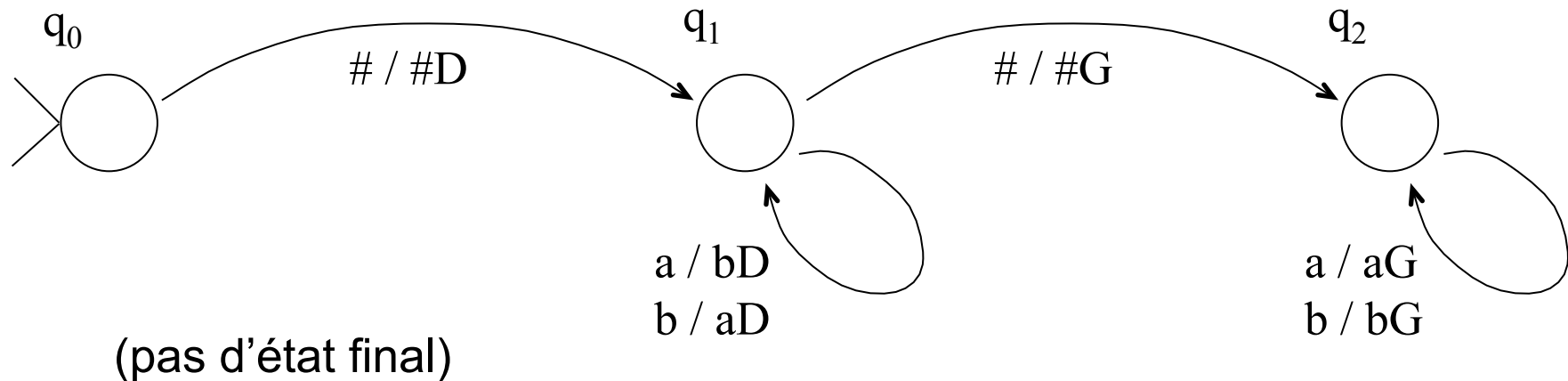
- $\Sigma = \{a, b\}$,

- $\Gamma = \{a, b, \#\}$

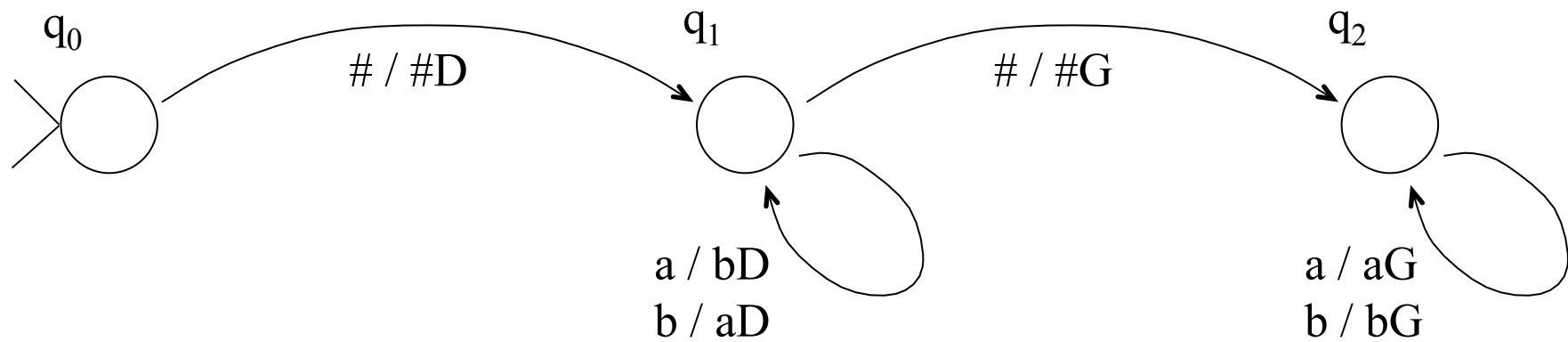
δ	#	a	b
q_0	$(q_1, \#, D)$		
q_1	$(q_2, \#, G)$	(q_1, b, D)	(q_1, a, D)
q_2		(q_2, a, G)	(q_2, b, G)

(δ : fonction **partielle**)

- Représentation graphique de M :



Démo



Et maintenant ?

- Les machines de Turing peuvent être utilisées :
 - soit pour reconnaître (ou accepter) un langage,
 - soit pour calculer une fonction.
- Et si ça devient trop compliqué ?
 - On fait des combinaisons de machines plus simples
- Ce formalisme peut-il être étendu pour construire des machines plus puissantes ? Pour reconnaître une classe de langages plus grande ? Ou calculer plus de fonctions ?
 - On imagine des extensions et on regarde ...

Les machines de Turing pour reconnaître des langages

- Dans ce contexte, il faut modifier le concept de machine de Turing standard introduit au paragraphe précédent :
→ ajouter la notion d'état final.
- Une machine de Turing augmentée avec des états finaux est le sextuplet $M = (K, \Sigma, \Gamma, \delta, q_0, F)$ où :
 $F \subseteq K$ est l'ensemble des états finaux.

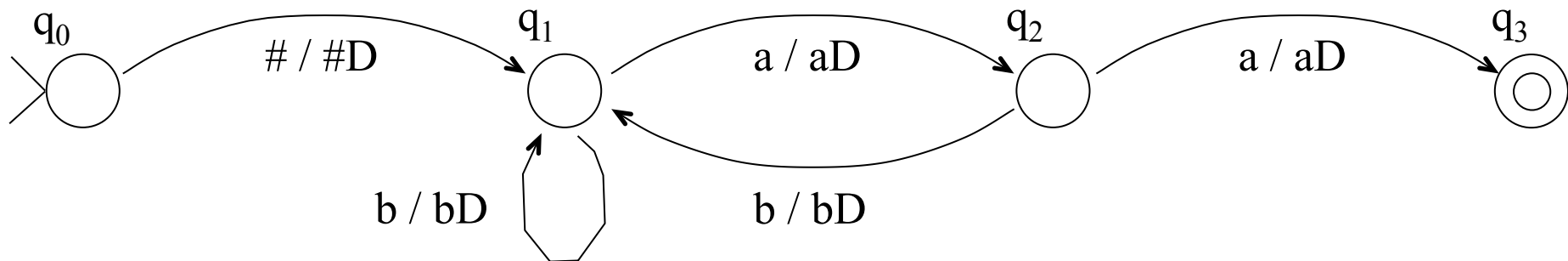
(le reste ne change pas)

Les machines de Turing pour reconnaître des langages

- Soit $M = (K, \Sigma, \Gamma, \delta, q_0, F)$ une machine de Turing. Une chaîne $w \in \Sigma^*$ est acceptée par M ssi :
 $(q_0, \#w) \vdash_M^* (q_f, w'\underline{a}w'')$ où :
 - $q_f \in F$,
 - $a \in \Gamma$,
 - $w', w'' \in \Gamma^*$,
 - $\delta(q_f, a)$ n'est pas défini.
- Le langage accepté par M , noté $L(M)$, est l'ensemble de toutes les chaînes acceptées par M .

Les machines de Turing pour reconnaître des langages

- Exemple



- La suite de configurations associée au mot aabb est :
 $(q_0, \underline{\#}aabb) \vdash_M (q_1, \#a\underline{a}bb) \vdash_M (q_2, \#a\underline{a}bb) \vdash_M (q_3, \#aabb)$
Comme l'état q_3 est final, et qu'il n'y a pas de transition depuis q_3 , le mot $w = aabb$ est accepté.
- Pour tout mot w ne contenant pas aa , le calcul s'arrête sur le premier $\#$ à droite de w sur le ruban dans un état non final.

Les machines de Turing pour reconnaître des langages

- Le langage accepté par une machine de Turing est dit Turing-acceptable ou rékursivement énumérable.
- Si la machine de Turing s'arrête sur toutes les entrées possibles (c-à-d pour tous les mots w , $w \in L$ ou $w \notin L$), alors le langage est dit Turing-décidable ou récuratif.

On dit que M semi-décide L , ou encore M accepte L .

- On a alors :
 - $\forall w \in L, (q_0, \#w) \vdash_M^* (q_f, \#Y) (q_f \in F) \rightarrow \text{YES (accepté)}$
 - $\forall w \notin L, (q_0, \#w) \vdash_M^* (q_f, \#N) (q_f \in F) \rightarrow \text{NO (rejeté)}$

Les machines de Turing pour calculer des fonctions

- L'idée est d'utiliser les machines de Turing pour calculer des fonctions de chaînes vers chaînes.
- Soient
 - Σ_0 et Σ_1 deux alphabets ne contenant pas le symbole blanc ($\#$),
 - f une fonction de Σ_0^* vers Σ_1^* .
- Une machine de Turing $M = (K, \Sigma_0, \Gamma, \delta, q_0, F)$ calcule la fonction f ssi :
 - $\forall w \in \Sigma_0^*$ tel que $f(w) = u$, on a
 $(q_0, \#w) \vdash_M^* (q_f, \#u)$ où :
 - $q_f \in F$,
 - $\delta(q_f, \#)$ n'est pas défini.

Les machines de Turing pour calculer des fonctions

- Lorsqu'une telle machine de Turing existe, la fonction est dite Turing-calculable.
- La notion de Turing-calculable n'est pas restreinte aux fonctions de chaînes vers chaînes.
 - elle peut être étendue de plusieurs façons :
 - Nombre quelconque d'arguments
 - Pour des fonctions de **N** dans **N**

Les machines de Turing pour calculer des fonctions

- Fonctions avec un nombre quelconque d'arguments de la forme $f : (\Sigma_0^*)^k \rightarrow \Sigma_1^*$:
- Une machine de Turing $M = (K, \Sigma_0, \Gamma, \delta, q_0, F)$ calcule la fonction f ssi :
 - $\forall \sigma_1, \sigma_2, \dots, \sigma_k \in \Sigma_0^*$ tels que $f(\sigma_1, \sigma_2, \dots, \sigma_k) = u$, on a :
 $(q_0, \# \sigma_1 \# \sigma_2 \# \dots \# \sigma_k) \vdash_M^* (q_f, \# u)$ où :
 $q_f \in F$,
 $\delta(q_f, \#)$ n'est pas défini.

Les machines de Turing pour calculer des fonctions

- Fonctions de \mathbb{N} dans \mathbb{N} :
- Notons I un symbole fixé différent de $\#$.
 - Tout entier naturel n peut être représenté par la chaîne I^n en notation unaire
 - (dans ce cas l'entier zéro est représenté par la chaîne vide)
- Une fonction $f : \mathbb{N} \rightarrow \mathbb{N}$ est calculée par une machine de Turing M , si M calcule la fonction $f' : \{I\}^* \rightarrow \{I\}^*$ telle que $f'(I^n) = I^{f(n)}$ pour tout $n \in \mathbb{N}$.

Les machines de Turing pour calculer des fonctions

- La fonction successeur définie par $\text{succ}(n) = n + 1$, $\forall n \geq 0$, est calculée par la machine de Turing

$M = (K, \Sigma, \Gamma, \delta, q_0, F)$ où :

– $K = \{q_0, q_1, q_2\}$,

– $\Sigma = \{I\}$,

– $\Gamma = \{I, \#\}$,

– $F = \{q_2\}$

δ	#	I
q_0	$(q_1, \#, D)$	
q_1	(q_2, I, G)	(q_1, I, D)
q_2		(q_2, I, G)

Combinaison des machines de Turing

- On présente ici une méthode pour combiner des machines de Turing simples
 - machines plus complexes
 - ⇒ machine de Turing = module ou sous-routine
pour faciliter la conception.
- Deux types de machines de base :
 - Les machines qui écrivent un symbole
 - Les machines qui déplacent la tête d'une position

Combinaison des machines de Turing

1. Les machines qui écrivent un symbole

- Une machine pour chaque symbole de l'alphabet Γ .
- Une telle machine :
 - écrit le symbole spécifié sur le symbole courant (dont le contenu est ignoré),
 - et s'arrête sans bouger la tête.
- Elle est simplement appelée a si a est le symbole spécifié.

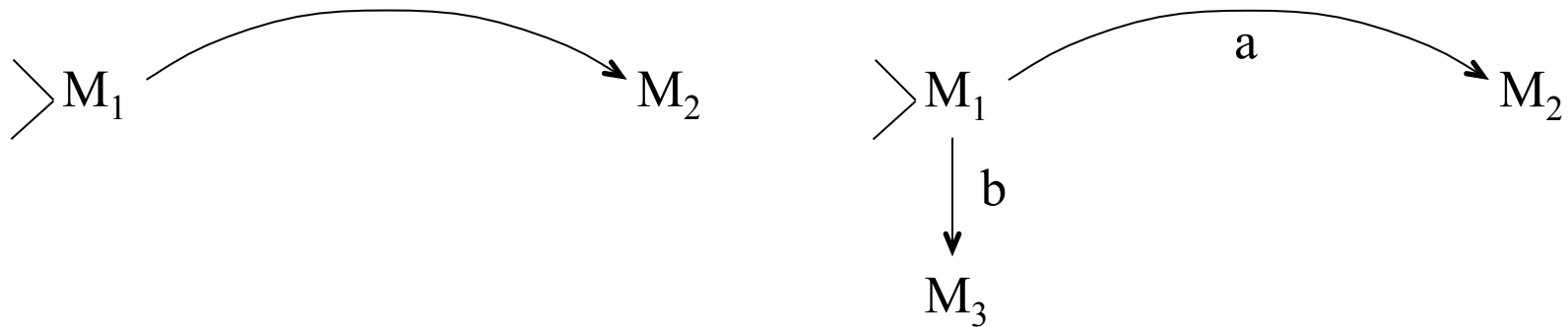
Combinaison des machines de Turing

2. Les machines qui déplacent la tête d'une position

- Il existe deux machines de ce type :
 - $G = (\{q_0, q_f\}, \Sigma, \Sigma \cup \{\#\}, \delta_G, q_0, \{q_f\})$
avec $\delta_G(q_0, \sigma) = (q_f, \sigma, G), \forall \sigma \in \Gamma,$
 - $D = (\{q_0, q_f\}, \Sigma, \Sigma \cup \{\#\}, \delta_D, q_0, \{q_f\})$
avec $\delta_D(q_0, \sigma) = (q_f, \sigma, D), \forall \sigma \in \Gamma.$

Combinaison des machines de Turing

- Ces deux types de machines peuvent être connectées entre elles en utilisant des règles de combinaisons.
- Exemple
 - Soient M_1 , M_2 , M_3 des machines de Turing quelconques.



Combinaison des machines de Turing

$\rangle D \quad \text{noté } D_{\#}$ déplace la tête à droite du symbole courant jusqu'au premier blanc.

$\rangle G \quad \text{noté } G_{\#}$ déplace la tête à gauche du symbole courant jusqu'au premier blanc.

$\rangle D \quad \text{noté } D_{\neg\#}$ déplace la tête à droite du symbole courant jusqu'au premier non blanc.

$\rangle G \quad \text{noté } G_{\neg\#}$ déplace la tête à gauche du symbole courant jusqu'au premier non blanc.

Extensions des machines de Turing

- Est-il possible d'accroître la puissance des machines de Turing ?
- Examinons des extensions :
 - (a) un ruban infini dans les deux directions,
 - (b) plusieurs rubans,
 - (c) plusieurs têtes sur le ruban,
 - (d) un ruban bidimensionnel,
 - (e) le non-déterminisme,
 - (f) l'accès aléatoire.
- Et montrons que ces machines étendues peuvent être simulées par des machines standard

Extensions des machines de Turing

- Pour chaque type d'extension, nous montrons que l'opération de la machine étendue peut être simulée par une machine de Turing normale.
- La démonstration consiste dans chaque cas :
 - (1) à montrer comment construire une machine normale à partir de la machine étendue considérée,
 - (2) à prouver que la machine normale construite simule correctement le comportement de la machine de départ.

Machine de Turing à ruban infini dans les deux sens

- Soit une machine de Turing $M = (K, \Sigma, \Gamma, \delta, q_0, F)$ dont le ruban n'a pas de borne à gauche :
 - la chaîne d'entrée peut se trouver n'importe où sur le ruban,
 - la tête pointe sur le premier blanc à gauche de la chaîne.
- Dans cette machine, une configuration est de la forme : $(q, w\underline{a}u)$ avec $q \in K$, $w, u \in \Gamma^*$, $a \in \Gamma$, où :
 - w ne commence pas par un blanc
 - u ne finit pas par un blanc.
- On étend la relation entre configurations pour prendre en compte les déplacements à gauche :
 - si $\delta(q, a) = (p, b, G)$ alors $(q, \underline{a}u) \vdash_M (p, \#bu)$.

Machine de Turing à ruban infini dans les deux sens

- Montrons qu'une machine M avec ruban infini dans les deux sens n'est pas plus puissante qu'une machine normale (dans le sens qu'elle ne permet pas de reconnaître plus de langages, ou calculer plus de fonctions).
- Pour cela montrons comment construire une machine $M' = (K', \Sigma, \Gamma', \delta', q_0', F')$, à partir de M et qui simule M :
 - si M s'arrête sur un mot w , alors M' s'arrête sur ce même mot w ,
 - si M ne s'arrête pas sur un mot w , alors M' ne s'arrête pas non plus sur ce même mot w .

Machine de Turing à ruban infini dans les deux sens

- Pour simuler le ruban doublement infini de M dans celui de M' , on définit pour M' un ruban à 2 pistes :
- Ce ruban est obtenu en coupant en 2 celui de M de façon arbitraire.
- Exemple

Ruban de M :

...	f	g	h	i	j	k	...
B				A			

Ruban de M' :

\$	i	j	k	...
	h	g	f	...

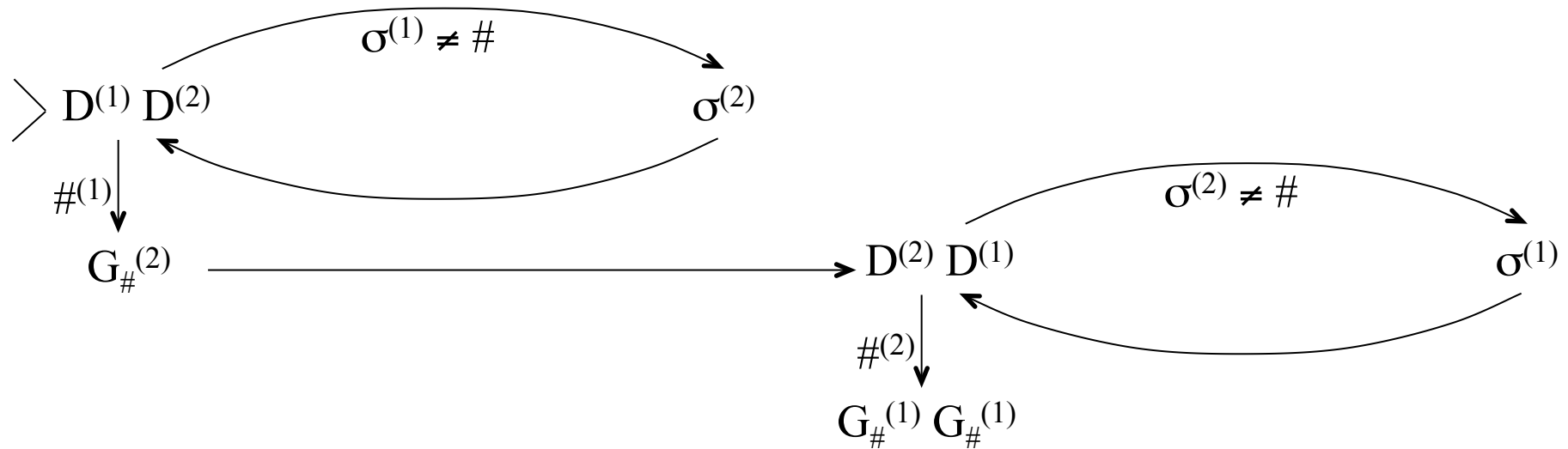
Principe de la simulation au tableau...

Machine de Turing à plusieurs rubans

- Une machine de Turing étendue peut être caractérisée par k rubans, chaque ruban étant munie d'une tête autonome.
- Une telle machine respecte les conventions :
 - La chaîne d'entrée est initialement placée sur le premier ruban, cadrée à gauche et précédée d'un blanc, avec la tête sur ce blanc.
 - Les autres rubans sont remplis de blancs avec la tête à l'extrême gauche.
 - Lorsque la machine s'arrête, la chaîne de sortie se trouve sur le premier ruban, les autres rubans sont ignorées.

Machine de Turing à plusieurs rubans

- Exemple : machine à copier à deux rubans



- L'exposant (1) ou (2) indique qu'on se trouve sur le premier ou le deuxième ruban.

Machine de Turing à plusieurs têtes

- Machine étendue avec plusieurs têtes sur le même ruban.
 - Simplifier la construction de certaines machines
- ⇒ il est possible d'implanter une machine à copier avec 2 têtes.

Machine de Turing multi-dimensionnelles

- Pour une machine bidimensionnelle, par exemple, on n'a pas de ruban, mais un plan (discret).
- Il faut donc tenir compte des mouvements : D, G, H, B mais aussi les déplacements en diagonale.

Machine de Turing à mémoire à accès direct

- *Random Access* : on peut accéder à chaque case en une étape, contrairement au ruban d'une machine de Turing qui est à accès séquentiel
- Cette machine comporte :
 - T : ruban à accès direct
 - $T[0]$, $T[1]$, $T[2]$, $T[3]$... : cases du ruban
 - R_0 , R_1 , R_2 , R_3 : registres
 - K : compteur de programme (qui est un registre particulier).

Machine de Turing à mémoire à accès direct

- Instructions :

- read j $R_0 := T[R_j]$ placer dans le 1^{er} registre le contenu de la $R_j^{\text{ème}}$ case, R_j étant la valeur du $j^{\text{ème}}$ registre
- write j $T[R_j] := R_0$
- store j $R_j := R_0$ placer le contenu du 1^{er} registre dans le $j^{\text{ème}}$ registre
- load j $R_0 := R_j$
- load = c $R_0 := c$
- add j $R_0 := R_0 + R_j$
- add = c $R_0 := R_0 + c$ c : nombre entier
- sub j $R_0 := \max\{ R_0 - R_j, 0 \}$
- sub = c $R_0 := \max\{ R_0 - c, 0 \}$
- half $R_0 := \text{integer}\{ R_0 / 2 \}$

Machine de Turing à mémoire à accès direct

- Instructions de contrôle :

- jump s $K := s$ s : numéro d'instruction
- jpos s if $R_0 > 0$ then $K := s$
- jzero s if $R_0 = 0$ then $K := s$
- halt $k := 0$

- Remarques

- Chaque instruction incrémente K : $K := K + 1$
- R_0 : rôle particulier (accumulateur)

Machine de Turing à mémoire à accès direct

- Ainsi une machine de Turing à accès direct est un couple $M = (k, \Pi)$ où :
 - $k > 0$ est le nombre de registres,
 - Π est une suite finie d'instructions (le programme).
- Une configuration d'une machine $M = (k, \Pi)$ est un $(k+2)$ -uplet $(m, R_0, \dots, R_{k-1}, T)$ où :
 - m est le compteur de programme,
 - R_j ($0 \leq j < k$) est le contenu du $j^{\text{ème}}$ registre
 - T est un ensemble de couples d'entiers : $(i, j) \in T$ signifie que la $i^{\text{ème}}$ case du ruban contient la valeur j .

Machine de Turing à mémoire à accès direct

- Une telle machine peut être simulée par une machine de Turing à plusieurs rubans :
 - un ruban pour la mémoire,
 - un ruban pour le programme
 - un ruban pour chaque registre.
 - Le contenu de la mémoire est représenté par des paires de mots (adresse, contenu).
- La simulation pourrait être la répétition du cycle suivant :
 - parcourir le ruban du programme jusqu'à l'instruction correspondant à la valeur trouvée dans le compteur de programme,
 - lire et décoder l'instruction,
 - exécuter l'instruction → modifications éventuelles des rubans correspondant à la mémoire et / ou aux registres,
 - incrémenter le compteur de programme.

Machine de Turing à mémoire à accès direct

- Exemple

1. store 2	11. add 2
2. load 1	12. store 4
3. jzero 19	13. load 2
4. half	14. add 2
5. store 3	15. store 2
6. load 1	16. load 3
7. sub 3	17. store 1
8. sub 3	18. jump 2
9. jzero 13	19. load 4
10. load 4	20. halt

read j	$R_0 := T[R_j]$
write j	$T[R_j] := R_0$
store j	$R_j := R_0$
load j	$R_0 := R_j$
add j	$R_0 := R_0 + R_j$
sub j	$R_0 := \max\{R_0 - R_j, 0\}$
half	$R_0 := \text{integer}\{R_0 / 2\}$
jump s	$K := s$
jpos s	if $R_0 > 0$ then $K := s$
jzero s	if $R_0 = 0$ then $K := s$
halt	$k := 0$

Ca fait quoi ?

Si initialement R_0 contient x et R_1 contient y , que contient R_0 à la fin de l'exécution ? (convention : tous les registres contiennent initialement 0)

Machine de Turing non déterministe

- Pour un état et un symbole courant, il peut y avoir plusieurs choix de comportements possibles.
- Une telle machine est décrite par $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ où :
 - Δ est un sous-ensemble de $K \times \Gamma \times K \times \Gamma \times \{G \cup D\}$.
 - (Machine de Turing classique : δ est une fonction partielle de $K \times \Gamma$ dans $K \times \Gamma \times \{G \cup D\}$.)
- Ainsi une machine de Turing non déterministe peut produire deux sorties différentes pour une même entrée.
 - Une machine non déterministe est donc un accepteur dont le seul résultat qui nous intéresse est de savoir si la machine s'arrête ou non, sans considérer le contenu du ruban.

Machine de Turing non déterministe

- Le non déterminisme n'apporte aucune puissance supplémentaire.

En effet, pour toute machine de Turing non déterministe M , on peut construire une machine normale M' telle que pour toute chaîne $w \in \Sigma^*$ on a:

- si M s'arrête avec w en entrée, alors M' s'arrête sur w ,
- si M ne s'arrête pas sur l'entrée w , alors M' ne s'arrête pas non plus sur w .

Principe de la simulation au tableau...

- Théorème

Tout langage accepté par une machine de Turing non déterministe est accepté par une machine de Turing déterministe

Machines de Turing universelles

- Existe-t-il une machine de Turing qui peut simuler n'importe quelle machine de Turing ?
- Le but est de construire une machine de Turing M à laquelle on fournit :
 - la description d'une machine de Turing quelconque M'
 - un mot wet qui simule l'exécution de M sur w .
- En clair construire une machine de Turing qui serait un interpréteur de machines de Turing...
- Ces machines de Turing existent et sont appelées machines de Turing universelles.

Grammaires

- Une grammaire (générale) est un quadruplet

$G = (V, \Sigma, R, S)$ où :

- V : symboles non terminaux
- Σ : symboles terminaux ($V \cap \Sigma = \emptyset$)
- $S \in V$: symbole de départ
- R est l'ensemble de règles :

sous ensemble fini de $(V \cup \Sigma)^* V (V \cup \Sigma)^* \times (V \cup \Sigma)^*$

Au moins un non-terminal

(Dans une grammaire algébrique, $R \subset V \times (V \cup \Sigma)^*$.)

Un et un seul non-terminal

Grammaires

- Soient u et $v \in (V \cup \Sigma)^*$.
On dit que v dérive directement de u , et on note $u \Rightarrow_G v$,
ssi $\exists x, y, w \in (V \cup \Sigma)^*, \exists A \in V$ tels que
 $u = xAy$ et $v = xwy$ et $A \rightarrow w \in R$
- La relation \Rightarrow_G^* est la fermeture réflexive transitive de la relation \Rightarrow_G .
- Soient u et $v \in (V \cup \Sigma)^*$.
On dit que v dérive de u , et on note $u \Rightarrow_G^* v$,
ssi $\exists w_0, \dots, w_n \in (V \cup \Sigma)^*$ tels que
 $u = w_0$ et $v = w_n$ et $w_i \Rightarrow_G w_{i+1} \forall i < n$.

Grammaires

- La suite $w_0 \Rightarrow_G w_1 \Rightarrow_G \dots \Rightarrow_G w_n$ est appelée une dérivation
- La valeur de n ($n \geq 0$) est la longueur de la dérivation.
- Soit $G = (V, \Sigma, R, S)$ une grammaire. Le langage engendré par G , noté $L(G)$, est :

$$L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G^* w \}$$

- Deux grammaires qui engendrent le même langage sont dites équivalentes.

Grammaires

- Exemple :

$G = (V, \Sigma, R, S)$ où :

- $V = \{ S, A, B, C, T_a, T_b, T_c \}$
- $\Sigma = \{ a, b, c \}$
- $R = \{ S \rightarrow ABCS,$
 $S \rightarrow T_c,$
 $CA \rightarrow AC,$
 $BA \rightarrow AB,$
 $CB \rightarrow BC,$
 $CT_c \rightarrow T_c c,$
 $CT_c \rightarrow T_b c,$
 $BT_b \rightarrow T_b b,$
 $BT_b \rightarrow T_a b,$
 $AT_a \rightarrow T_a a,$
 $T_a \rightarrow e \}$

le langage généré est $\{ a^n b^n c^n \mid n \geq 1 \}$.

Preuve : TD

Grammaires

- Théorème
 - Un langage L est engendré par une grammaire générale ssi il est récursivement énumérable.
(c-à-d accepté par une machine de Turing)

Grammaires

Calculabilité grammaticale

- Soit $G = (V, \Sigma, R, S)$ une grammaire et $f : \Sigma^* \rightarrow \Sigma^*$ une fonction. On dit que G calcule f si $\forall w, v \in \Sigma^*$ on a :

$$SwS \Rightarrow_G^* v \text{ ssi } v = f(w)$$

c-à-d toute dérivation par G de SwS donne v

- Une fonction $f : \Sigma^* \rightarrow \Sigma^*$ est grammaticalement calculable ssi il existe une grammaire la calculant.
- Théorème
Une fonction $f : \Sigma^ \rightarrow \Sigma^*$ est récursive (Turing-calculable) ssi elle est grammaticalement calculable*

Fonctions numériques

Fonctions primitives récurives

- Fonctions de base

($k \geq 0$)

(a) $\text{zéro}_k : N^k \rightarrow N$, définie par

$$\forall n_1, \dots, n_k \in N, \text{zéro}_k(n_1, \dots, n_k) = 0.$$

(b) $j^{\text{ème}}$ k -projecteur ($j^{\text{ème}}$ k -identité) : $\text{id}_{k,j} : N^k \rightarrow N$

$$\forall n_1, \dots, n_k \in N, \text{id}_{k,j}(n_1, \dots, n_k) = n_j \text{ (pour } 1 \leq j \leq k)$$

(c) successeur : $\forall n \in N, \text{succ}(n) = n+1$

Fonctions numériques

Fonctions primitives récurives

- Opérations sur les fonctions

(1) composition :

$$g : \mathbb{N}^k \rightarrow \mathbb{N}$$

$$h_1, \dots, h_k : \mathbb{N}^p \rightarrow \mathbb{N}$$

Fonction f : composée de g avec h_1, \dots, h_k

$$f : \mathbb{N}^p \rightarrow \mathbb{N}$$

$$f(n_1, \dots, n_p) = g(h_1(n_1, \dots, n_p), \dots, h_k(n_1, \dots, n_p))$$

(2) récursivité :

$$g : \mathbb{N}^k \rightarrow \mathbb{N}$$

$$h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$$

Fonction f : définie récursivement par g et h :

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

$$f(n_1, \dots, n_k, 0) = g(n_1, \dots, n_k)$$

$$f(n_1, \dots, n_k, m+1) = h(n_1, \dots, n_k, m, f(n_1, \dots, n_k, m))$$

Fonctions numériques

Fonctions primitives récurives

- Fonctions primitives récurives : ensemble de fonctions de $\mathbb{N}^k \rightarrow \mathbb{N}$ (pour tout $k \in \mathbb{N}$) pouvant être définies :
 - à partir des fonctions de base,
 - à l'aide des opérateurs composition et récursivité.

Fonctions numériques

Fonctions primitives récursives

- plus : $\mathbb{N}^2 \rightarrow \mathbb{N}$ $(n, m) \mapsto n + m$
 $\text{plus}(n, 0) = n$
 $= \text{id}_{1,1}(n)$
 $\text{plus}(n, m+1) = \text{succ}(\text{plus}(n, m))$
 $= \text{succ}(\text{id}_{3,3}(n, m, \text{plus}(n, m)))$
- Exemple de prédicat primitif récursif
 $\text{iszéro}(0) = 1$
 $\text{iszéro}(n+1) = 0$ (1 signifie vrai, 0 signifie faux)
- Définition par cas :

$$f(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n) & \text{si } p(x_1, \dots, x_n) \\ g_2(x_1, \dots, x_n) & \text{sinon} \end{cases}$$

Fonctions numériques

Minimisation

- Définition (minimisation d'une fonction)

Soit g une fonction $(k+1)$ -aire, pour un certain $k \geq 0$.

La minimisation de g est la fonction $f : \mathbb{N}^k \rightarrow \mathbb{N}$ définie par

$$f(n_1, \dots, n_k) = \begin{cases} \text{le plus petit } m \text{ (s'il existe)} \\ \text{tel que } g(n_1, \dots, n_k, m) = 1 \\ 0 \text{ sinon} \end{cases}$$

Fonctions numériques

Minimisation

Cette minimisation n'est pas toujours une fonction calculable :

L'algorithme

$m := 0$

tant que $g(n_1, \dots, n_k, m) \neq 1$ faire $m := m+1$ fait

retourner m


peut ne pas se terminer.

Fonctions numériques

Minimisation

- On dit qu'une fonction g est minimisable si sa minimisation est calculable par l'algorithme précédent, c-à-d ssi :

$$\forall n_1, \dots, n_k \in \mathbb{N}, \exists m \in \mathbb{N} \text{ tel que } g(n_1, \dots, n_k, m) = 1$$

On note alors $\mu m[g(n_1, \dots, n_k, m)] =$  $\begin{cases} \text{le plus petit } m \text{ (s'il existe)} \\ \text{tel que } g(n_1, \dots, n_k, m) = 1 \\ 0 \text{ sinon} \end{cases}$

Fonctions numériques

Fonctions μ -récurives

- Les fonctions μ -récurives sont les fonctions obtenues à partir :
 - des fonctions de base
 - des opérations de composition et de récursivité
 - de la minimisation pour les fonctions minimisables

Fonctions numériques

Fonctions μ -récurives

- Exemple

Si $\text{Log}(m, n) = \lceil \log_{m+2}(n+1) \rceil$

Alors on a $\text{Log}(m, n) = \mu p[(m+2)^{\uparrow p} \geq n+1]$

- Théorème (équivalence μ -réursive et réursive)

*Une fonction $f : N^k \rightarrow N$ est μ -réursive ssi elle est réursive
(= Turing-calculable).*