Calculabilité
& Complexité Algorithmique

Nicolas Bedon

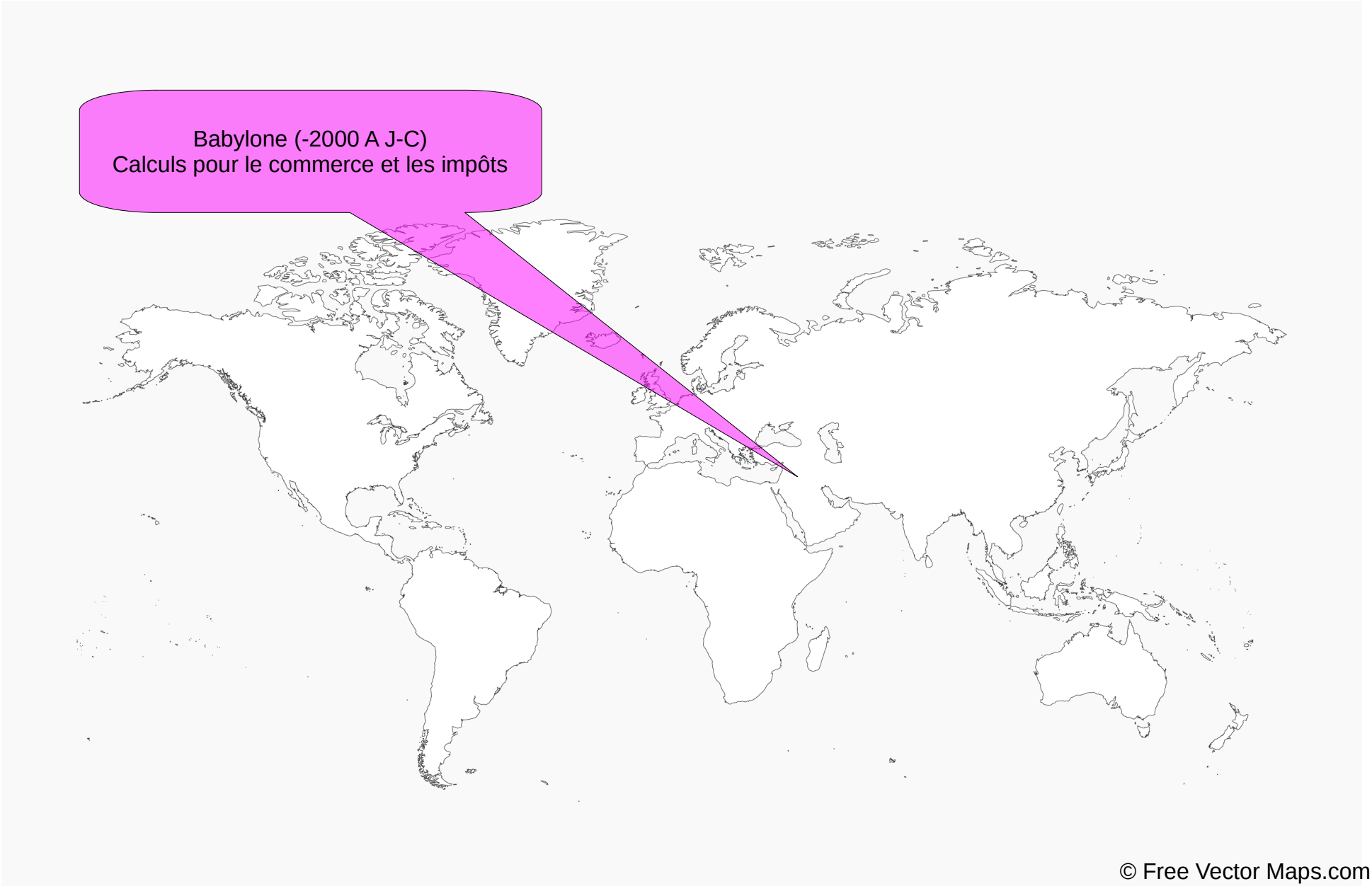
Cours de Master 1
Université de Rouen

Bibliographie

- Introduction to Automata Theory, Languages, and Computation, John E. Hopcroft & Jeffrey D. Ullman, Addison-Wesley, 1979
- Computational Complexity, Christos H. Papadimitriou, Addison-Wesley, 1993
- Introduction to the theory of computation, Michael Sipser
- Calculabilité et Décidabilité, Jean-Michel Autebert, Masson, 1992
- Complexité Algorithmique, Sylvain Périfel, Ellipses, 2014

Un peu d'histoire

Un peu d'histoire

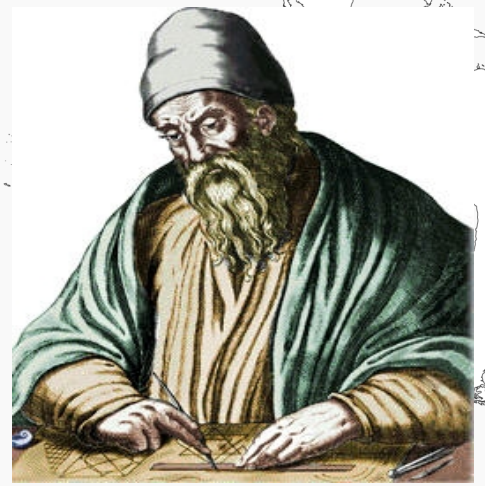


Un peu d'histoire

Euclide (Grèce, -300 A J-C)
Calcul du pgcd

$a_1 = k_1 * a_2 + a_3$
$a_2 = k_2 * a_3 + a_4$
$a_4 = k_3 * a_5 + a_6$
…
$a_n = k_n * a_{n+1}$

$pgcd(a_1, a_2) = a_n$



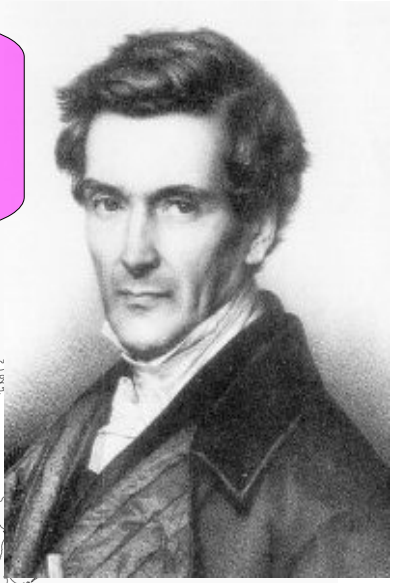
Un peu d'histoire

Al Khuwarizmi (Perse, +900 JC)
Ouvrage consacré aux algorithmes

Un peu d'histoire

Pascaline (Rouen, XVIIe siècle)
Machine à calculer

Wantzel (Paris, XIXe siècle)
Quelles longueurs sont constructibles
à la règle et au compas ?
+ algorithme de construction



Leibnitz (Allemagne, XVIIe siècle)
Construction d'automates

Un peu d'histoire

- Fin XIXè: progrès importants en méta-mathématiques
  - Cantor: développement de la théorie moderne « naïve » des ensembles
  - Zermelo & Fraenkel: axiomatisation de la théorie moderne des ensembles
- Début XXè: programme de Hilbert
  - Ignorabimus « Nous ne savons pas et ne saurons jamais »
    vs « Nous devons savoir. Nous saurons »
  - Formalisation des fondements des mathématiques… et d'autres sciences !
  - L'axiomatisation permet la mécanisation de la connaissance
  - 23 problèmes dont:

1) Hypothèse du continu

2) Cohérence de la présentation axiomatique de l'arithmétique

6) Axiomatisation de la physique

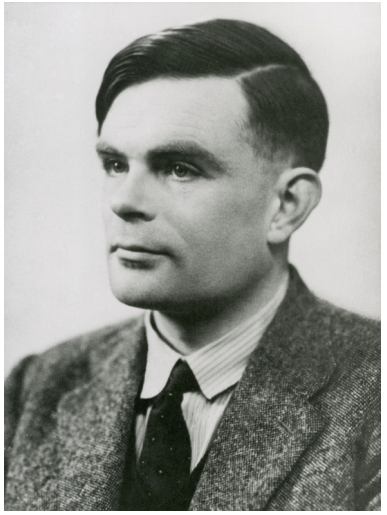
10) Trouver un algorithme déterminant si une équation diophantienne a des solutions.

Un peu d'histoire

- 1928: *Entscheidungsproblem* de Hilbert
  - Existence d'un algorithme permettant de décider de la véracité d'un énoncé mathématique ?
- 1929: Gödel montre la complétude du calcul des prédicats du premier ordre
  - si un énoncé est conséquence sémantique d'une théorie, c'est-à-dire qu'il est vérifié dans tous les modèles de cette théorie, alors il est conséquence syntaxique de cette théorie : il existe une démonstration formelle dans la syntaxe du calcul des prédicats qui dérive cet énoncé à partir des axiomes de la théorie en utilisant les règles d'un système de déduction
- 1931: Théorèmes d'incomplétude de Gödel
  - Toute théorie cohérente capable de formaliser l'arithmétique est incomplète
  - La cohérence d'une théorie T (satisfaisant certaines hypothèses raisonnables) n'est pas montrable dans T

Fin de l'idée de “mécanisation”, par l'axiomatisation, des mathématiques de Hilbert !

Un peu d'histoire



- Church (1930) et Turing (1936):
  - fonctions définissables en λ-calcul
    = fonctions récursives
    = machines de Turing
  - Idée de calcul effectif, formalisation de la calculabilité
  - Machine du Turing = modèle « réaliste » d'ordinateur
  - Pas encore d'ordinateur construit !
  - Existence de fonctions/nombres non calculables
- WWII (1939-45)
  - Mécanisation du chiffrement (Enigma)
  - Turing: déchiffrement (Bombe)
  - Nécessité de calculateurs pour le déchiffrement (Colossus) et pour la balistique
- 1946: ENIAC (calcul balistique): premier ordinateur Turing-complet. Architecture de Von Neuman

Un peu d'histoire

- 1956: Lettre de Gödel à Von Neumann
  - Existence d'un algorithme quadratique pour décider du problème SAT ?
- 1965: Cobham et Edmonds
  - Notion d'algorithme « efficace » (polynomial en temps)
- 1965: Hartmanis et Stearns
  - Certains problèmes sont insolubles sous contrainte de temps
  - Premières hiérarchies de problèmes

Qu'est-ce-qu'un algorithme ?

- Idée intuitive
  - ensemble fini d'instructions élémentaires
  - algorithme: suite finie d'instructions, donnant le résultat en un temps fini
- Formalisation:
  - on fixe formellement
    - un environnement d'exécution des instructions
    - l'ensemble fini des instructions autorisées
    - l'effet de l'exécution de chaque instruction sur l'environnement d'exécution
  - fonction calculable (Church)
    - il existe un algorithme qui la calcule
- Thèse de Church
  - Les fonctions calculables (au sens intuitif) sont exactement celles du λ-calcul
- Plus tard
  - les fonctions définissables en λ-calcul sont exactement les fonctions récursives
  - qui sont exactement celles calculées par les machines de Turing
  - qui sont exactement les solutions des systèmes d'équations diophantiennes (Théorème de Matiiassevitch, 1970, solution négative au 10è problème de Hilbert)
- A la date d'aujourd'hui, les modèles de calcul pour décrire les algorithmes sont encore ceux-ci
- Nous en choisissons un pour définir formellement la notion d'algorithme : les machines de Turing

Ensemble E dénombrable (ou énumérable)

- Les éléments de E peuvent être donnés un par un sans limite de temps
  - Ex: N: 0, 1, 2, 3,…
  - Ex: Q: 0/1, 0/2, 1/1, 0/3, ½, 2/1, 0/4, 1/3, 2/2, 3/1, ...
  - Ex: {a,b}*: a, b, aa, ab, ba, bb, aaa, …
- E peut être mis en bijection avec N
- Indénombrable = non dénombrable

Ensembles indénombrables

- Ex: B^{ω}
- Supposons B^{ω} énumérable.

S	0	1	2	...	i	...
0	$B_{0,0}$	$B_{0,1}$	$B_{0,2}$			
1	$B_{1,0}$	$B_{1,1}$	$B_{1,2}$			
2	$B_{2,0}$	$B_{2,1}$	$B_{2,2}$			
...				...		
i					$B_{i,i}$	
...						

- Soient $w=B_{0,0}B_{1,1}B_{2,2}...B_{i,i}\ldots$ et w' sa négation bit à bit
- Supposons que w' soit dans le tableau, mettons à la position j ($w'=w_j$)
- La $j^{ième}$ lettre de w' est $B_{j,j}$=not $B_{j,j}$: CONTRADICTION !
  Donc w' ne peut pas être dans le tableau.
- C'est l'argument diagonal de Cantor (très utile !)

Ensembles indénombrables et fonctions non calculables

- Mot de B^ω = fonction $N \to B$. Ils sont indénombrables
- Donc les fonctions $N \to N$ aussi
- Programme = nombre fini d'instructions = mot fini sur l'alphabet (fini) des instructions
- Les programmes sont énumérables
- Le nombre de programmes est donc strictement inférieur au nombre de fonctions $N \to N$: toutes les fonctions ne sont pas calculables
- Nombre réel = (u,n) avec u un mot de $[0\text{-}9]^\omega$ et n un entier (la position de la virgule)
- Et de nombreuses fonctions non calculables sont utiles ! Nous en verrons des exemples.

Machines de Turing

Plusieurs modèles de même expressivité
(mais pas nécessairement algorithmiquement équivalents)

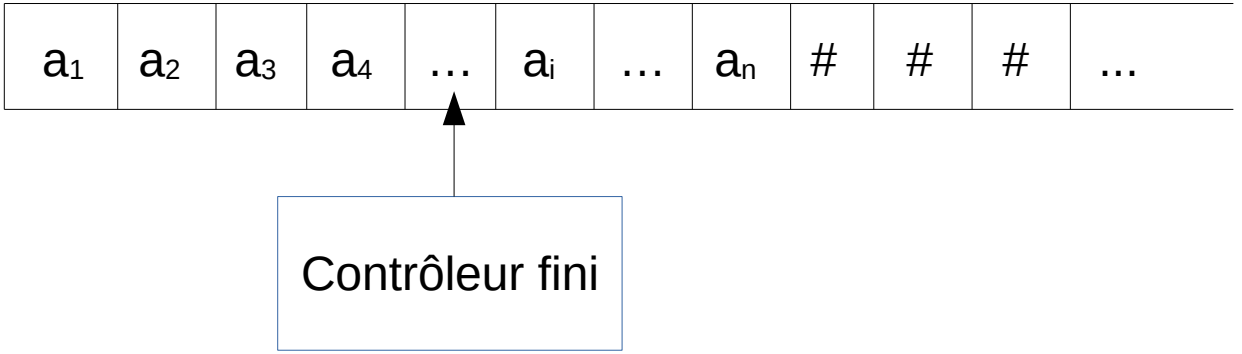
Modèle le plus simple

M=(Q,T,I,#,next,i,F)

- Une bande infinie à droite, divisée en cellules
- Chaque cellule contient un symbole pris dans un alphabet fini T
- Les cellules contenant le symbole spécial # de T sont « vierges »
- Le contrôleur est un automate fini, qui déplace une tête de lecture/écriture sur la bande
- Next: Q*T→Q*T*{G,D} fonction partielle

Machines de Turing: exécution

M=(Q,T,I,#,next,i,F)

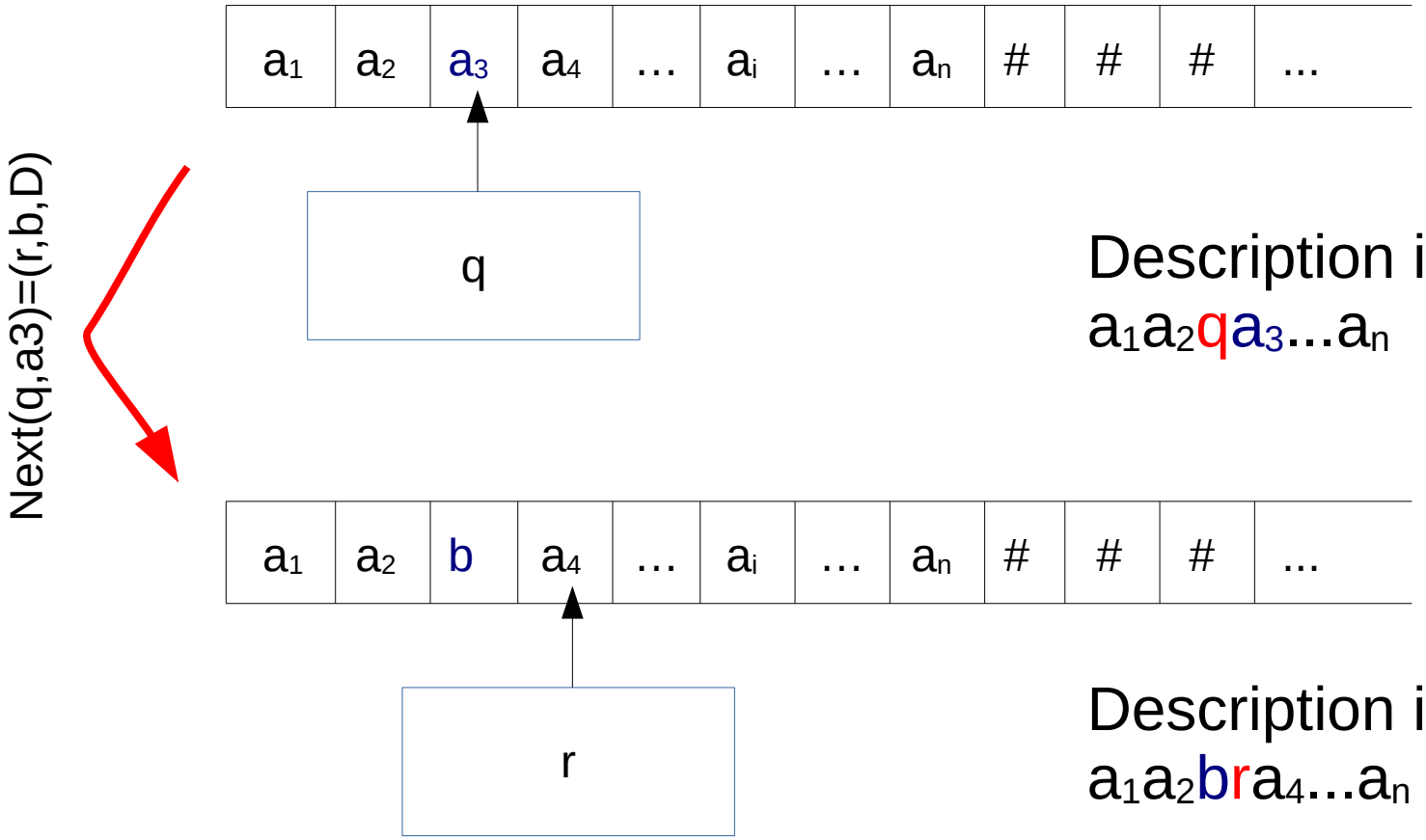


- Au départ,
  - un préfixe fini de la bande est initialisé avec des symboles de la partie I de T, sans #
  - le contrôleur est dans son état I et sa tête est sur la cellule la plus à gauche
- A chaque étape du calcul,
  - le contrôleur lit le symbole sous sa tête.
  - Il le remplace éventuellement par un autre symbole de T
  - Il se déplace d'un cran à gauche ou à droite

    (si demandé et impossible à gauche alors la machine s'arrête sur échec)
- M accepte l'entrée s'il existe une suite d'étapes amenant vers un état final
- M s’arrête quand next n’est pas définie
- Remarques :
  - On peut supposer sans perte de généralité que M s’arrête après être entrée dans un état final
  - Quand l’entrée n’est pas acceptée, M peut s’arrêter (next indéfinie) ou continuer son exécution

Machine de Turing: notation

M=(Q,T,I,#,next,i,F)



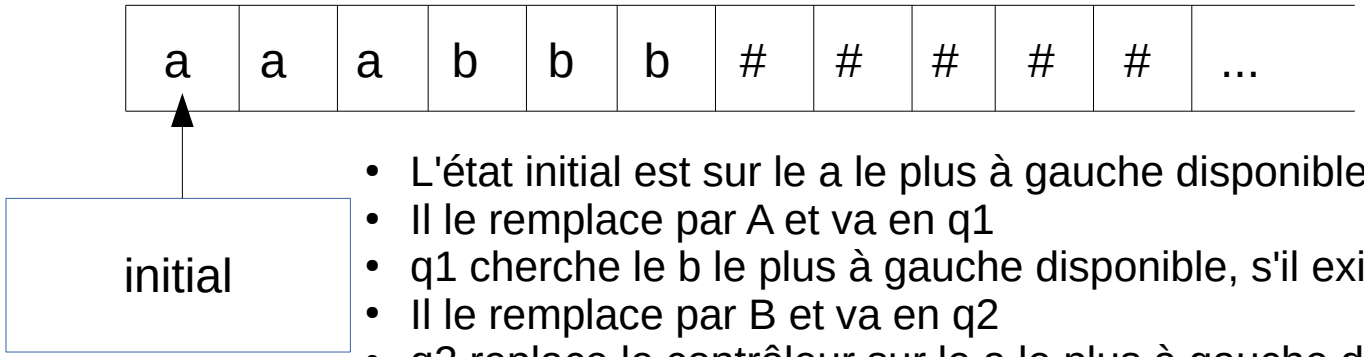
Description instantanée:
$a_1a_2qa_3...a_n$

Description instantanée:
$a_1a_2bra_4...a_n$

$a_1a_2qa_3...a_n \vdash a_1a_2bra_4...a_n$

Machines de Turing comme accepteurs
Exemple: $a^n b^n$

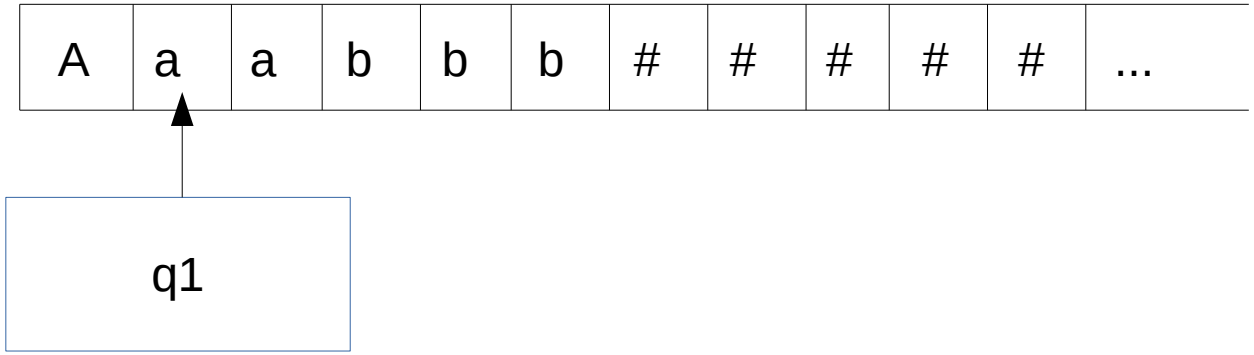
	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



- L'état initial est sur le a le plus à gauche disponible (s'il existe)
- Il le remplace par A et va en q1
- q1 cherche le b le plus à gauche disponible, s'il existe sinon échec
- Il le remplace par B et va en q2
- q2 replace le contrôleur sur le a le plus à gauche disponible sinon sur un A et va en q3
- q3: il n'y a plus de a disponibles, on vérifie qu'il n'y a plus de b non plus

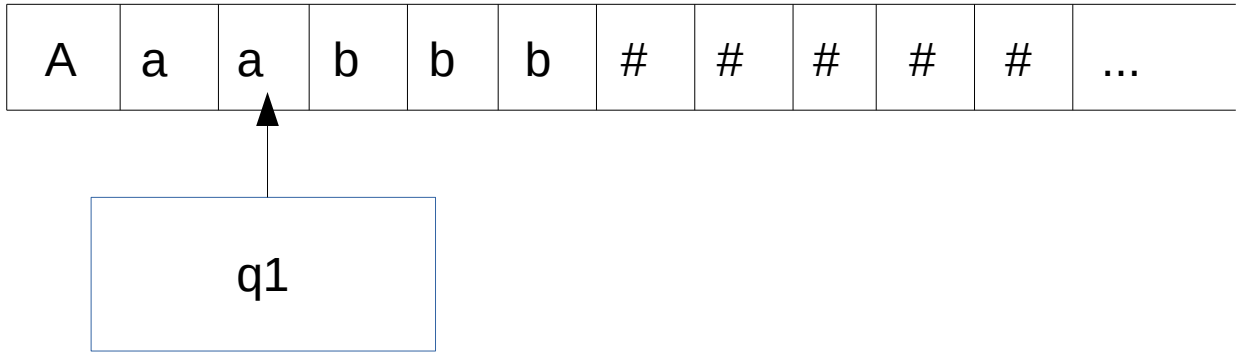
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



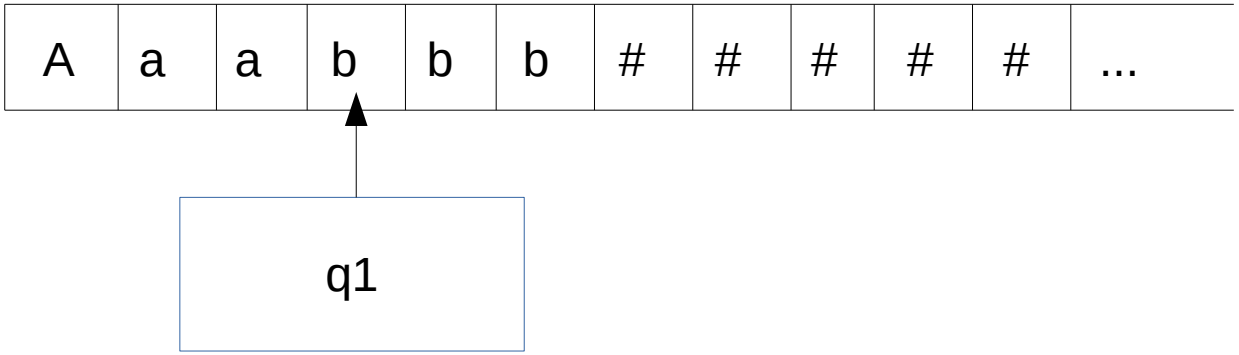
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



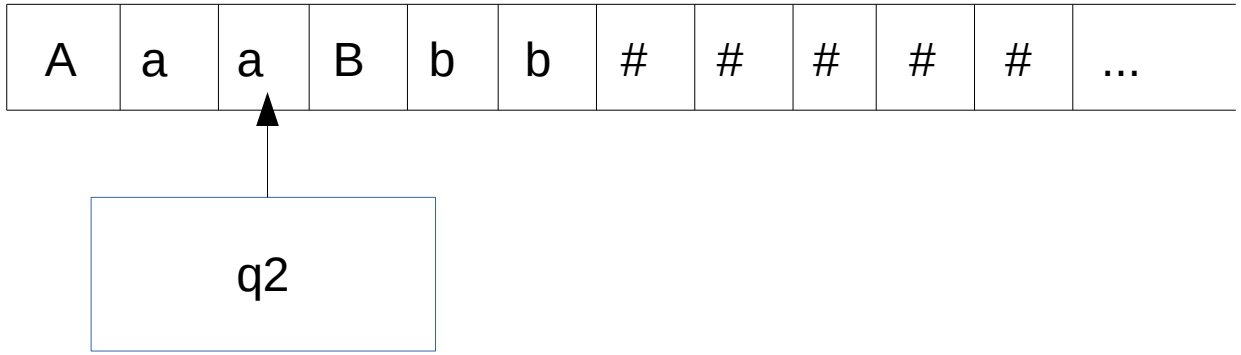
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



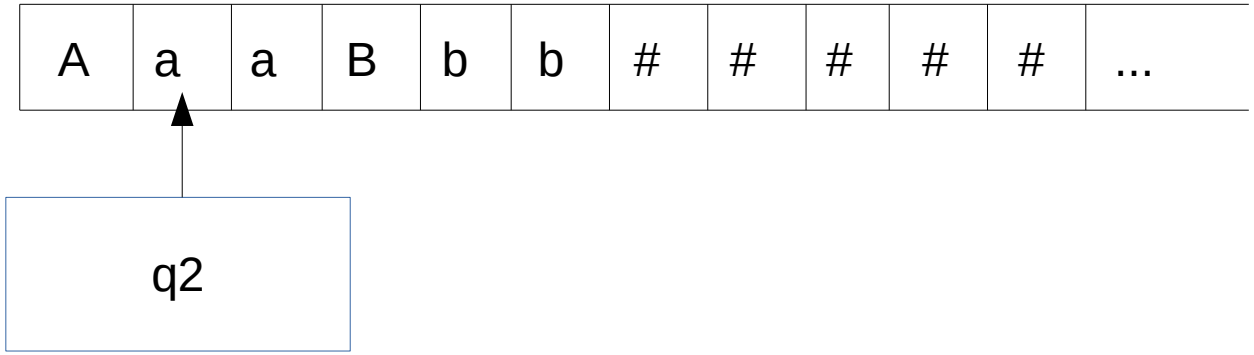
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



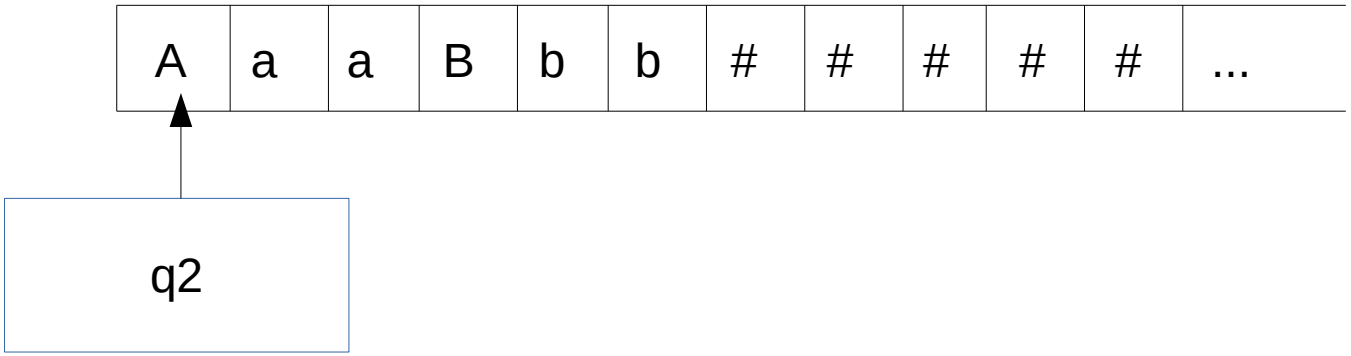
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



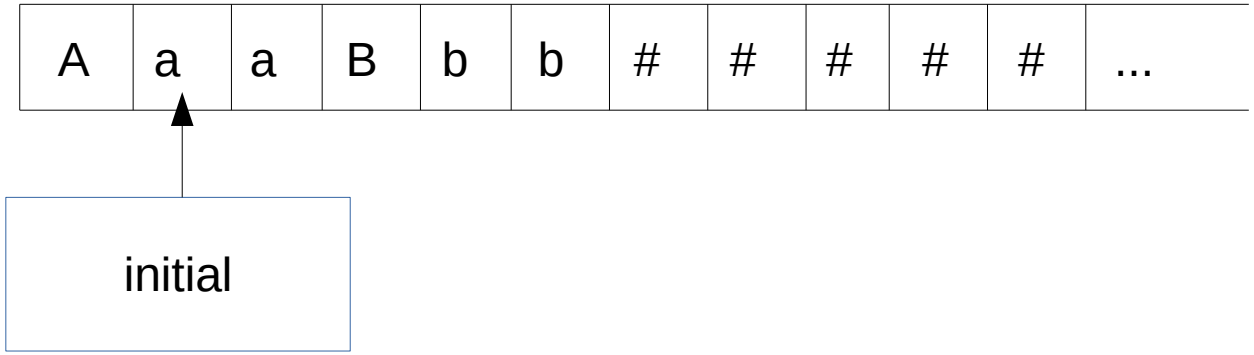
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



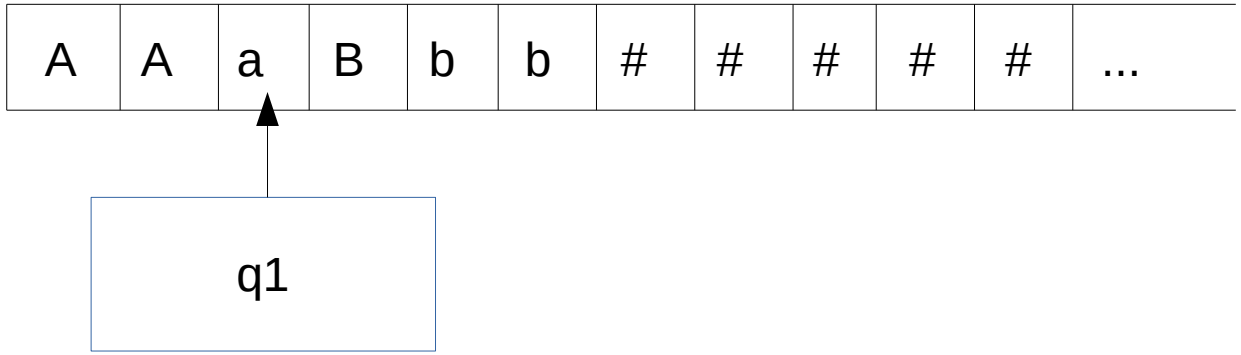
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



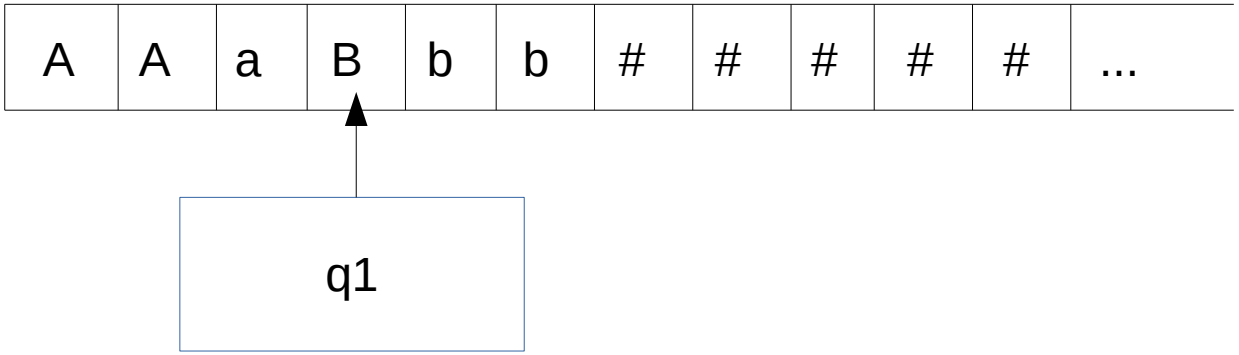
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



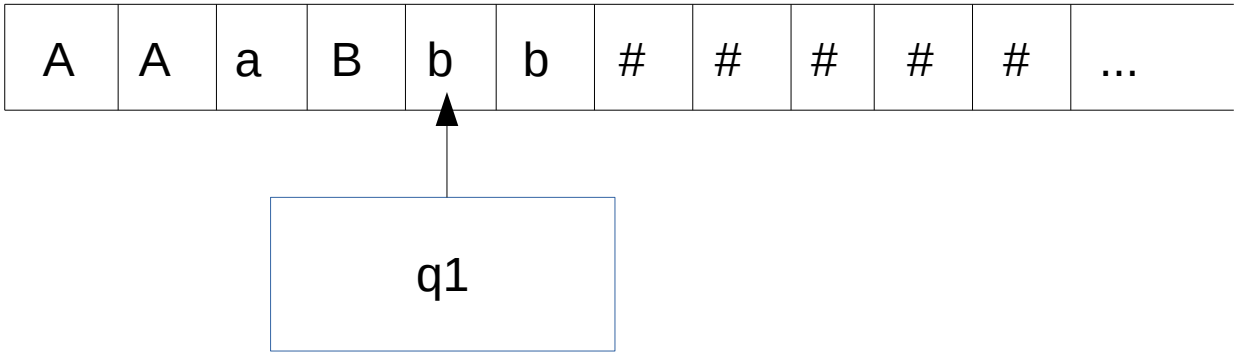
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



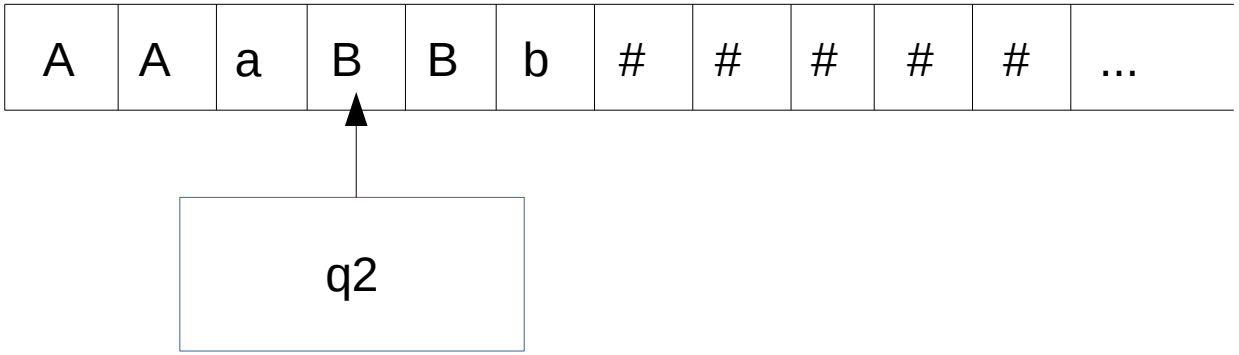
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



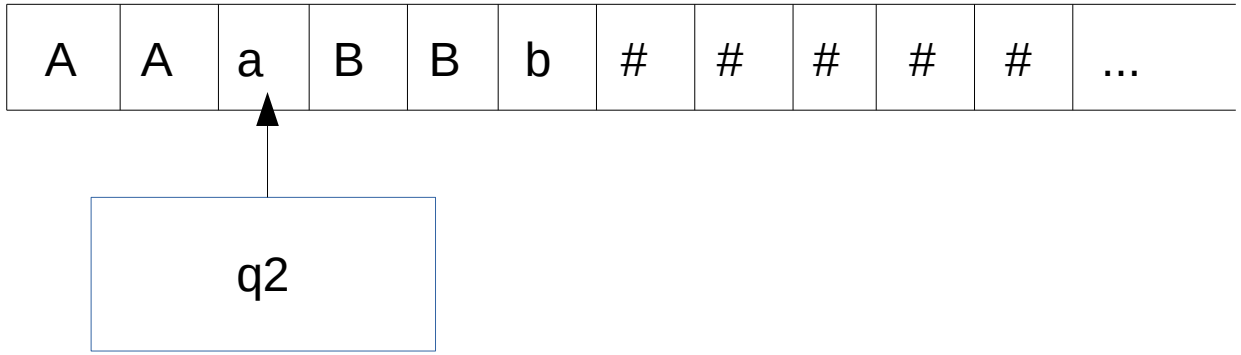
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



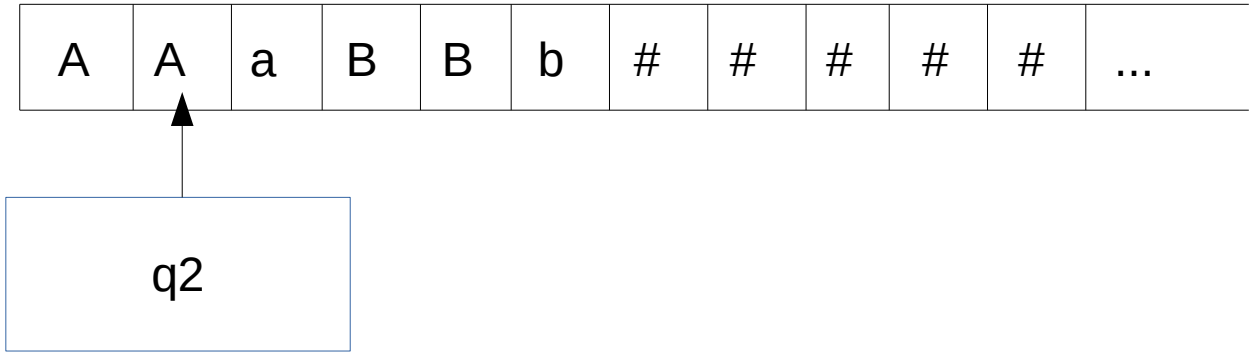
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



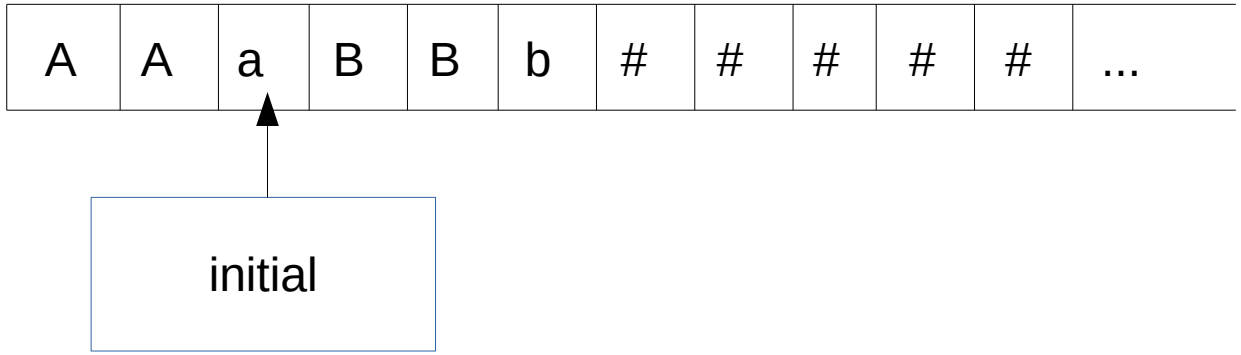
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



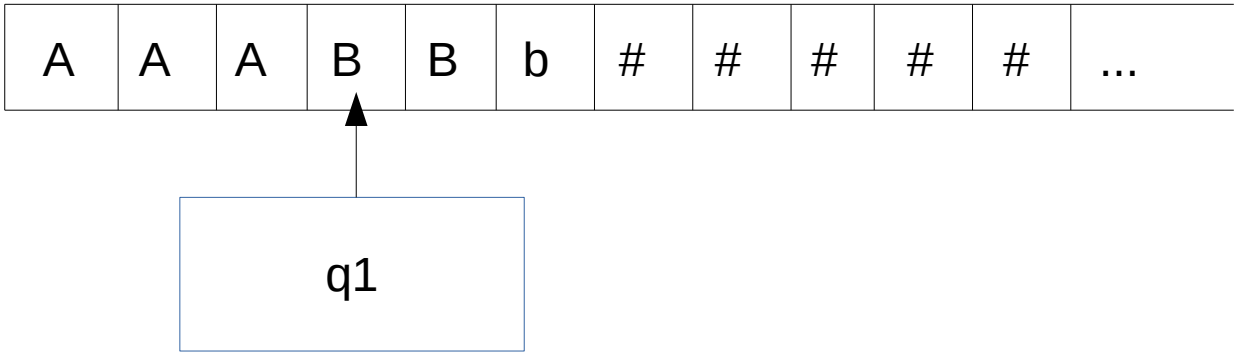
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



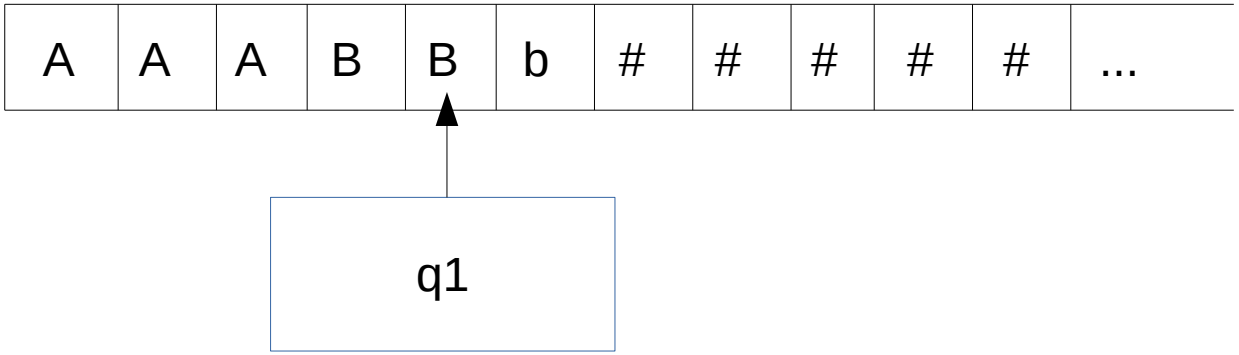
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



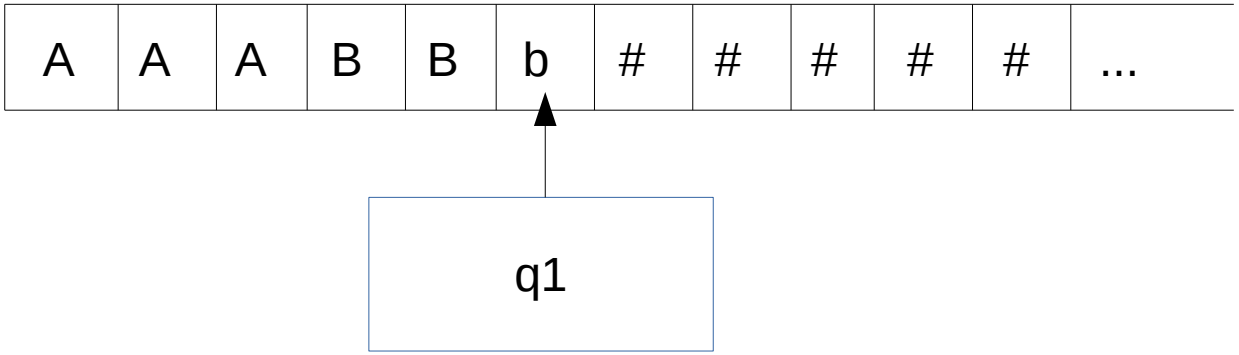
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



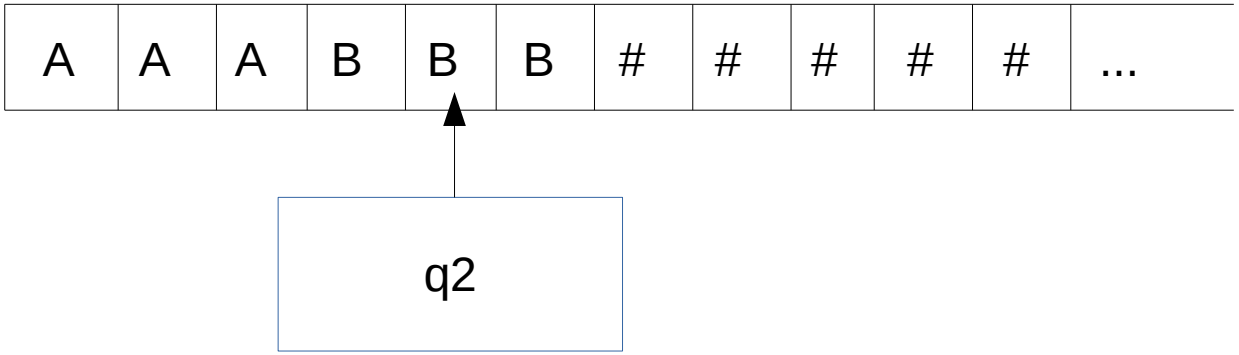
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



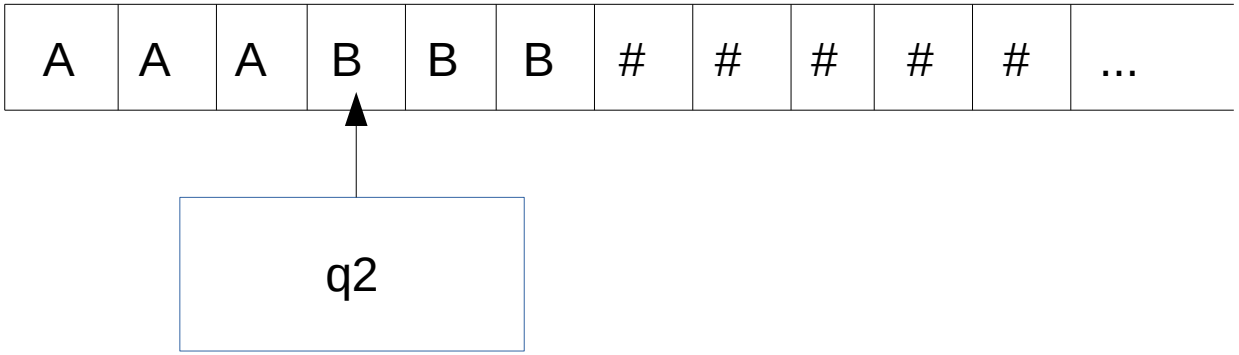
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



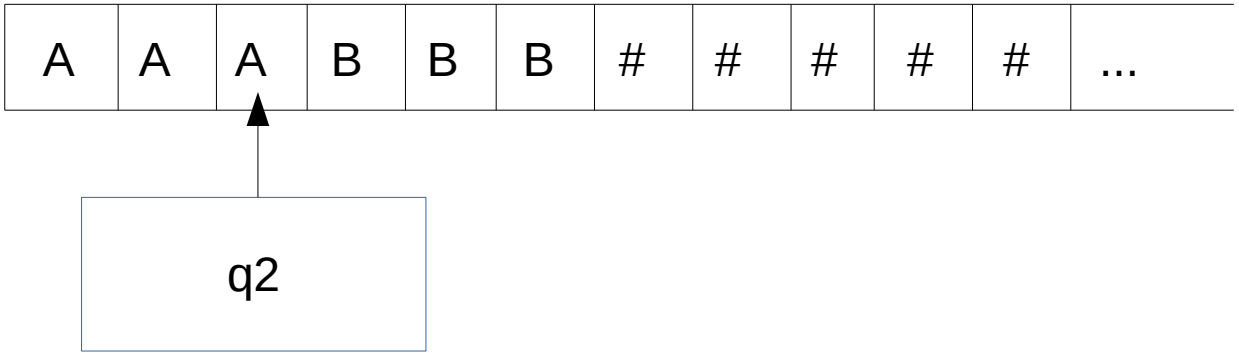
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



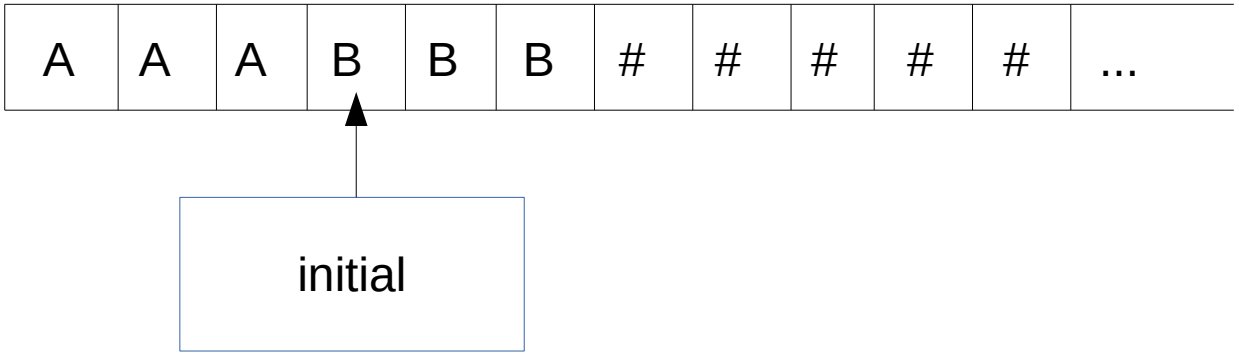
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



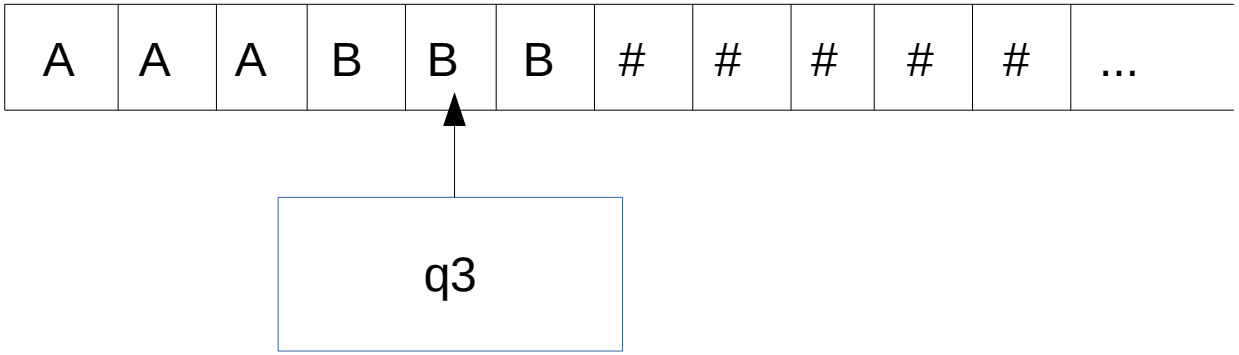
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



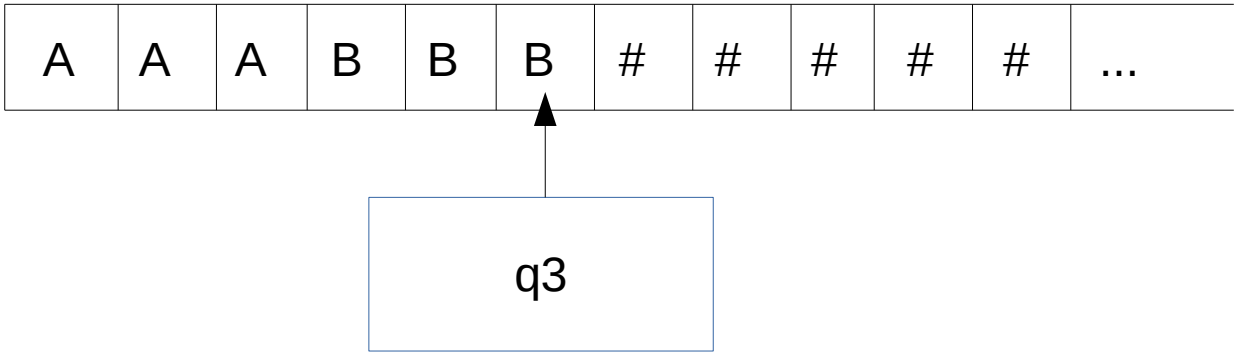
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



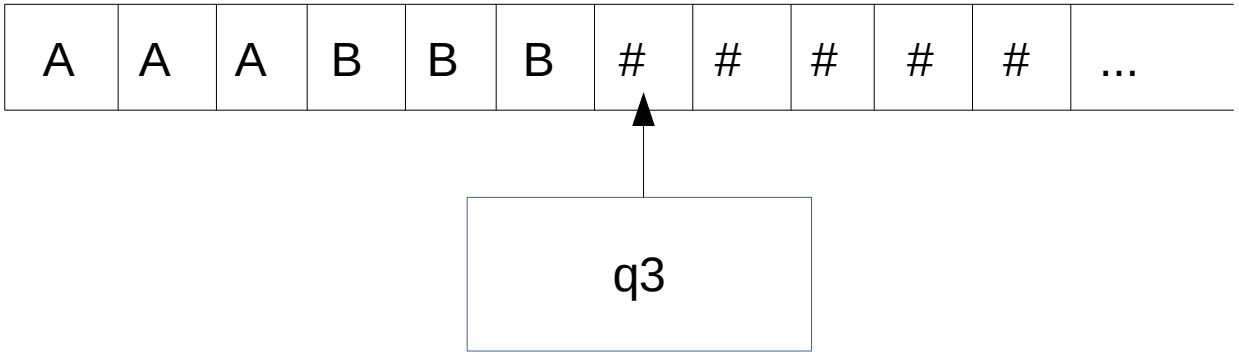
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



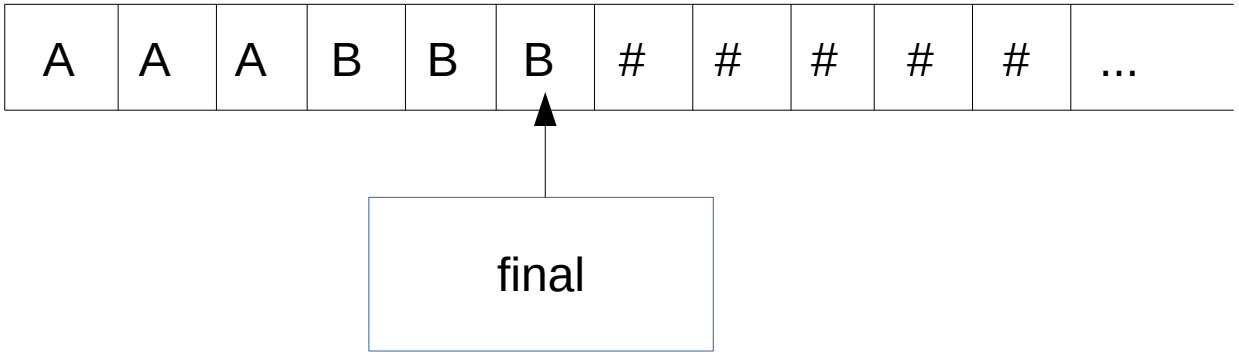
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					



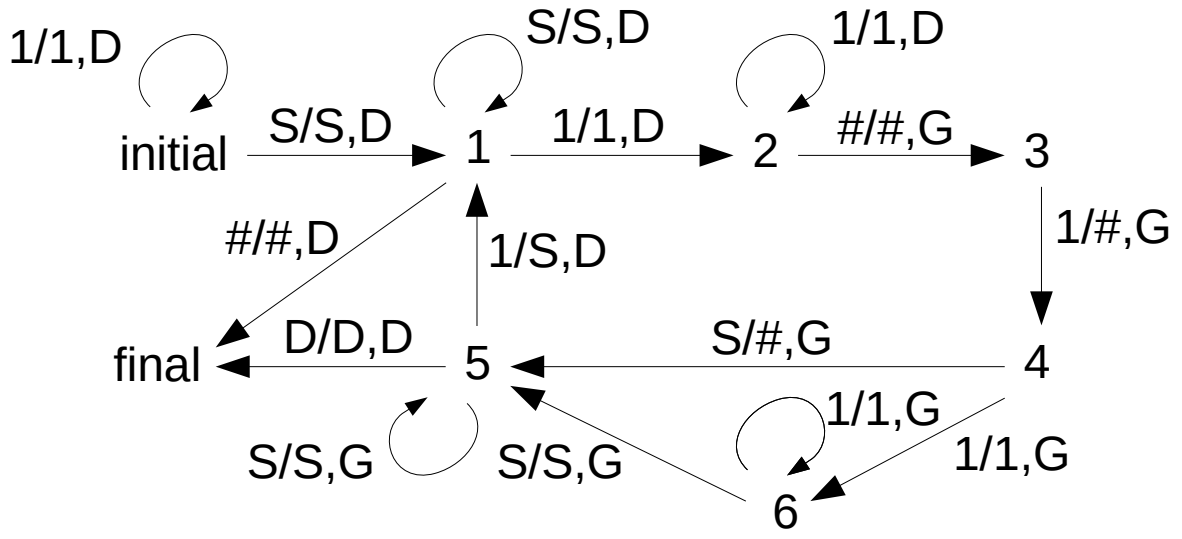
Machines de Turing comme accepteurs
Exemple: $a^n b^n$

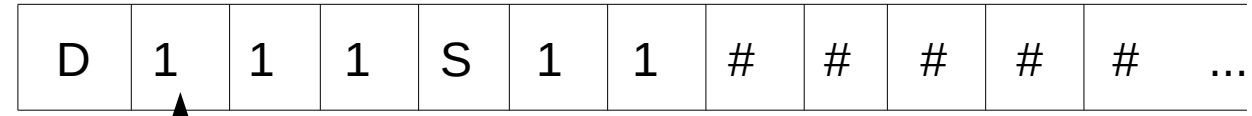
	a	A	b	B	#
initial	(q1,A,D)			(q3,B,D)	(final,#,D)
q1	(q1,a,D)		(q2,B,G)	(q1,B,D)	
q2	(q2,a,G)	(initial,A,D)		(q2,B,G)	
q3				(q3,B,D)	(final,#,G)
final					

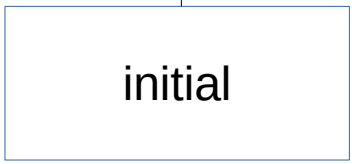


Remarques:
- Cette machine est déterministe
- Elle réalise de l'ordre de $O(n^2)$ mouvements

Machines de Turing comme fonctions
Exemple: n-m avec n≥m







Remarque:
- sortie : $D1^{n-m}S^{m}\#$
- de l'ordre de $O((n+m)^2)$ mouvements

Faire des exécutions sur:
- Dinitial111S11
- Dinitial1S11

Plusieurs définitions équivalentes

1) Bande infinie vers la droite
   Aller vers la gauche quand la tête est sur la case la plus à gauche:
   la machine s'arrête

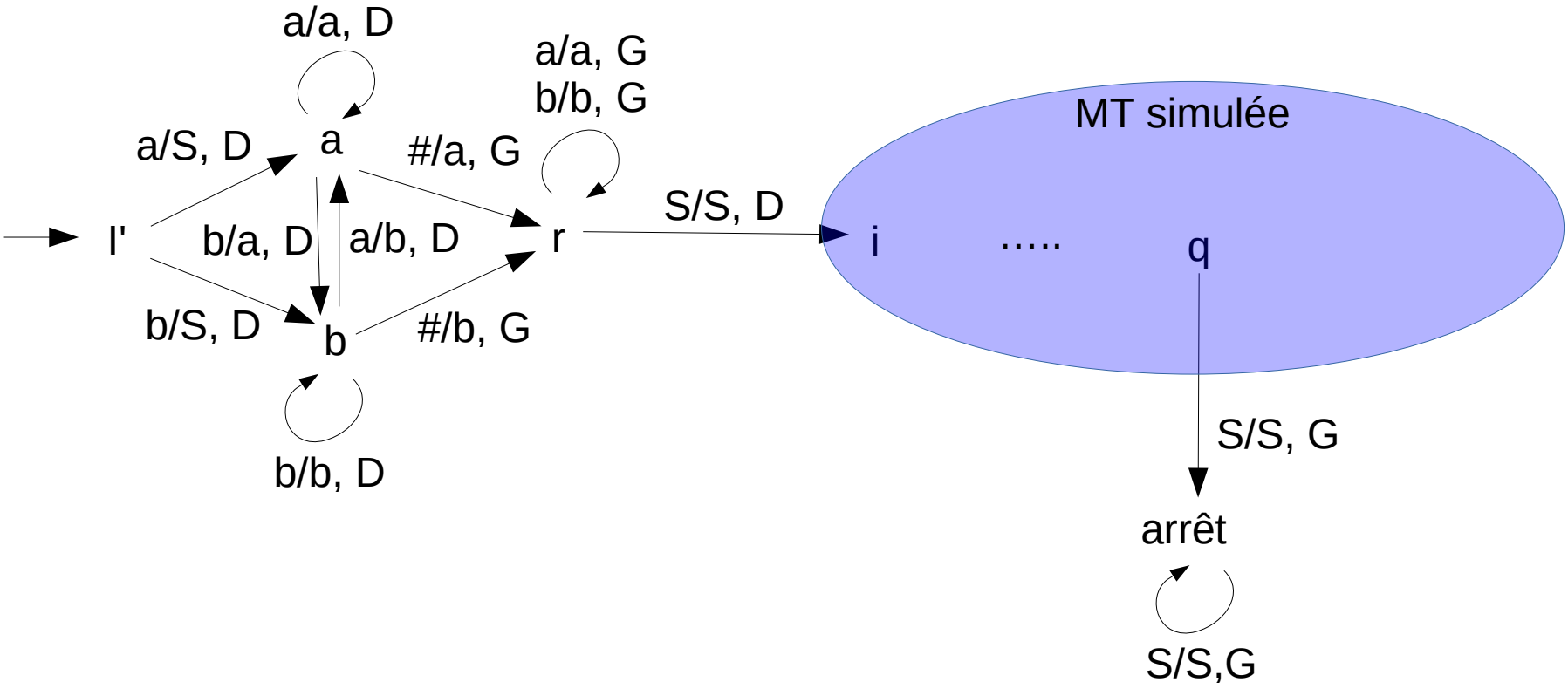
2) Bande infinie vers la droite
   Aller vers la gauche quand la tête est sur la case la plus à gauche:
   la tête ne se déplace pas, elle reste sur la case la plus à gauche, et le calcul continue

3) Bande bi-infinie (infinie à droite et à droite)
   La bande est de la forme
   ……..####ababababbbb####……..
   ^
   La tête se trouve initialement sur le symbole le plus à gauche qui n'est pas #
   Elle peut toujours se déplacer, sans problème, vers la droite ou vers la gauche

Il existe beaucoup d'autres modèles, tous équivalents
(chacun permet de simuler tous les autres)

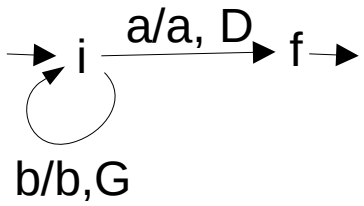
Simuler 1) avec 2)

i. Pour pouvoir détecter le début de bande, on insère un symbole spécial S sur la première case (il faut décaler l'entrée d'une case à droite)
ii. Le contrôle est passé à la machine à simuler
iii. Simulation de l'échec de tentative d'aller à gauche quand c'est impossible: nouvel état puits



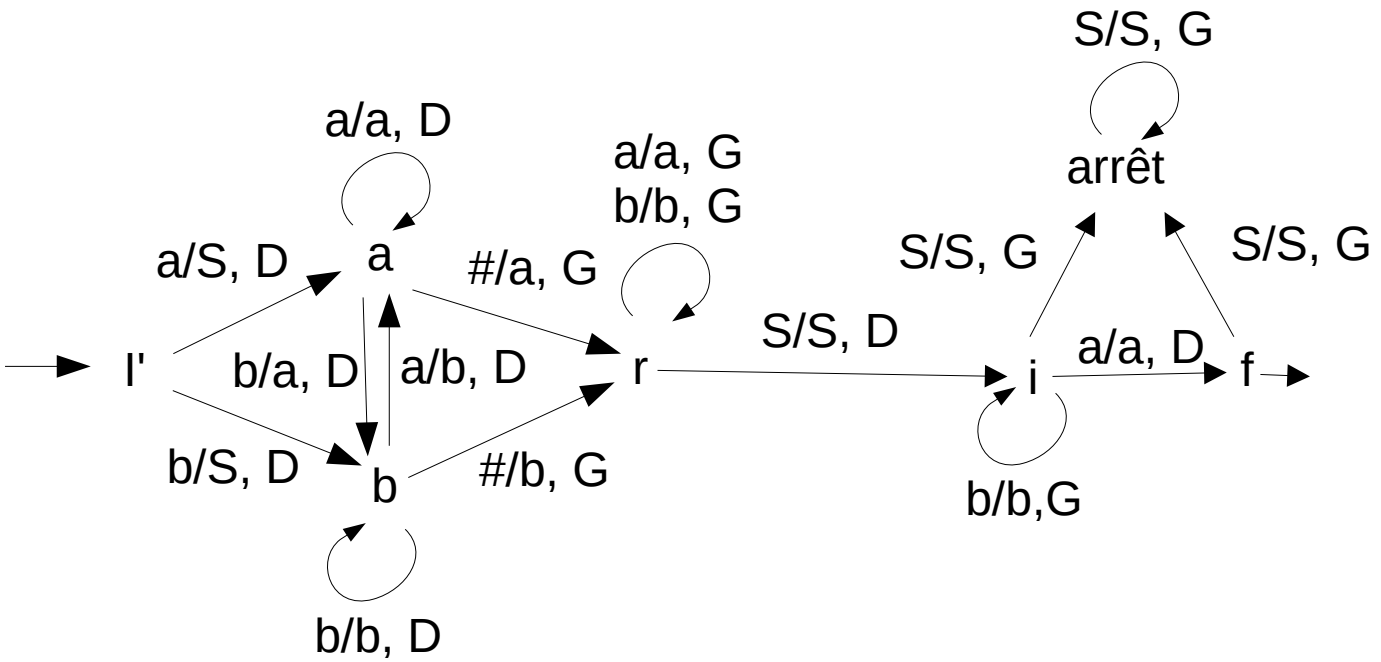
Simuler 1) avec 2) : exemple

A={a,b}, T={a,b,#}, L=aA*



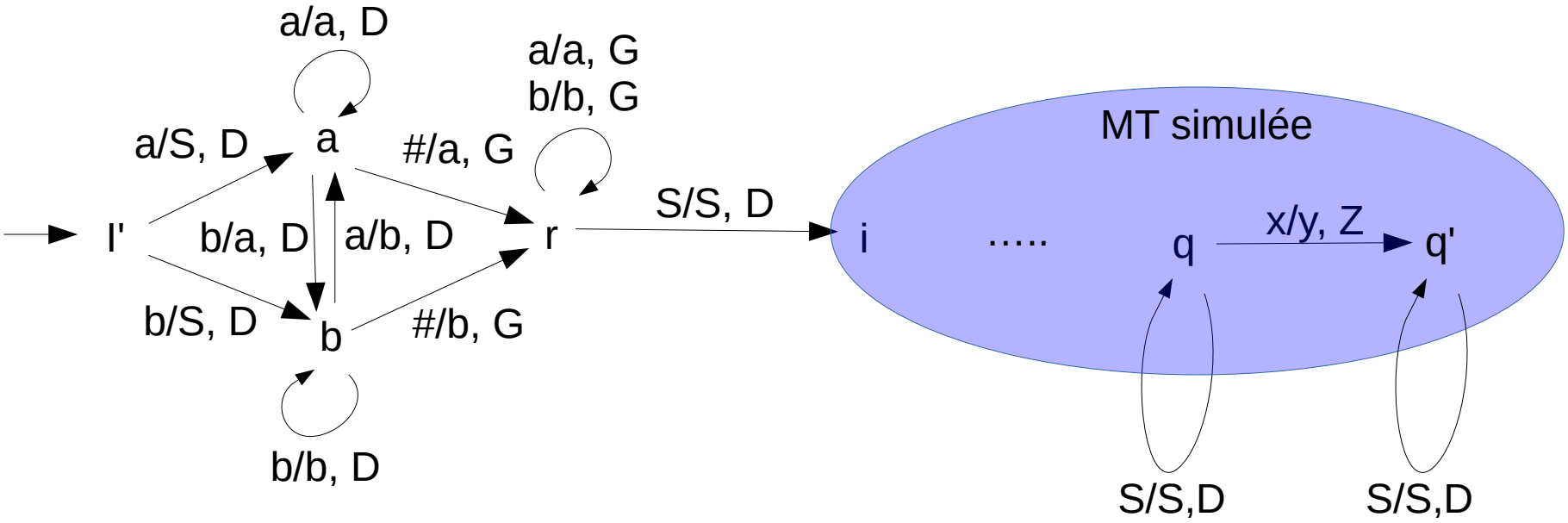
Première lettre = b: on va à gauche, arrêt
Première lettre = a: on va à droite, succès
peu importe le reste

Simulation avec une MT qui ne s'arrête pas en allant à gauche de la première case:



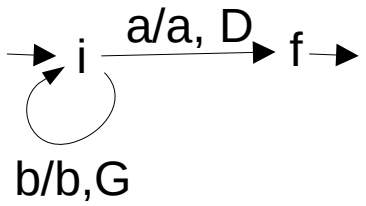
Simuler 2) avec 1)

i. Pour pouvoir détecter le début de bande, on insère un symbole spécial S sur la première case (il faut décaler l'entrée d'une case à droite)
ii. Le contrôle est passé à la machine à simuler
iii. Simulation de tentative d'aller à gauche quand c'est impossible: arrivée sur S → aller à droite



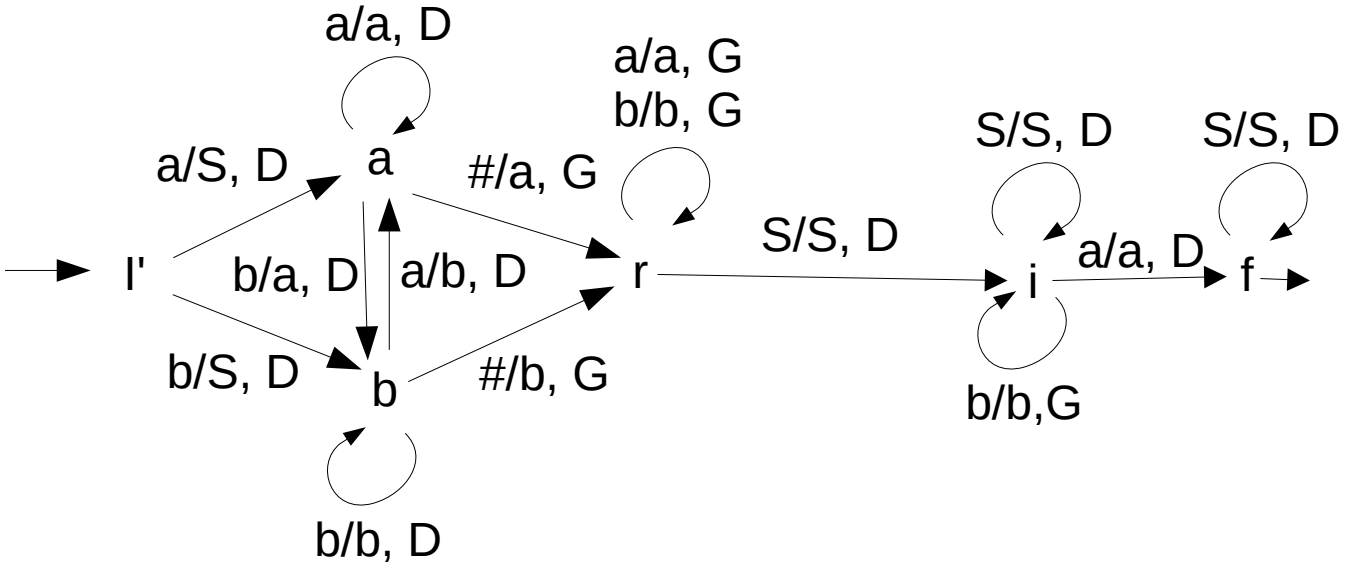
Simuler 2) avec 1) : exemple

A={a,b}, T={a,b,#}, L=aA*



Première lettre = b: on va à gauche (et donc, on fait du sur place)
Première lettre = a: on va à droite, succès peu importe le reste

Simulation avec une MT qui s'arrête en allant à gauche de la première case:



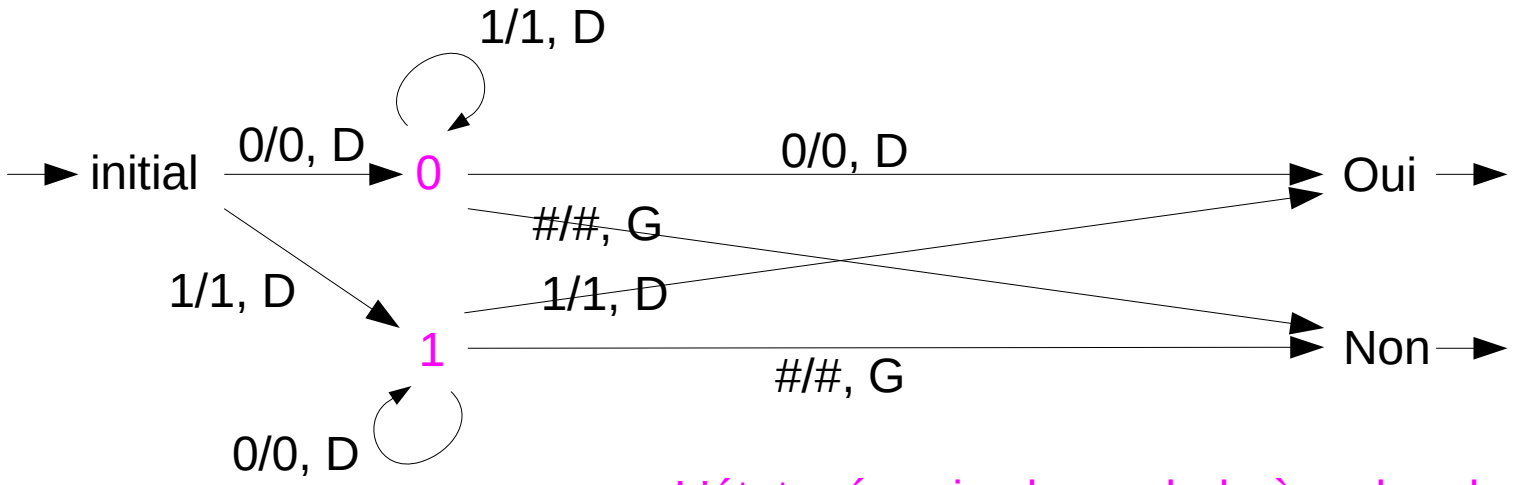
Arrêt des machines de Turing

- Un langage L est
  - *Récursif* s'il existe une MT M telle que L(M)=L et M s'arrête sur toute entrée
  - *Récursivement énumérable* s'il existe une MT M telle que L(M)=L et M s'arrête sur tout mot de L(M) (mais pas nécessairement sur toute entrée)
  - Par définition, les langages des MT sont récursivement énumérables
- Une fonction f est
  - *Récursive totale* s'il existe une MT M telle que M s'arrête sur un état final avec $f(n_1,\ldots,n_k)$ sur sa bande pour tout $n_1,\ldots,n_k$ (donc, f est définie pour tous ses arguments)
  - *Récursive partielle* s'il existe une MT M telle que M s'arrête sur un état final avec $f(n_1,\ldots,n_k)$ sur sa bande si $f(n_1,\ldots,n_k)$ est définie (donc, M ne s'arrête pas nécessairement si $f(n_1,\ldots,n_k)$ n'est pas définie).

Quelques techniques de construction

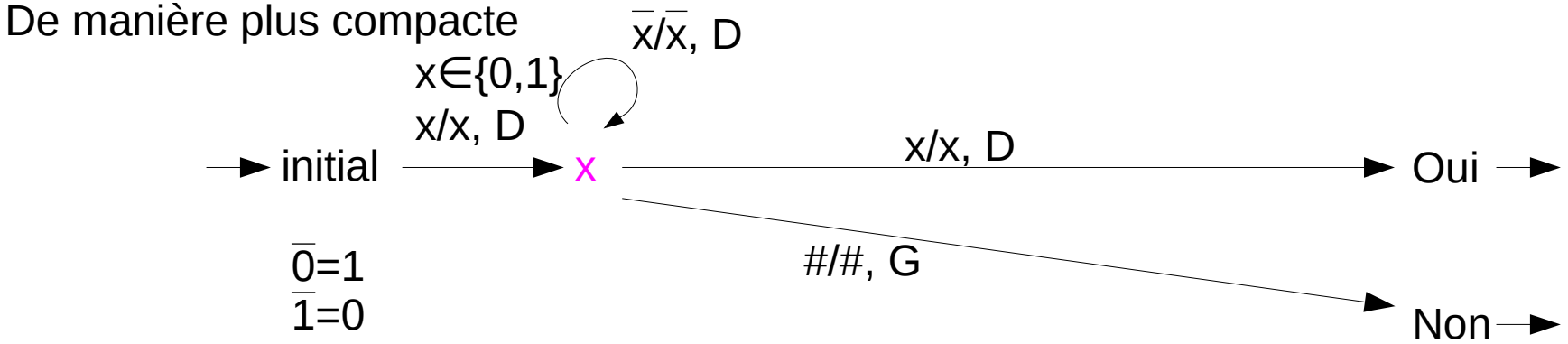
Stocker une information ***bornée*** dans les états

MT vérifiant si le premier symbole de la bande apparaît autre part



L'état mémorise le symbole à rechercher

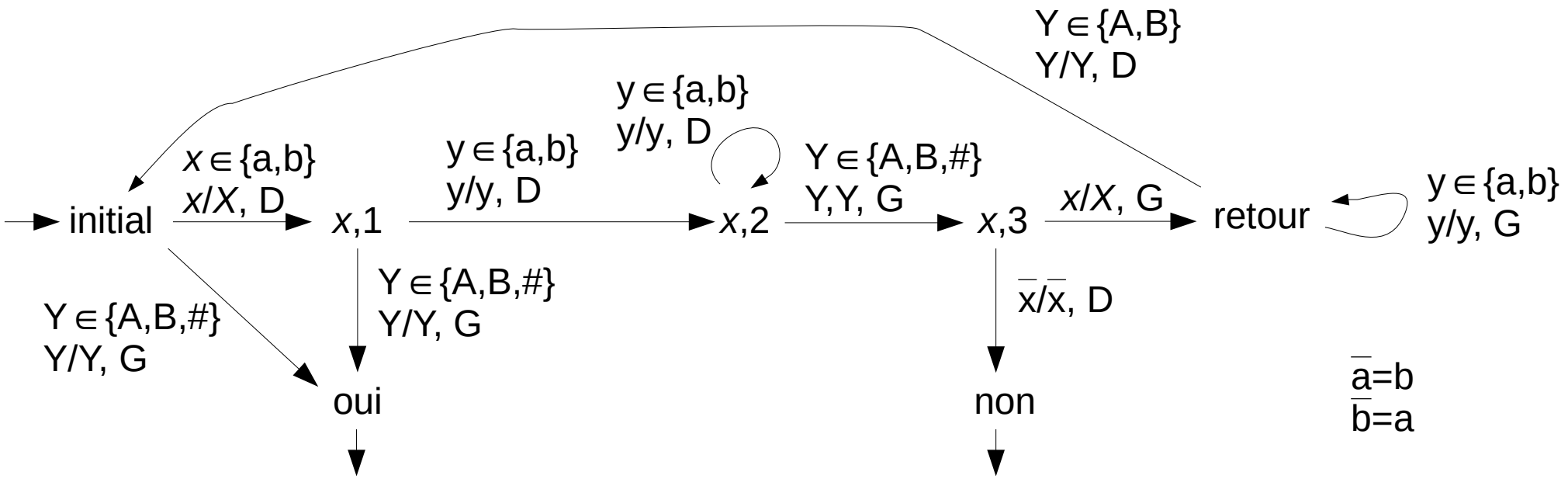
De manière plus compacte



Quelques techniques de construction

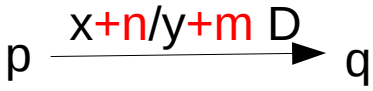
Extension de l'alphabet de bande

MT vérifiant si l'entrée sur {a,b} est un palindrome: T={a,b,A,B,#}



Quelques techniques de construction

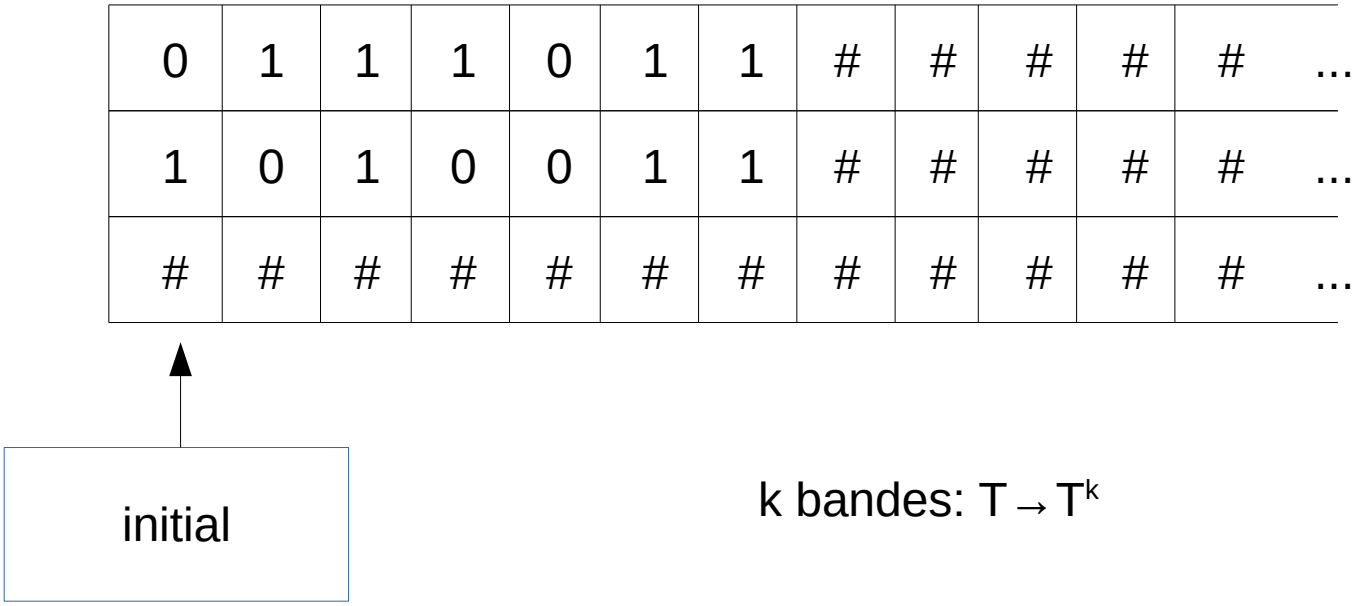
Extension des transitions



On peut combiner, et imaginer d'autres extensions !

Quelques techniques de construction

MT multi-bandes mono-tête



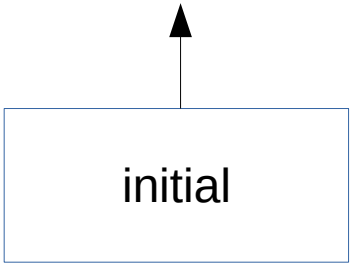
k bandes: $T \rightarrow T^k$

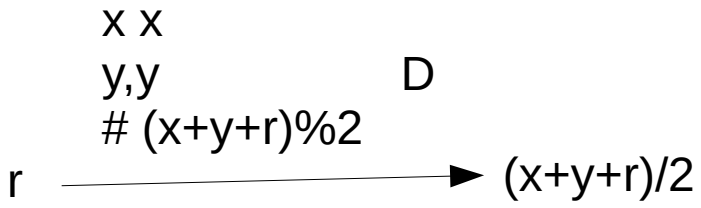
Quelques techniques de construction

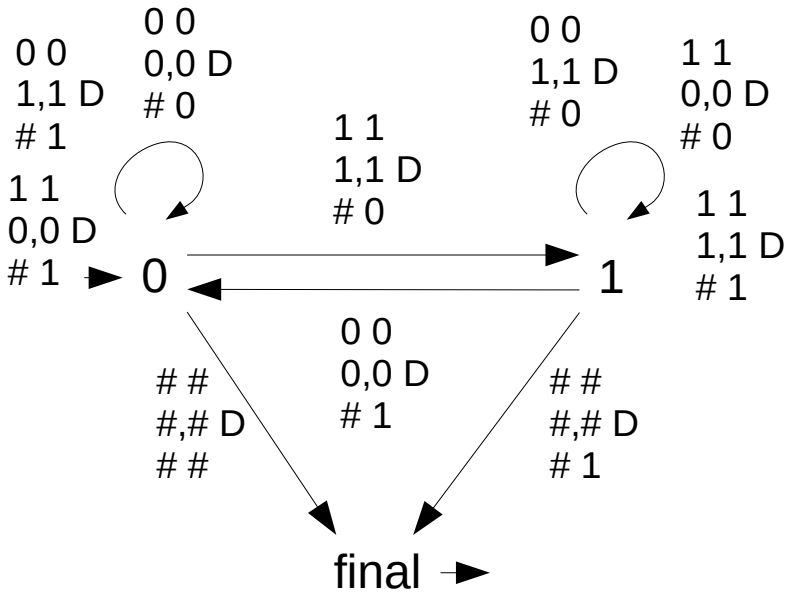
MT multi-bandes mono-tête

MT réalisant la somme de deux nombres binaires

0	1	1	1	0	1	1	#	#	#	#	#	...
1	0	1	0	0	1	1	#	#	#	#	#	...
#	#	#	#	#	#	#	#	#	#	#	#	...

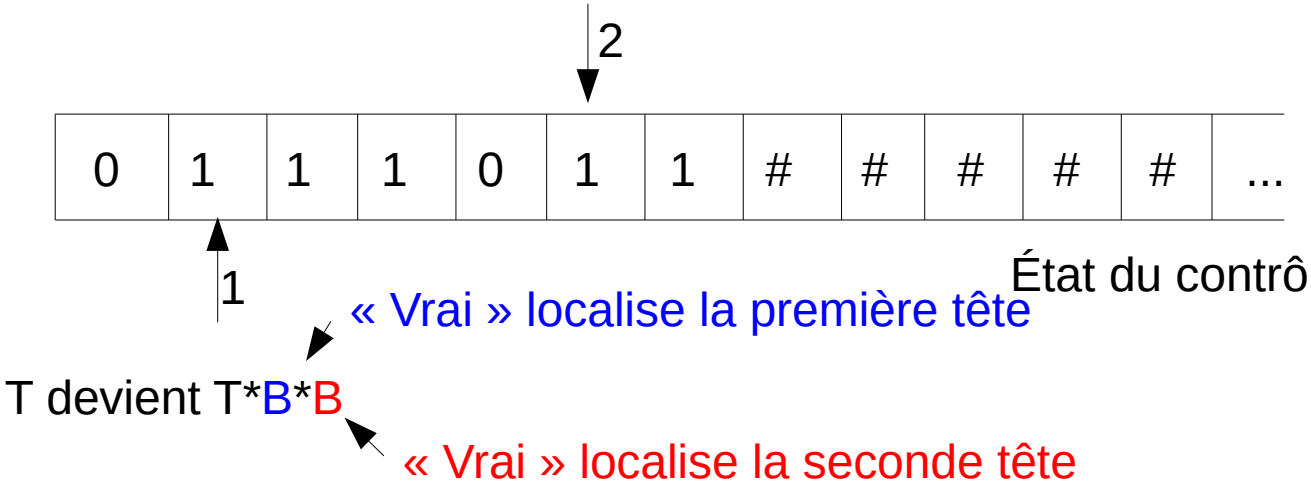


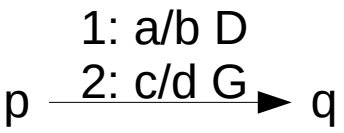




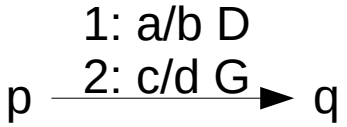
Quelques techniques de construction

MT multi-têtes, mono-bande





Simulation de



Etats: $(q, v_1, b_1, v_2, b_2, X)$ avec

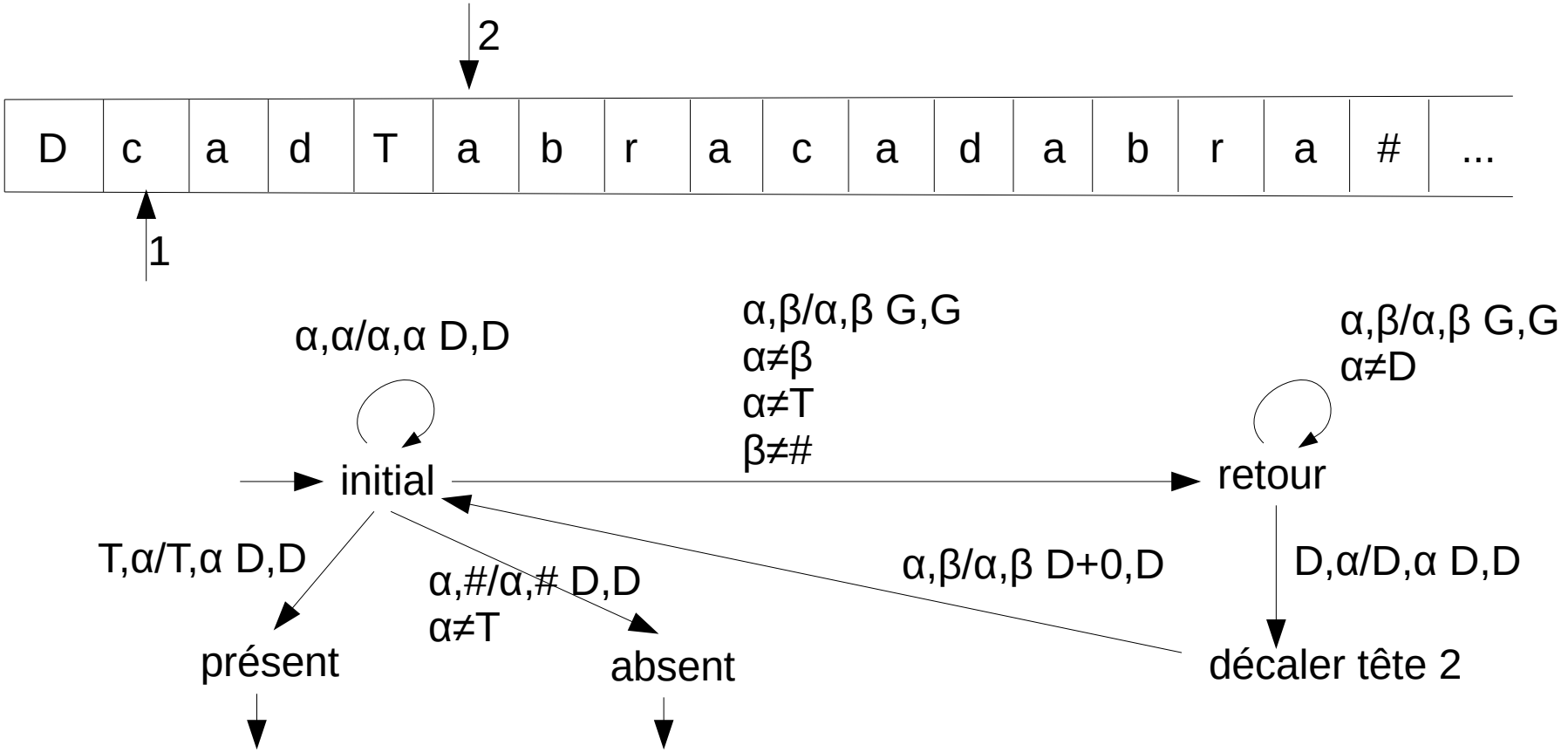
- q: état de la machine à simuler
- v_i: espace où stocker la valeur sous la ième tête
- b_i: booléen indiquant si on peut utiliser v_i
- X une direction: D ou G

Simuler un mouvement:

- On suppose que toutes les têtes à simuler toutes du même coté (D ou G) de la position courante de la tête
- On parcourt la bande dans la direction X tant que $b_1 \wedge b_2$ faux
  - pour toute tête t_i à simuler rencontrée
    - on mémorise le symbole sous t_i dans v_i et on met b_i à vrai
- Comme on sait maintenant ce qu'il y a sous chaque tête, on connaît l'action à réaliser
- On met tous les b_i à faux
- On parcours la bande dans la direction opposée à X tant que $b_1 \wedge b_2$ faux
  - pour toute tête t_i rencontrée
    - on réalise l'action pour t_i et on met b_i à vrai
- On met tous les v_i à faux

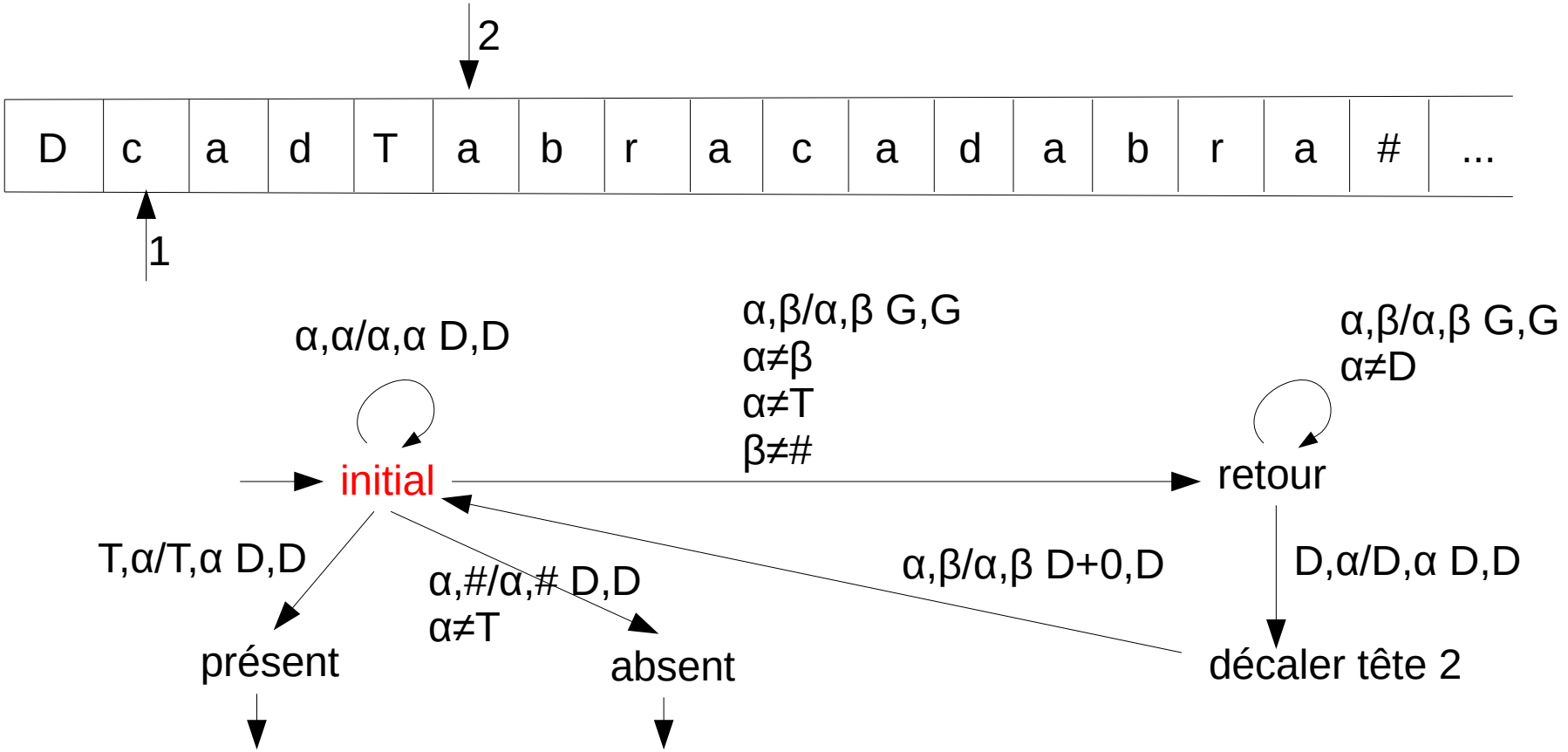
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



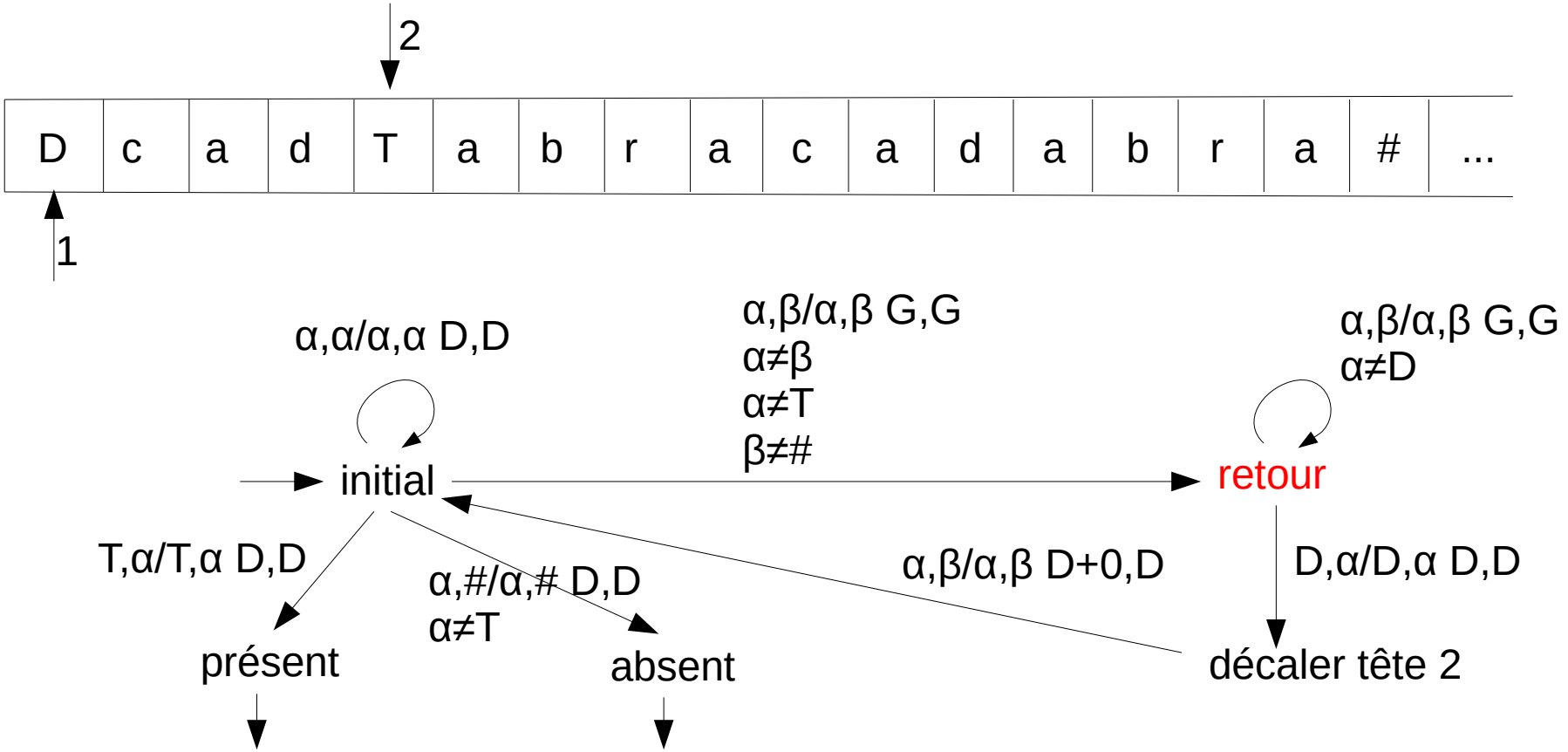
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



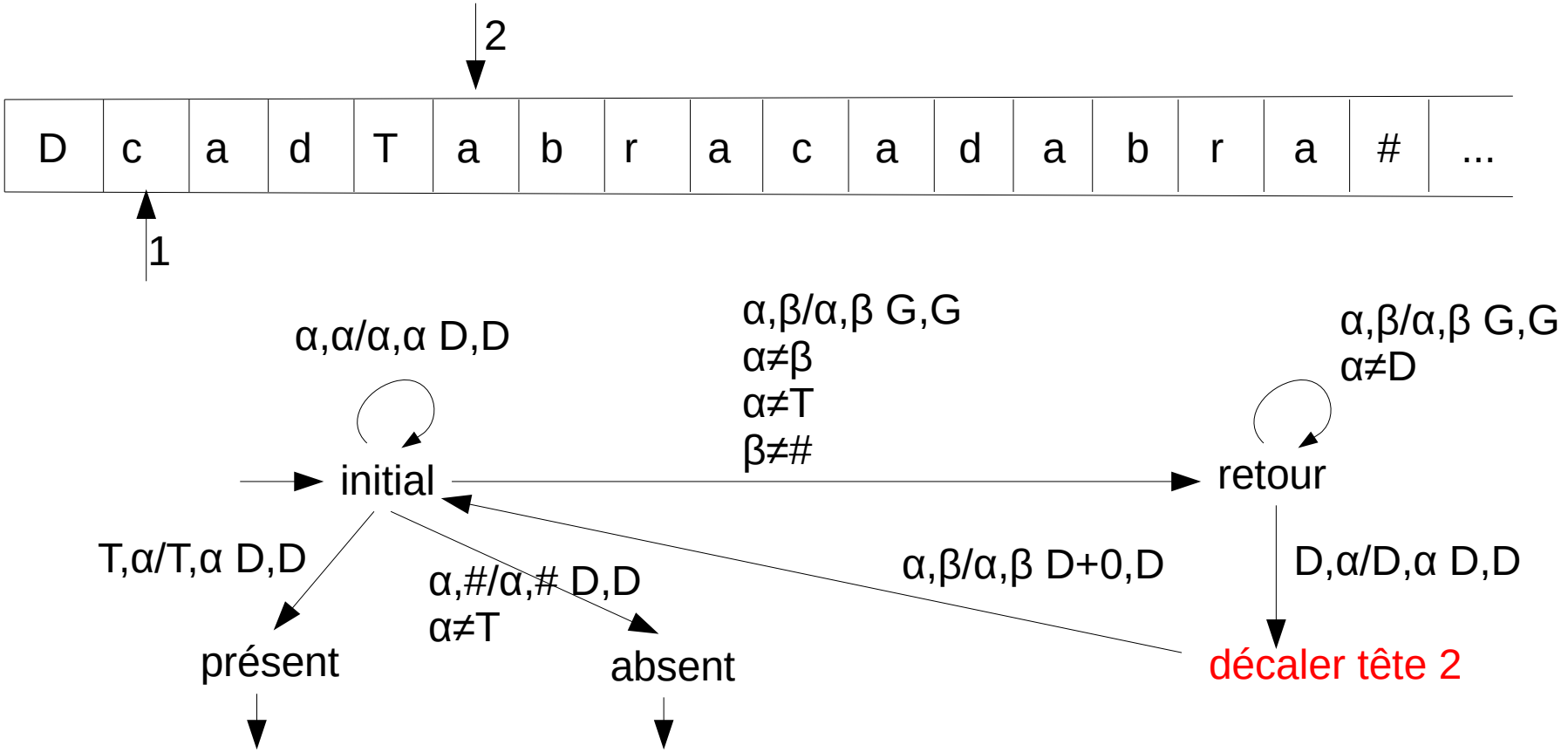
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



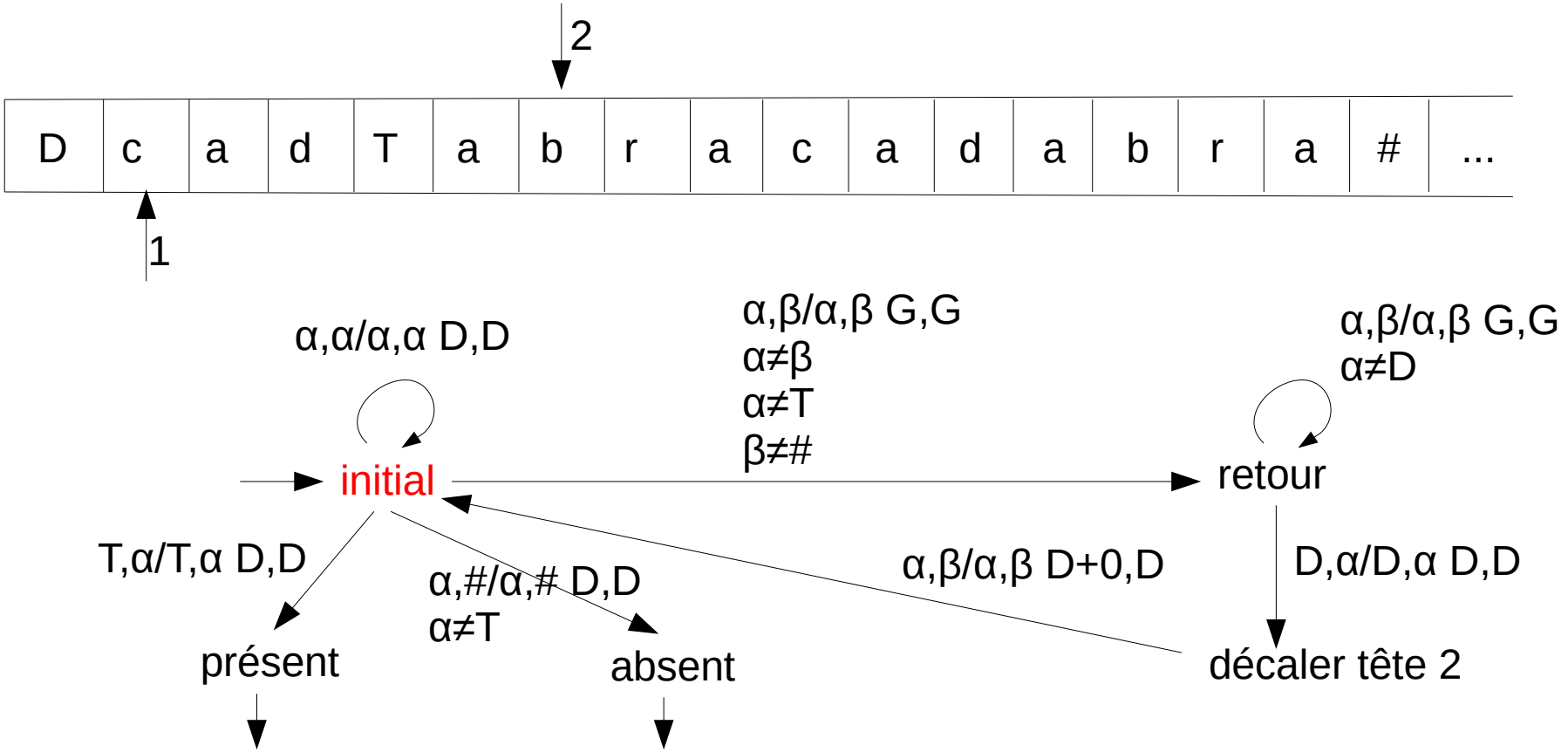
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



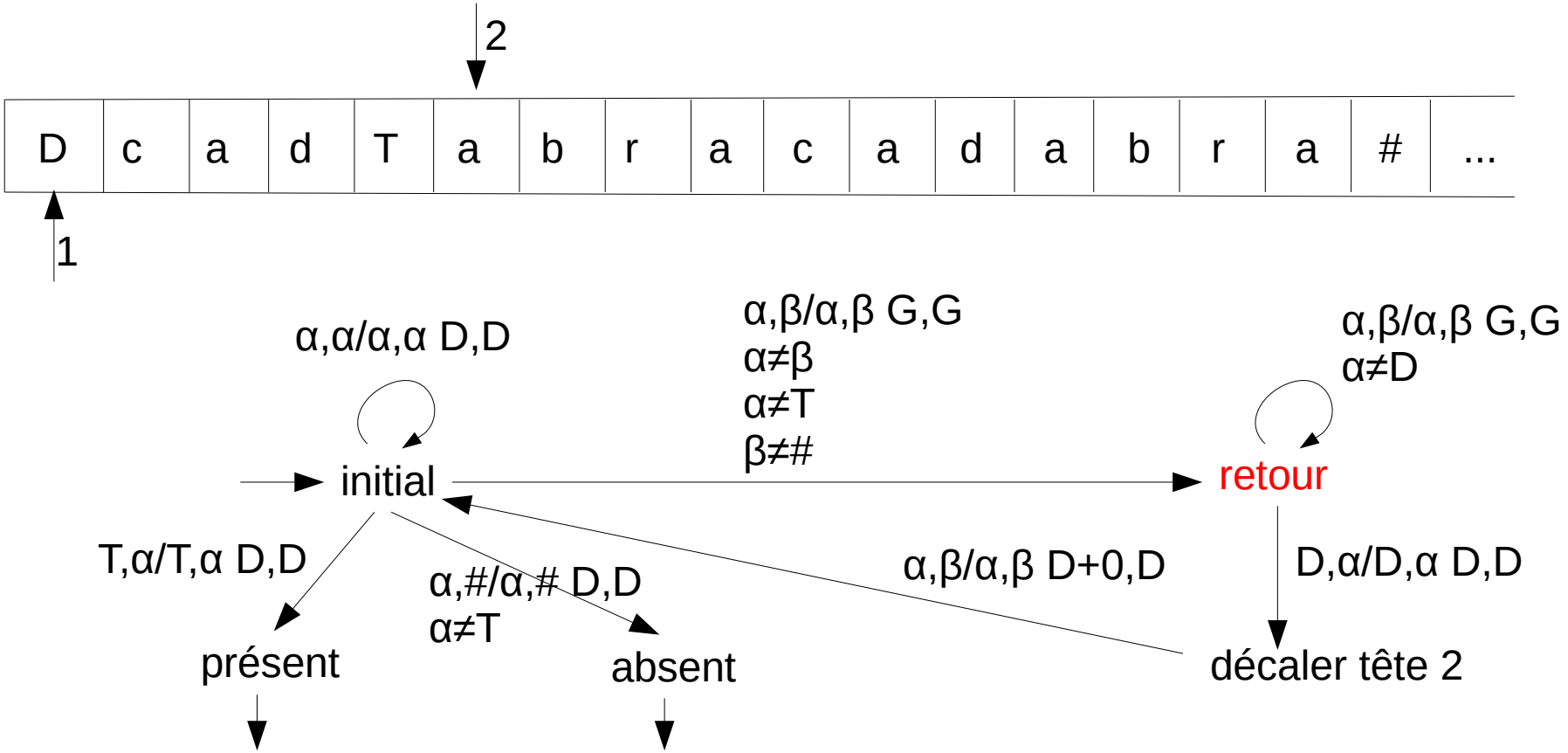
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



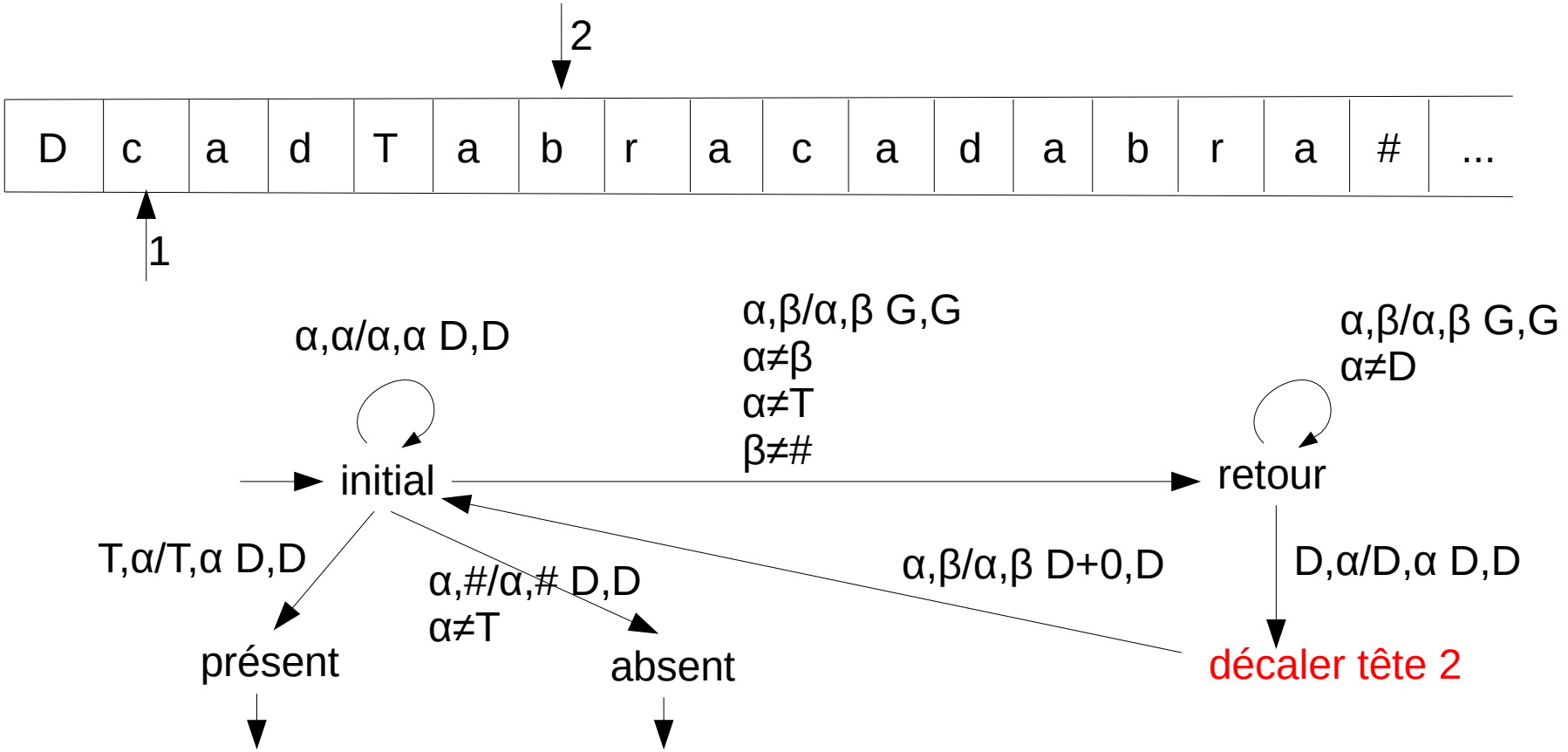
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



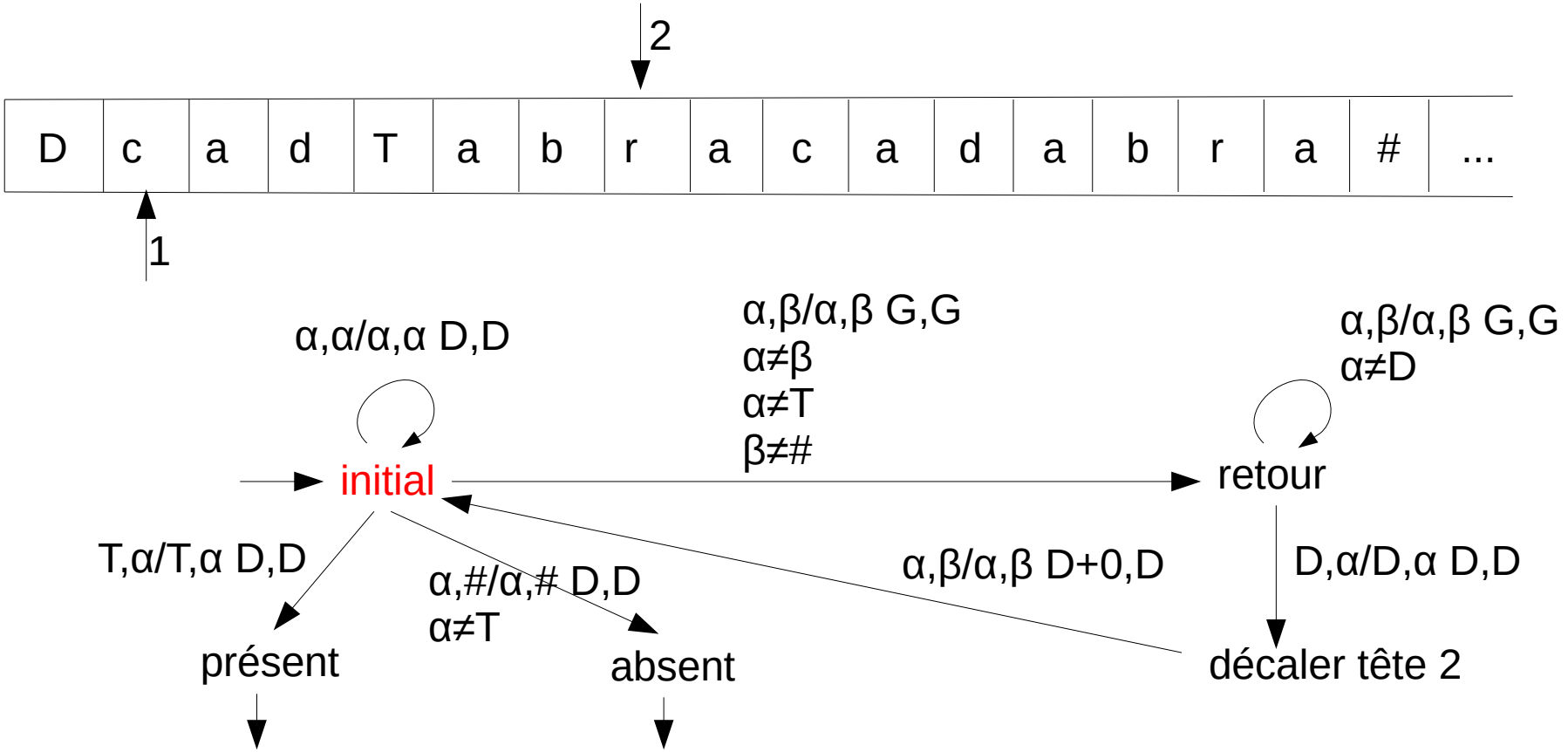
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



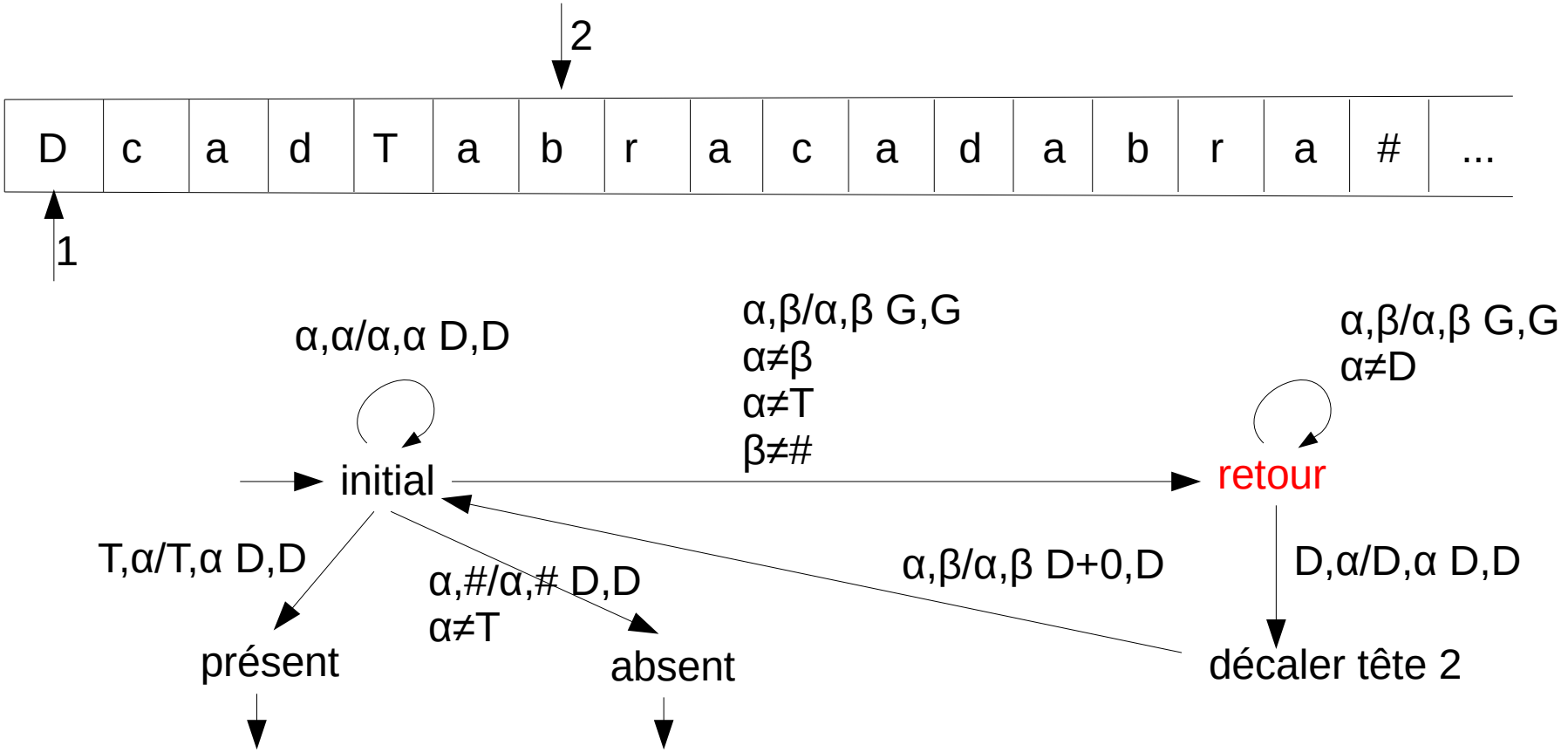
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



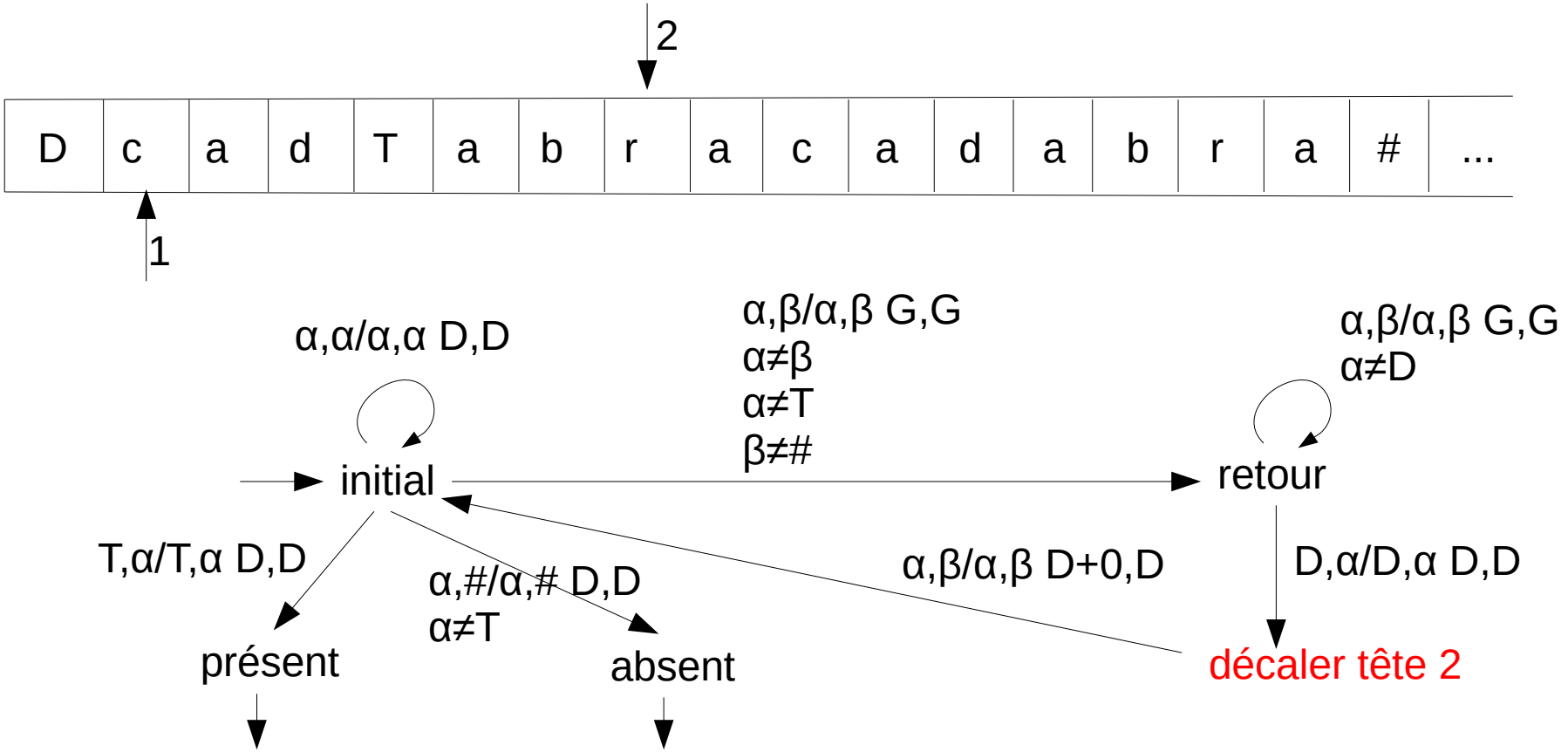
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



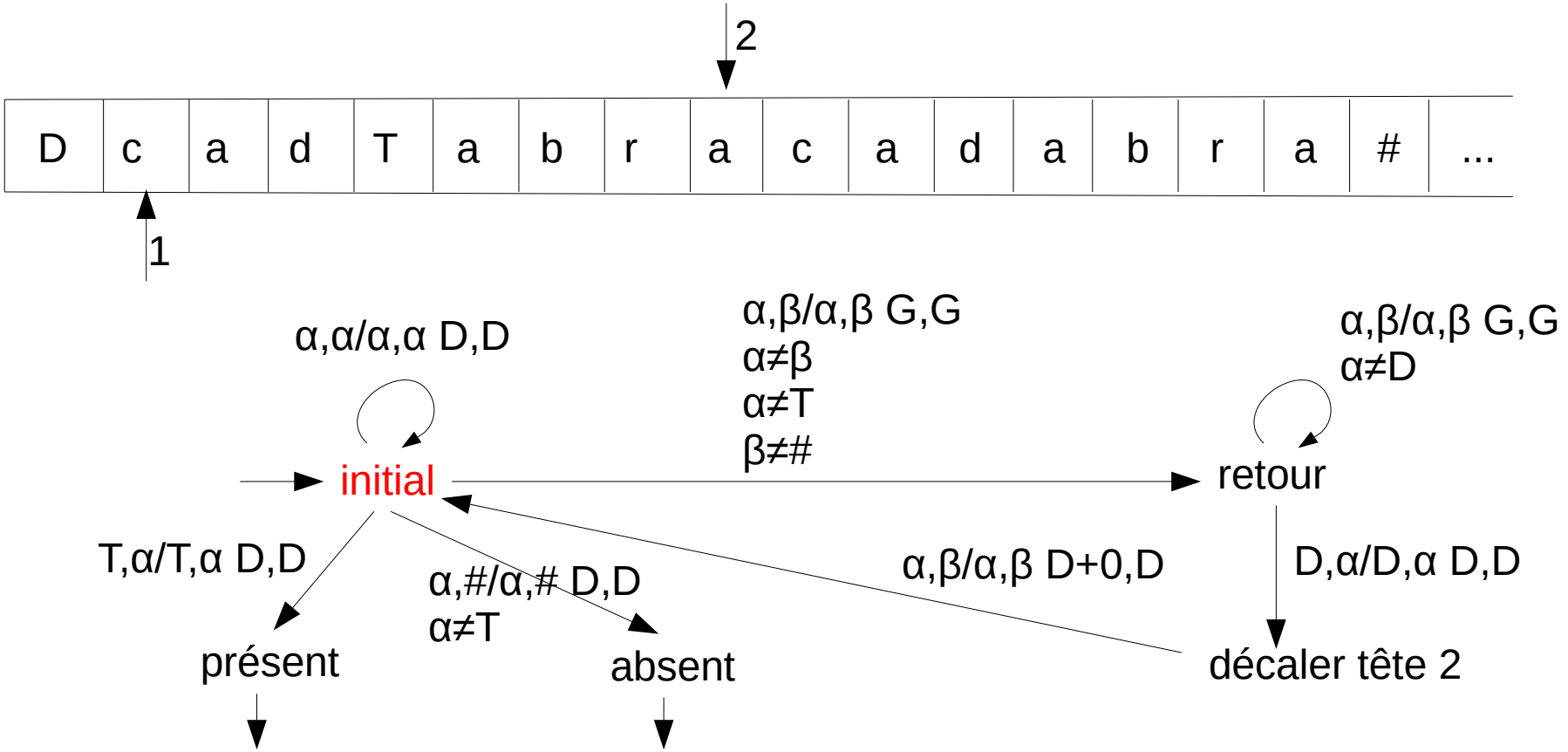
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



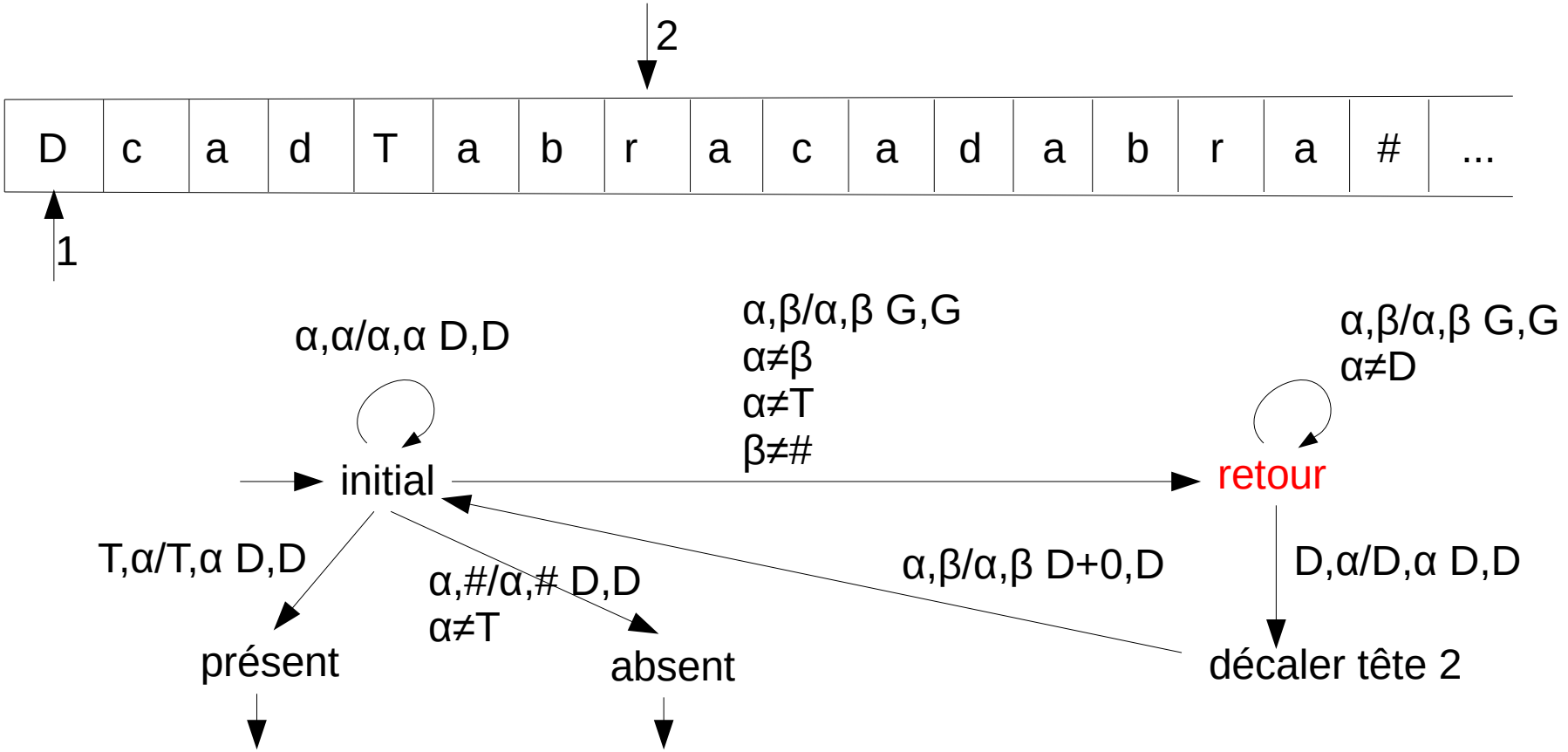
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



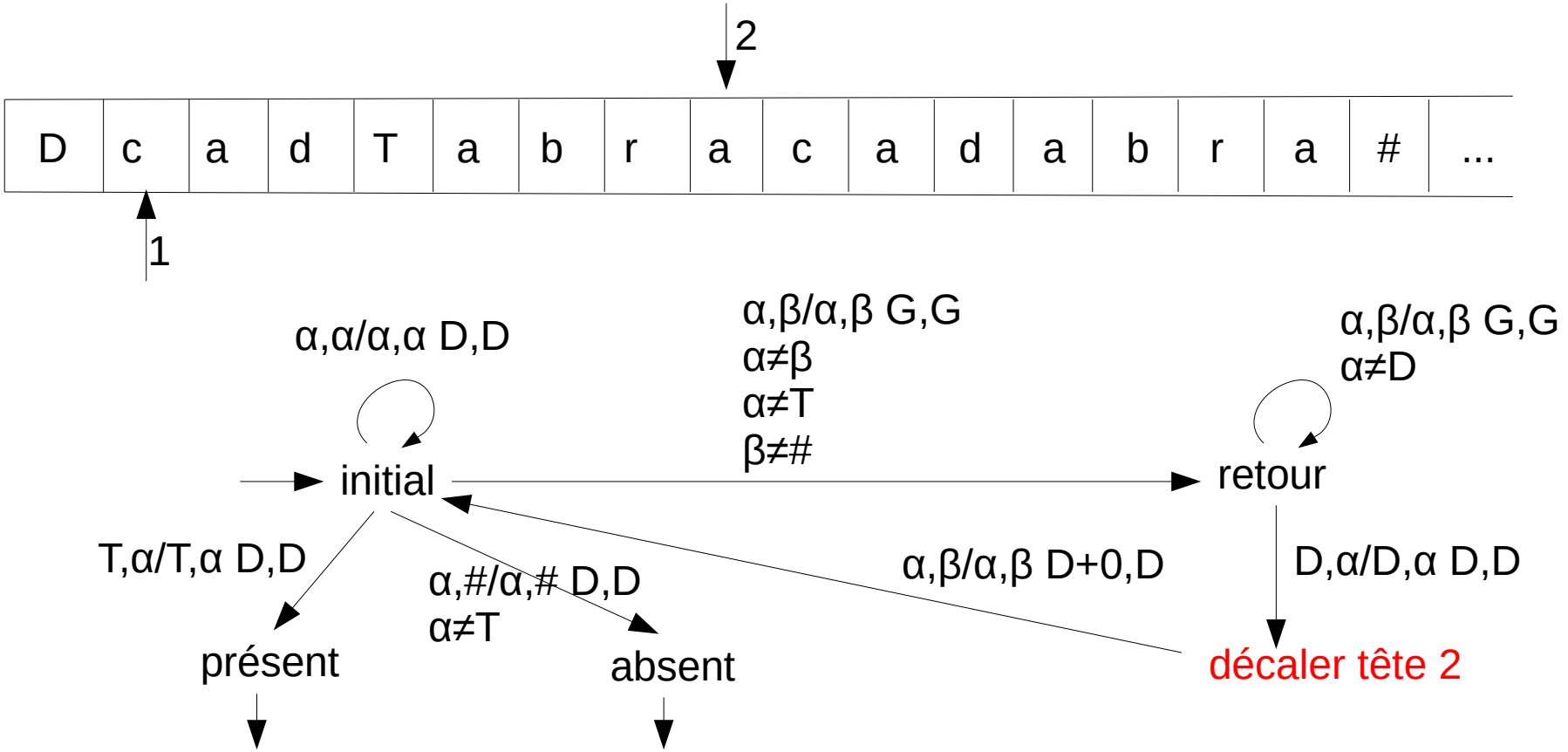
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



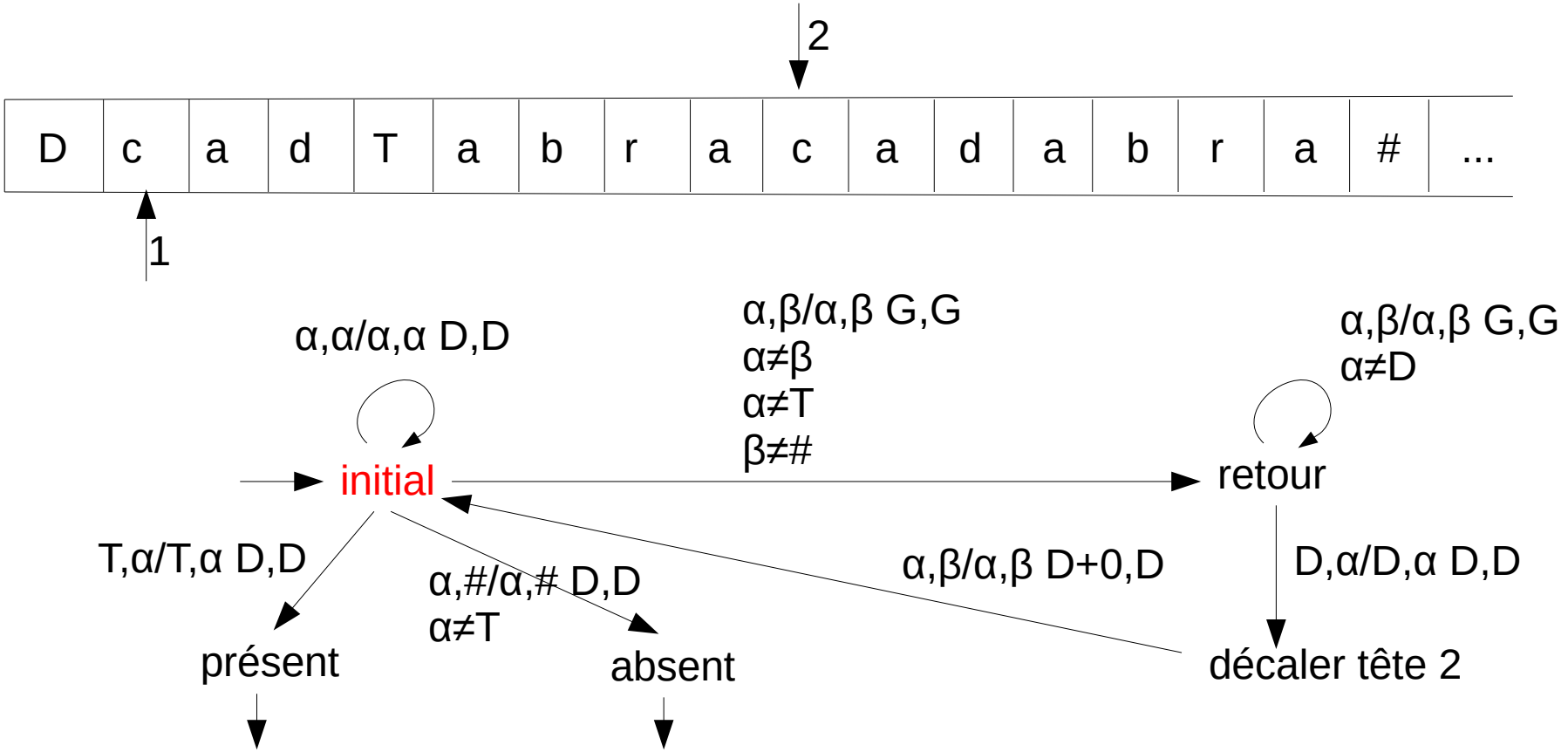
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



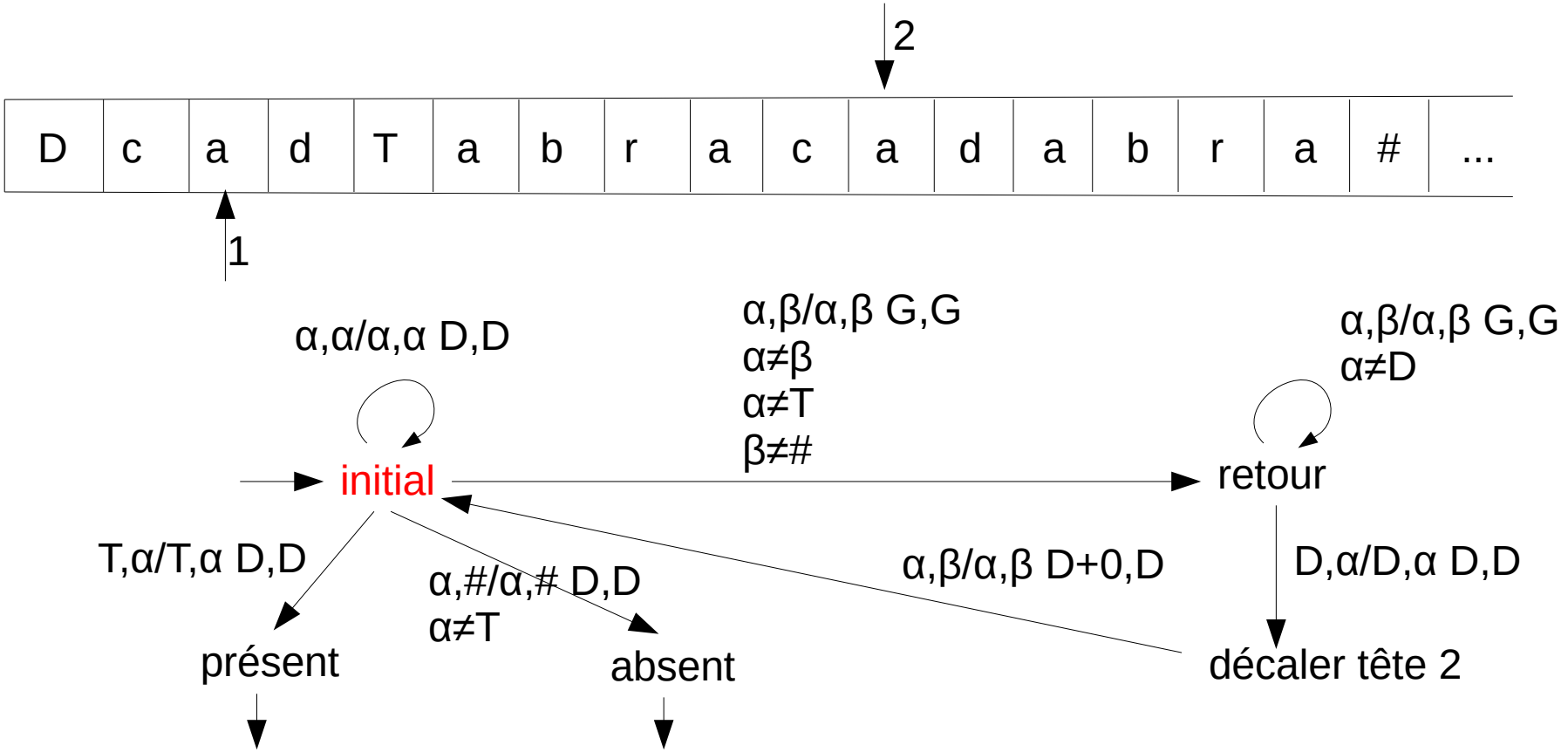
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



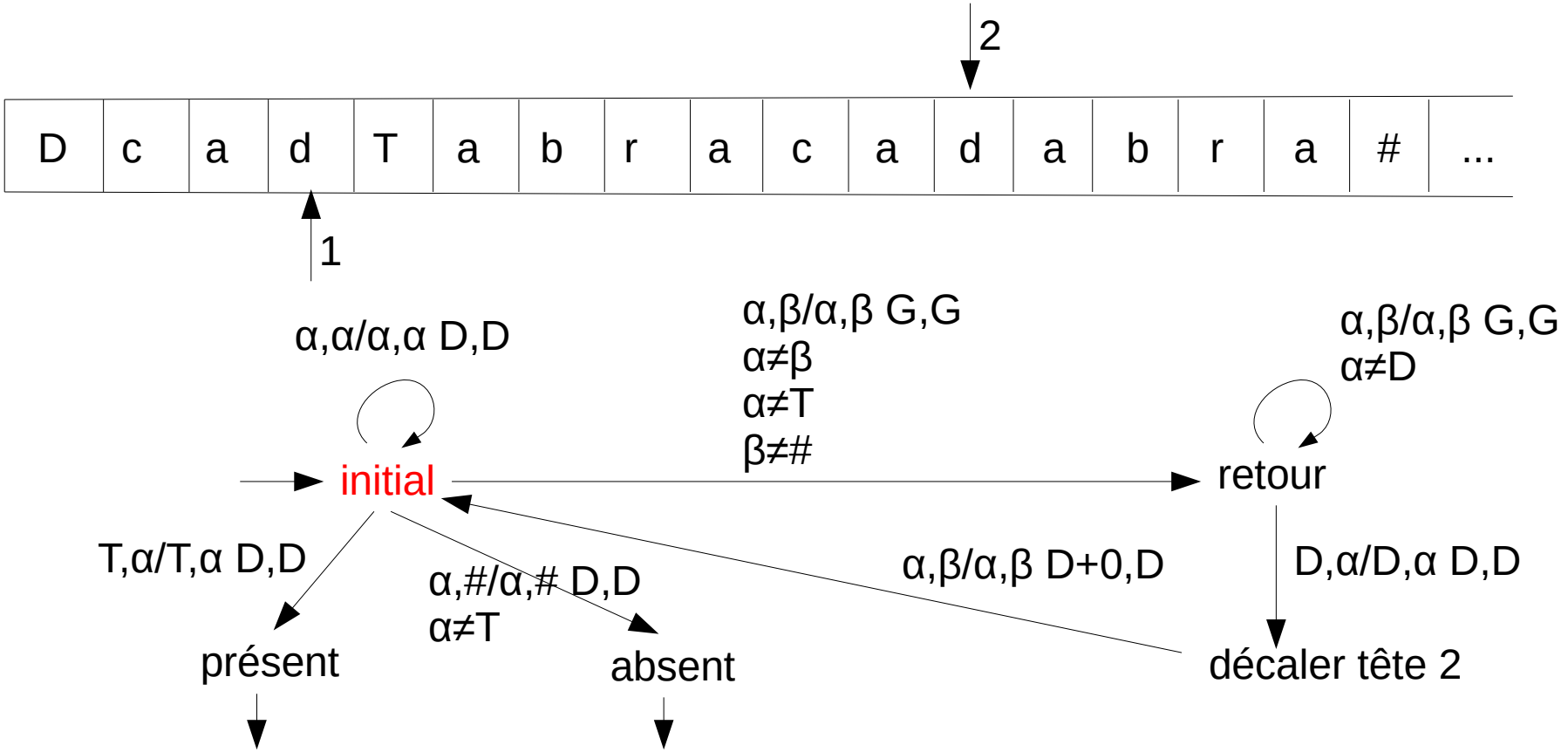
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



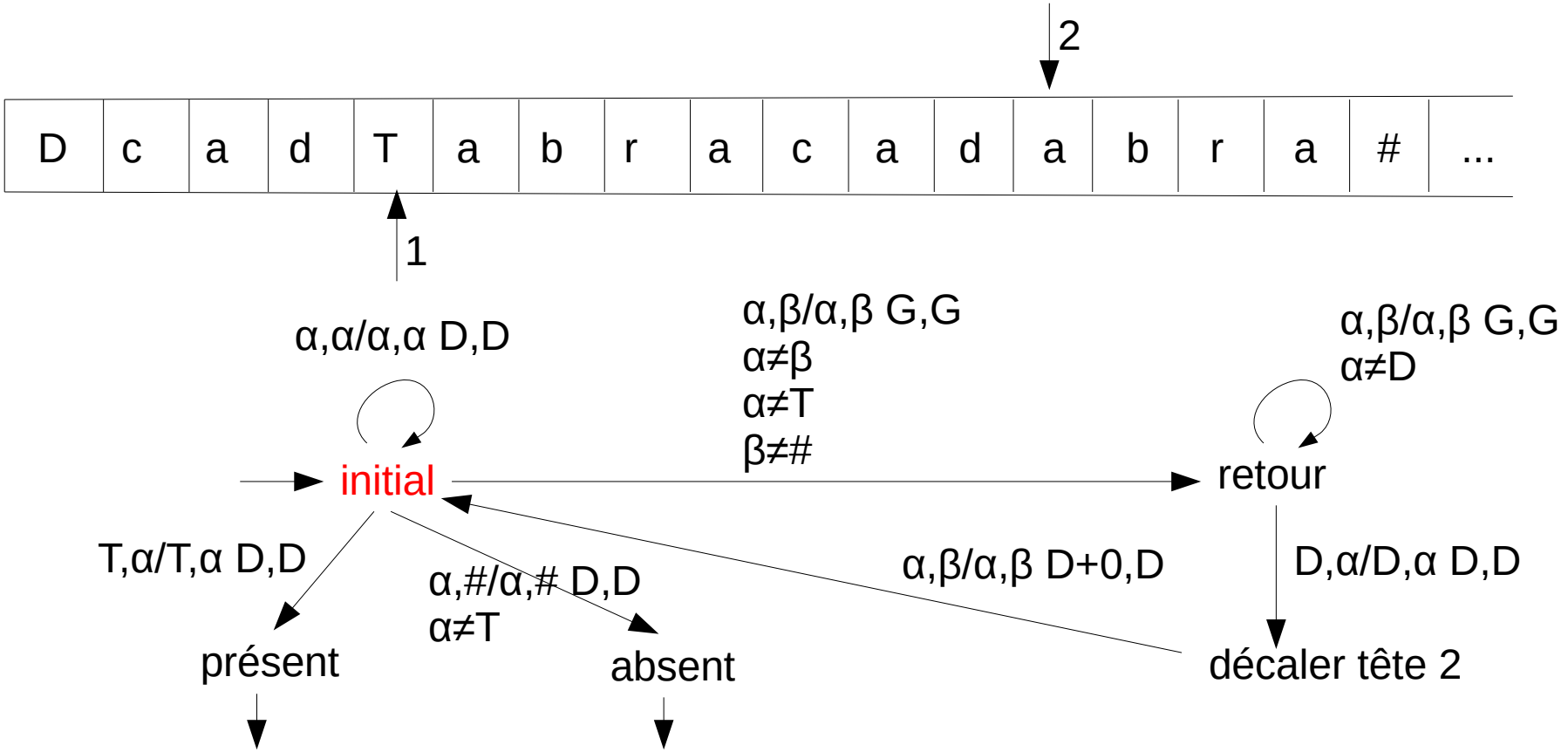
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



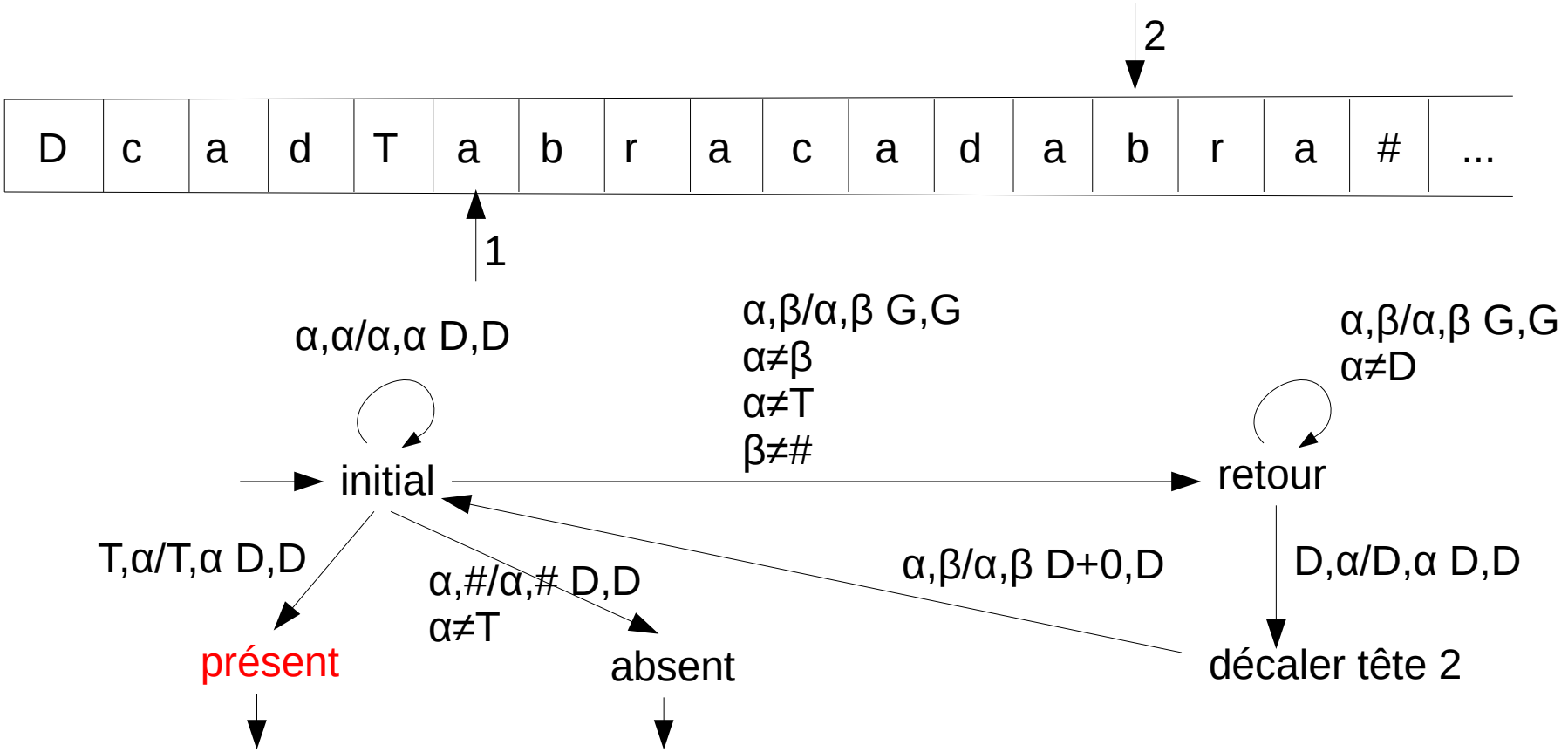
Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



Quelques techniques de construction

MT multi-têtes, mono-bande: exemple



Quelques techniques de construction

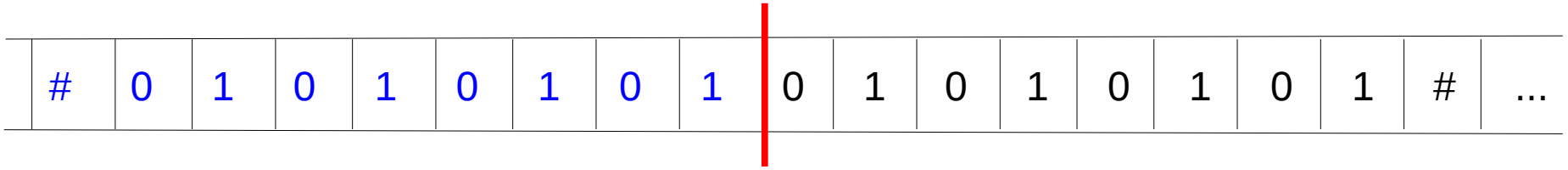
MT avec bande infinie à gauche et à droite

	#	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	#	...

se simule par une MT à deux bandes et une tête en pliant la bande infinie

Quelques techniques de construction

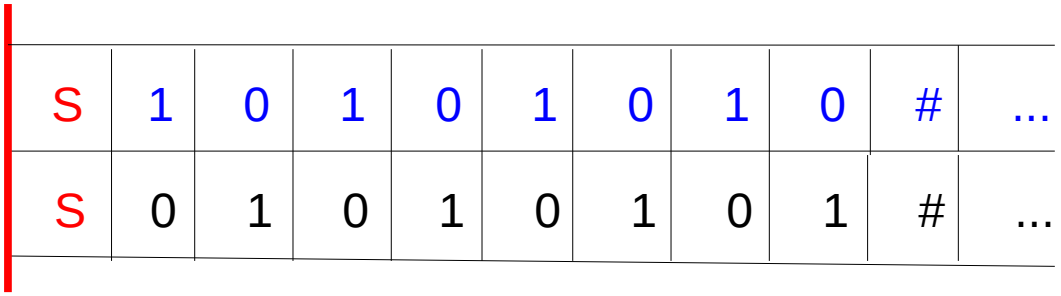
MT avec bande infinie à gauche et à droite



se simule par une MT à une bande et une tête en pliant la bande infinie

États:Q*{+,-}

Alphabet: (T∪S)*(T∪S)



$p,+ \xrightarrow{(\alpha,a)/(\alpha,b)\ X} q,+$ ssi $p \xrightarrow{a/b\ X} q$

$p,- \xrightarrow{(a,\alpha)/(b,\alpha)\ \overline{X}} q,-$

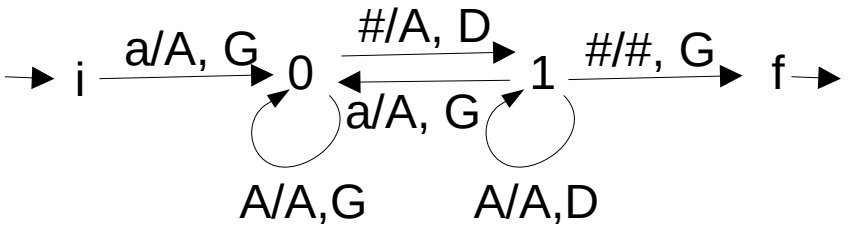
$p,+ \xrightarrow{(S,S)/(S,S)\ D} p,-$

$p,- \xrightarrow{(S,S)/(S,S)\ D} p,+$

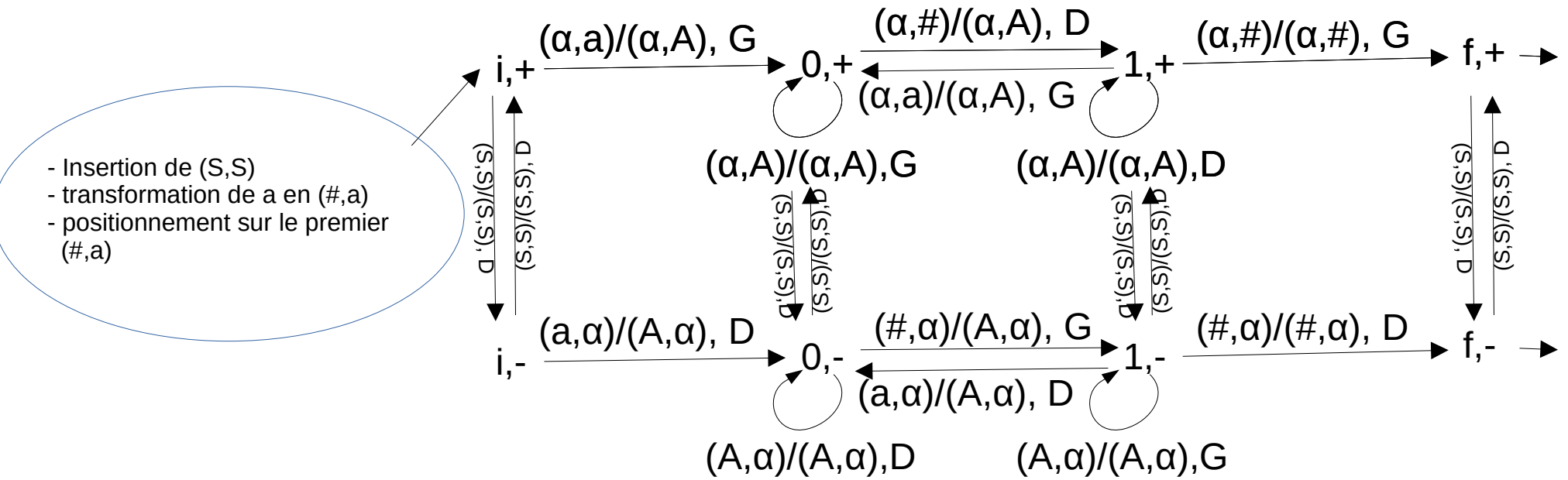
Quelques techniques de construction

MT avec bande infinie à gauche et à droite : exemple

On considère la MT suivante, prenant en entrée a^n et produisant en sortie A^{2n}, bande bi-infinie:



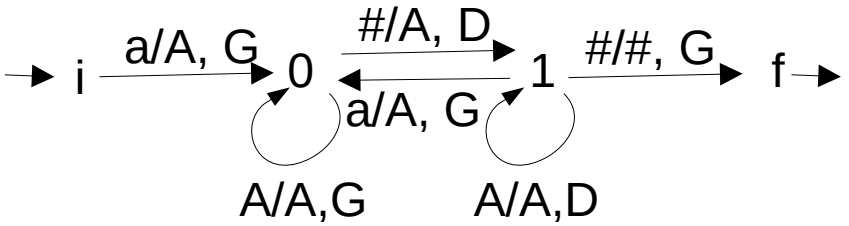
Simulation par MT avec bande simplement infinie à droite:



Quelques techniques de construction

MT avec bande infinie à gauche et à droite : exemple

On considère la MT suivante, prenant en entrée a^n et produisant en sortie A^{2n}, bande bi-infinie:



#iaa#

On a trouvé un *a*, on le remplace par *A* et on va à gauche (forcément à gauche de la position de départ) chercher un #

#0#Aa#

Qu'on remplace par un *A*

#A1Aa#

Puis on va chercher un *a* vers la droite

#AA1a#

Et on recommence

#A0AA#
#0AAA#
#0#AAA#
#A1AAA#
#AA1AA#
#AAA1A#
#AAAA1#

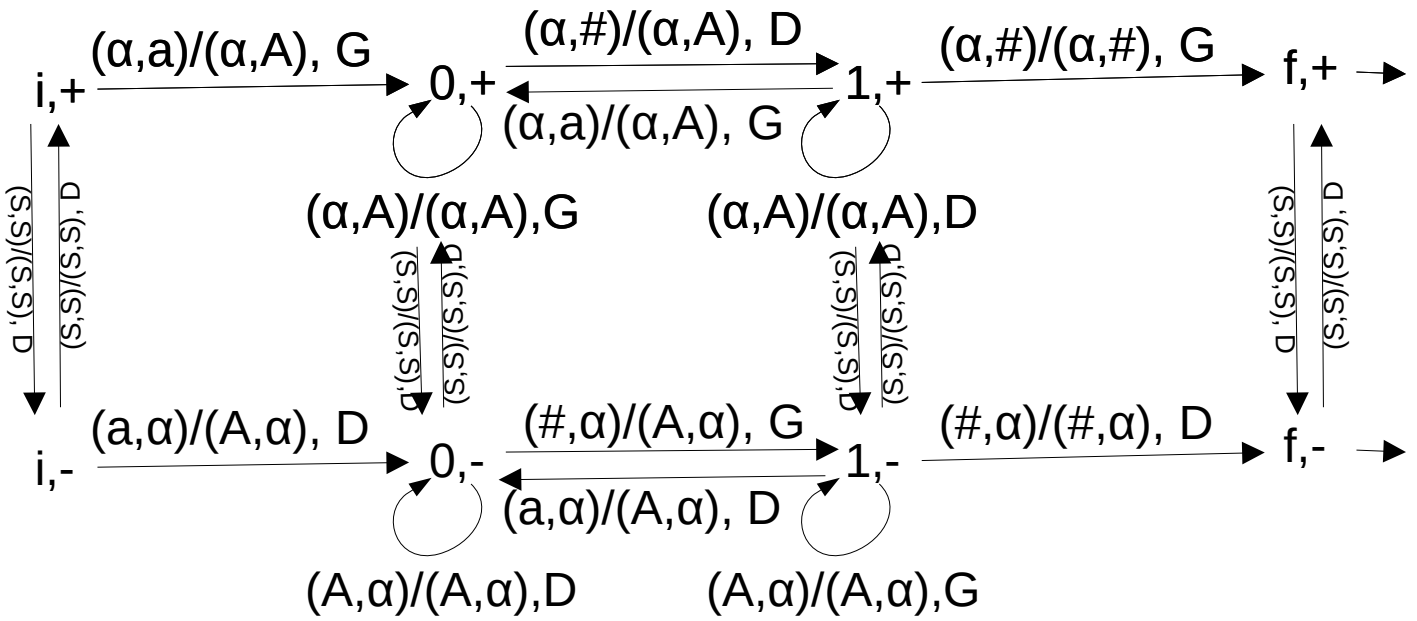
Quand il n'y a plus de *a*, c'est terminé

#AAAfA#

Quelques techniques de construction

MT avec bande infinie à gauche et à droite : exemple

(S,S)(i,+)(#,a)(#,a)(#,#)
(0,+)(S,S)(#,A)(#,a)(#,#)
(S,S)(0,-)(#,A)(#,a)(#,#)
(1,-)(S,S)(A,A)(#,a)(#,#)
(S,S)(1,+)(A,A)(#,a)(#,#)
(S,S)(A,A)(1,+)(#,a)(#,#)
(S,S)(0,+)(A,A)(#,A)(#,#)
(0,+)(S,S)(A,A)(#,A)(#,#)
(S,S)(0,-)(A,A)(#,A)(#,#)
(S,S)(A,A)(0,-)(#,A)(#,#)
(S,S)(1,-)(A,A)(A,A)(#,#)
(1,-)(S,S)(A,A)(A,A)(#,#)
(S,S)(1,+)(A,A)(A,A)(#,#)
(S,S)(A,A)(1,+)(A,A)(#,#)
(S,S)(A,A)(A,A)(1,+)(#,#)
(S,S)(A,A)(f,+)(A,A)(#,#)

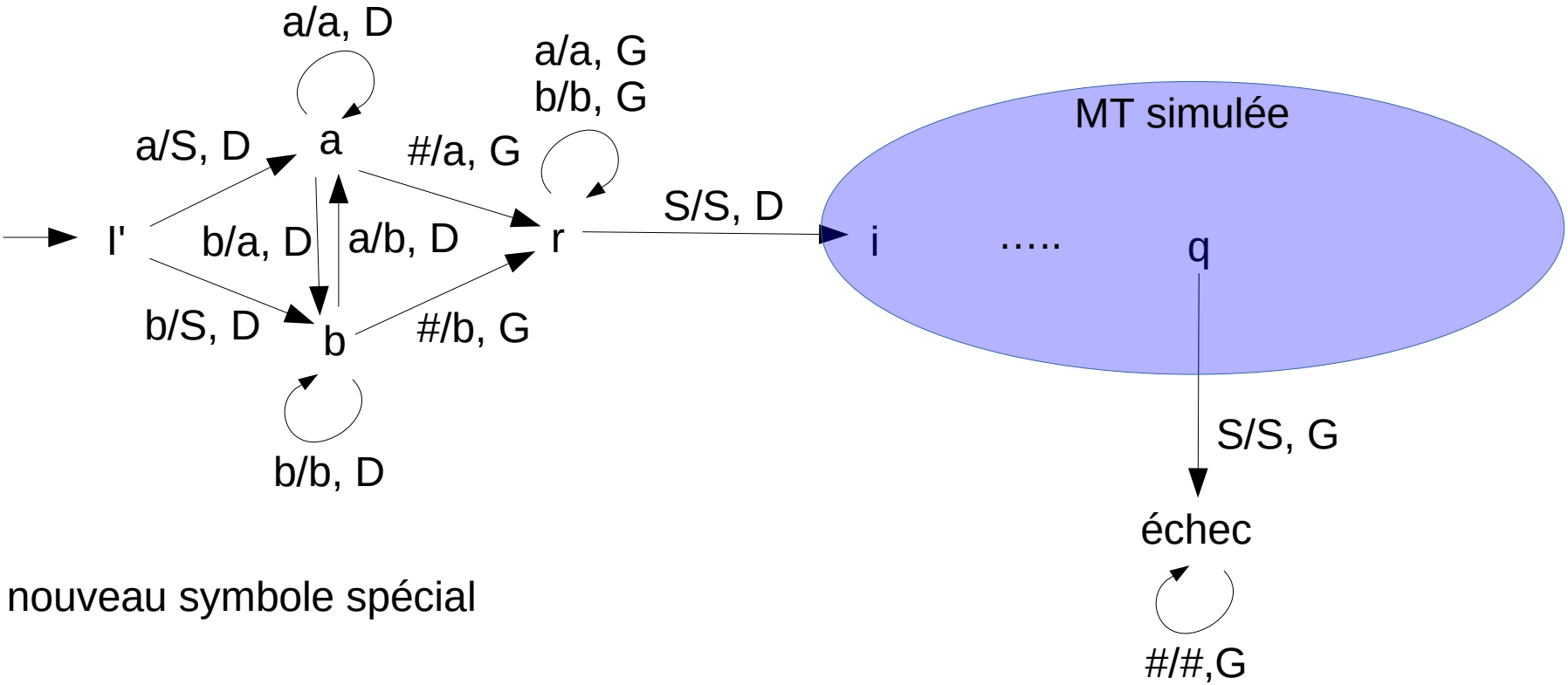


Quelques techniques de construction

MT avec bande infinie à gauche et à droite

...	#	0	1	0	1	0	1	0	1	0	1	0	1	0	1	#	...

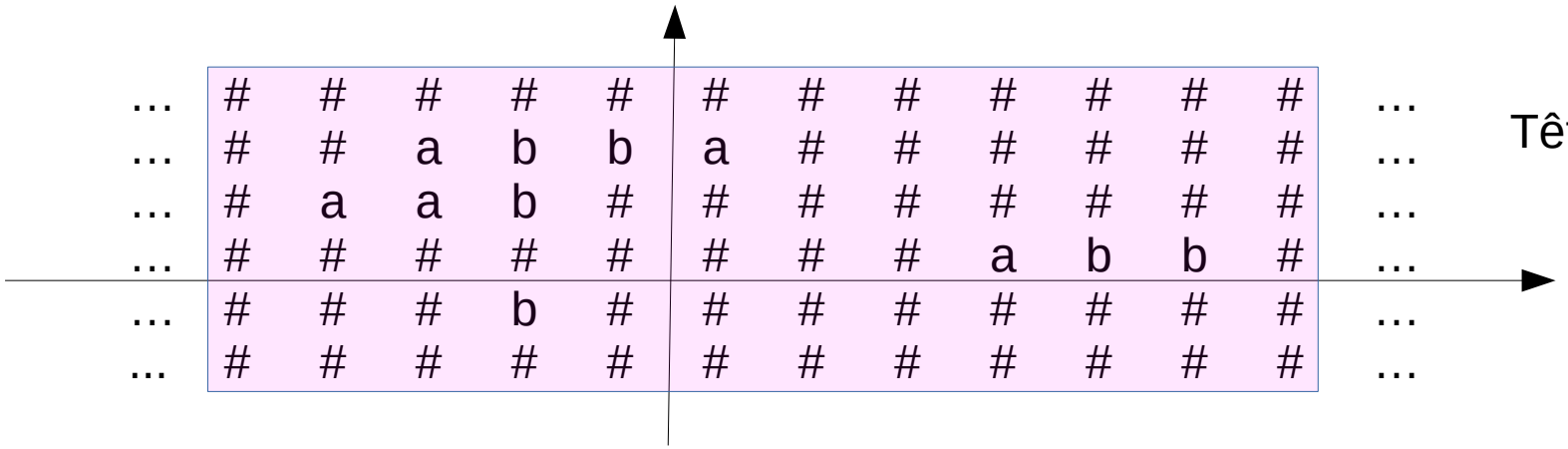
permet de simuler une machine à une bande et une tête avec échec quand le déplacement à gauche est impossible



S: nouveau symbole spécial

Quelques techniques de construction

MT avec bande infinie à *k* dimensions



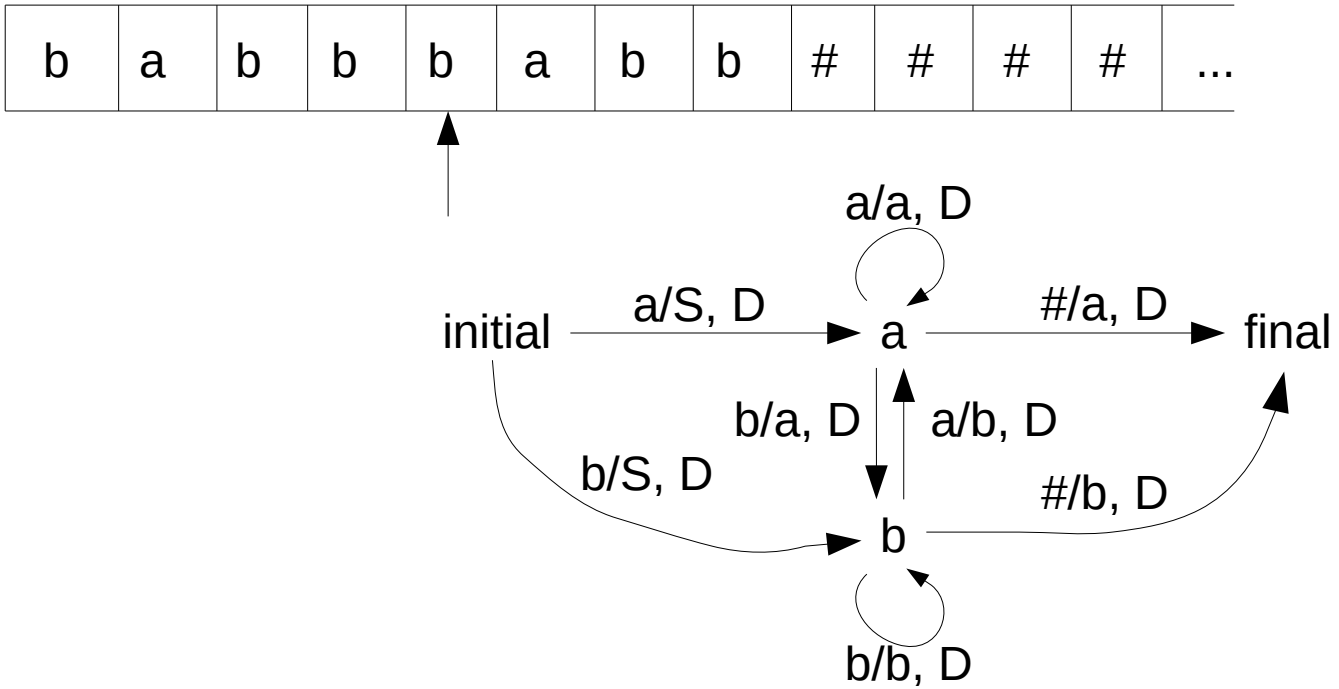
Se simule par une MT à 2 bandes infinies à droite:

XabbaXXXXXaabXXXXXXXXXXXXXXXabbXXbXXXXXXX#…
110,01X1111X1111111111X...

La première contient les données du rectangle
La seconde la position de la tête dans le repère et les dimensions et coordonnées du rectangle
Elle sert aussi pour les calculs: pour simuler un déplacement de la tête de (dx,dy)
dans le plan, il faut calculer de combien, à gauche ou à droite, se déplacer sur la bande 1
Il peut être nécessaire d'élargir le rectangle, donc de faire de la place sur la bande 1
pour y mettre de nouvelles données

Quelques techniques de construction

Faire de la place sur la bande



Insère un blanc (S) à l'emplacement de la tête.
Se généralise facilement à l'insertion de k blancs

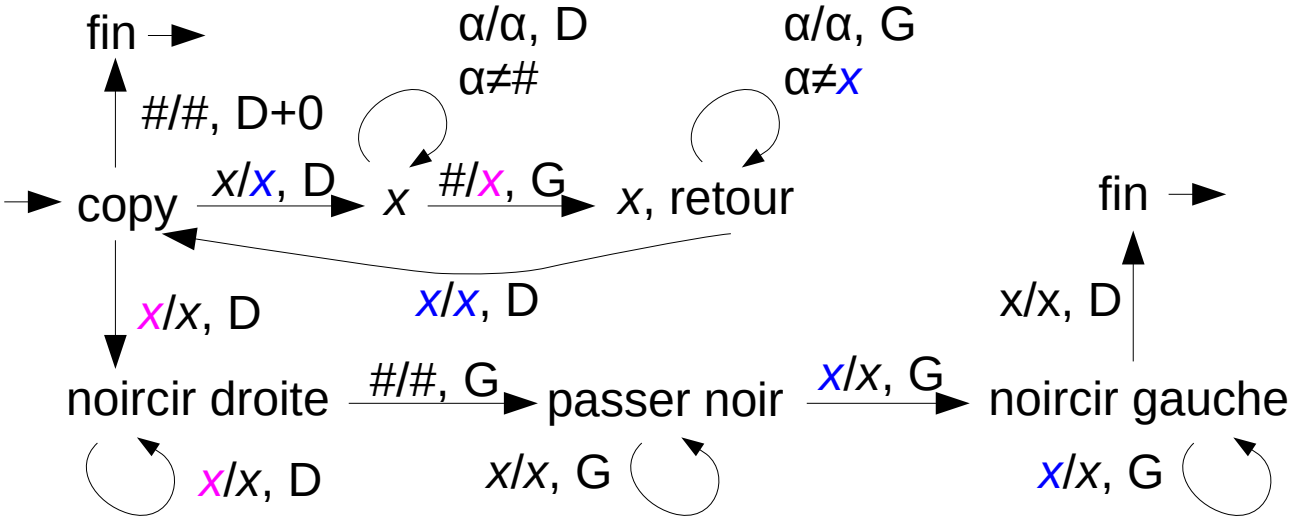
Quelques techniques de construction

Réutiliser les machines de Turing

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#
suppose u non vide

$x \in A$

$x, x \notin A$ nouveaux symboles associés à x



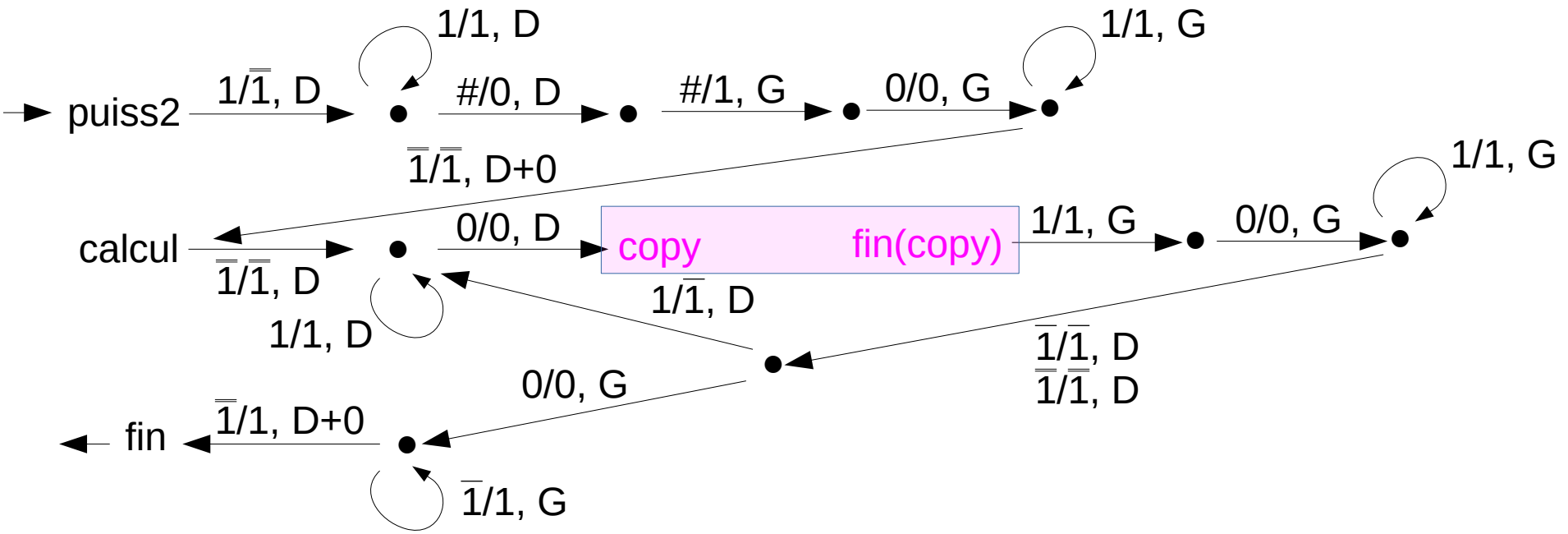
Quelques techniques de construction

Réutiliser les machines de Turing

puiss2: calcule 2^n, n>0

Entrée: upuiss2$1^n$# avec n>0

Sortie: ufin$1^n01^{2^{**}n}$#



Fonctions récursives primitives

- Définies ici sur les mots, mais on pourrait le faire de manière équivalente sur les entiers (par simple codage)
- L'alphabet $X = \{ x_1,\ldots,x_n \}$ a au moins deux lettres et est ordonné
  - si $x_i \in X$, alors $prochaineLettre(x_i) = x_{(i-1)[n]+1}$
- Fonctions de base : opérateurs élémentaires de manipulation de mots
  - $ajout_x(w) = wx$
  - premièreLettre(w) :
    - ε si w = ε
    - x s'il existe $x \in X$, $w' \in X^*$ tels que $w = xw'$
  - dernièreLettre(w) :
    - ε si w = ε
    - x s'il existe $x \in X$, $w' \in X^*$ tels que $w = w'x$
  - saufPremièreLettre(w) :
    - ε si w = ε
    - w' s'il existe $x \in X$, $w' \in X^*$ tels que $w = xw'$
  - saufDernièreLettre(w) :
    - ε si w = ε
    - w' s'il existe $x \in X$, $w' \in X^*$ tels que $w = w'x$
  - $égal_{x'}(x,y)$:
    - ε si x = y
    - x' sinon, où x' est une lettre fixée

Fonctions récursives primitives
Schémas de construction

- Schéma de composition générale
  - $h(w) = g(f_1(w),\ldots,f_k(w))$
- Schéma de récurrence simple
  - $h(\varepsilon,m) = f(m)$
  - $h(wx,m) = g(w,x,h(w,m),m)$
- Schéma de récurrence simultanée
  - permet de définir k récursions en même temps
  - $h_i(\varepsilon,m) = f_i(m)$
  - $h_i(wx,m) = g_i(w,x,h_1(w,m),\ldots,h_k(w,m),m)$
- Définition au cas par cas
  - $f_i:X^* \rightarrow X^*$, $i \in \{ 1,\ldots,n \}$, un ensemble de fonctions
  - $P_1,\ldots,P_n$ une partition de X^* en parts P_i
  - $t_i : X^* \rightarrow \{ 0,1 \}$, $i \in \{ 1,\ldots,n \}$, un ensemble de fonctions vérifiant $t_i(w) = 1$ ssi $w \in P_i$
- Fonction récursive primitive : construite à partir
  - des mots (constantes)
  - des fonctions de base
  - de la copie, de l'identité, de la projection et de la construction de k-uplets
  - des schémas de composition générale, de récurrence simple et de récurrence simultanée
    - en fait, le schéma de récurrence simultanée n'est pas nécessaire : il peut être construit à partir du reste
  - définition au cas par cas

Fonctions récursives primitives Exemples

- concat(w',w)=ww' est récursive primitive, car elle peut être
  - définie au cas par cas, en fonction de la dernière lettre de w'
  - avec pour chaque cas un schéma de récurrence simple
    - concat(ε,w)=id(w)
    - concat(w'x,w)=ajout$_x \circ \pi_3$(w',x,concat(w',w),w)=ajout$_x$(concat(w',w))
- miroir($x_1...x_{n,}\varepsilon$)=$x_n...x_1$ est récursive primitive, car elle peut être
  - définie au cas par cas, en fonction de la dernière lettre de w'
  - avec pour chaque cas un schéma de récurrence simple
    - miroir(ε,ε)=id(ε)
    - miroir($x_1...x_{n,}\varepsilon$)=g($x_1...x_{n-1}$,x_n,miroir($x_1...x_{n-1}$,ε),ε)
  - avec g définie par application du schéma de composition générale
    - g(c)=concat(π_3(c),π_2(c))
- condition(w,f) définie par condition(ε,f)=ε et condition(w,f)=f si w≠ε est récursive primitive
- succ(w), qui calcule le mot successeur de w dans l'ordre militaire, est récursive primitive
  - on définit les fonctions récursives primitives suivantes
    - $cas_1(\varepsilon)=\varepsilon$ et $cas_1(wx)$=condition(égal(x,x_k),ajout$_{x'}$(w)) avec x' lettre suivant x dans l'ordre de X
      - si la dernière lettre de wx n'est pas la dernière de l'alphabet, on la remplace par la lettre suivante
    - $cas_2(\varepsilon,u)=\varepsilon$ et $cas_2(wx,u)$=condition(différent(x,x_k),ajout$_{x1}$(u))
      - si la dernière lettre de wx est la dernière lettre de l'alphabet, on construit ux_1
    - $succ(\varepsilon)=x_1$ et succ(wx)=concat(cas_1(wx),cas_2(wx,succ(w)))
      - la fonction succ agit comme l'ajout de 1 sur un entier, en propageant une retenue et en parcourant le mot de la droite vers la gauche

Fonctions récursives

- Construites comme les fonctions récursives primitives, avec en plus l'utilisation du schéma de minimisation:
  - h(g,w) est le plus petit mot w' (pour l'ordre militaire) tel que
    - g(w',w) = ε
    - g(w'',w) est défini pour tout w''$<_{mil}$w'
  - on note que h(g,w) peut ne pas être définie, et donc qu'on peut obtenir une fonction récursive partielle en utilisant le schéma de minimisation sur des fonctions récursives primitives totales
- Il existe des fonctions récursives totales qui ne sont pas récursives primitives
  - les fonctions récursives primitives sont énumérables
    - il suffit de les énumérer en numérotant les différents schémas utilisés
  - on les énumère, une par ligne : f_0,f_1,...
  - on énumère, un par colonne, les mots par ordre militaire : w_0,w_1,…
  - la fonction définie par f(w_i)=successeur(f_i(w_i)) n'est pas dans f_0,f_1,… (par diagonalisation)
  - elle est récursive (totale), car définie à partir de fonctions récursives primitives par composition, et en utilisant (une seule fois) le schéma de minimisation pour trouver f_i à partir de w_i

Une autre fonction récursive non primitive: la fonction d'Ackermann

- Définie par
  - $A(m,n) = n+1$ si $m=0$
  - $A(m,n) = A(m-1,1)$ si $m>0$ et $n=0$
  - $A(m,n) = A(m-1,A(m,n-1))$ dans les autres cas
- ou par
  - $A(m,n)=2\uparrow^{m-2}(n+3)-3$
  - avec
    - $a\uparrow^n b = 1$ si $b=0$
    - $a\uparrow^n b = a^b$ si $n=1$
    - $a\uparrow^n b = a\uparrow^{n-1}(a\uparrow^n(b-1))$ sinon
- La définition récursive ci-dessus n'est pas en forme récursive primitive car les arguments ne décroissent pas dans la récursion
- La fonction d'Ackermann croit strictement plus vite que toute fonction récursive primitive
- Donc, il n'existe pas de définition de cette fonction sous forme récursive primitive
- Très facile à programmer !

Des MT aux fonctions récursives

- Les changements de configuration s'expriment avec des fonctions récursives. On pose
  - subst(q,x) = y
  - déplacement(q,x) = X
  - état(q,x) = p

  si δ(q,x) = (p,y,X) est une transition de la MT
- On définit par récursivité mutuelle
  - S(w,k): état atteint après k étapes quand la MT est démarrée avec une bande initialisée à w
  - G(w,k): mot à gauche de la tête de lecture après k étapes quand la MT est démarrée avec une bande initialisée à w
  - D(w,k): w'x si, après k étapes quand la MT est démarrée avec une bande initialisée à w,
    - le symbole sous la tête est x
    - w' est le miroir du mot à droite de la tête
  - L(w,k): lettre sous la tête après k étapes quand la MT est démarrée avec une bande initialisée à w
- Pour des raisons de simplicité, on assimile un mot w sur un alphabet à n lettres avec un entier écrit en base n
  - on a alors, par exemple, L(w,k) = D(w,k) [n]

Des MT aux fonctions récursives

- On définit par récursivité mutuelle
  - S(w,k): état atteint après k étapes quand la MT est démarrée avec une bande initialisée à w
    - S(w,0) = état initial de la machine
    - S(w,k+1) = état(S(w,k),L(w,k))
  - G(w,k): mot à gauche de la tête de lecture après k étapes quand la MT est démarrée avec une bande initialisée à w
    - G(w,0) = 0, car il n'y a rien à gauche de la tête de lecture au départ
    - G(w,k+1) = G(w,k)/n si déplacement(S(w,k),L(w,k))=G
    - G(w,k+1) = G(w,k)*n+subst(S(w,k),L(w,k)) si déplacement(S(w,k),L(w,k))=D
  - D(w,k): w'x si, après k étapes quand la MT est démarrée avec une bande initialisée à w,
    - D(w,0) = miroir(w)
    - D(w,k+1) = D(w,k)*n+subst(S(w,k),L(w,k)) si déplacement(S(w,k),L(w,k))=G
    - D(w,k+1) = D(w,k)/n si déplacement(S(w,k),L(w,k))=D
  - Ces trois fonctions sont récursives primitives.
  - Elles expriment l'exécution, mais pas l'existence d'une exécution acceptante

Des MT aux fonctions récursives

- Il reste à exprimer l'existence de l'acceptation
  - on peut toujours supposer que la machine n'a qu'un seul état final f
  - on peut également toujours supposer qu'avant d'accepter, la machine remet sa tête complètement à gauche de la bande
  - après acceptation, sur un parcours de k étapes, l'état de la bande est alors
    - miroir(D(w,k)), où k est le plus petit entier tel que S(w,k)=f
      [on utilise ici pour la première fois le schéma de minimisation]
- Donc, la fonction réalisée par la MT est récursive
- Passer des fonctions récursives aux MT est très facile
  - il suffit de simuler le calcul par un programme en assembleur d'une machine RAM
- Remarque: le schéma de minimisation n'a été employé, dans le sens MT→fonction récursive
  - qu'une seule fois
  - à la fin

  donc toute fonction récursive est équivalente à une fonction récursive dans laquelle le schéma de minimisation n'est employé qu'une fois au plus, et à la fin du calcul.
- Une fonction est récursive si et seulement si elle est définissable par une MT

Machines de Turing comme générateurs

Générateur: machine de Turing sans entrée qui écrit sur sa bande les mots d'un langage L.
Quand la machine a fini d'écrire un mot de L, elle le fait suivre d'un symbole spécial $, puis passe à la génération du mot suivant.

Si un langage est généré par une MT M, alors il est récursivement énumérable:
Le travail de la machine MT M' consiste à simuler M, et, à chaque fois que M produit w_i\$, à comparer w_i avec sa propre entrée.
Remarque: M' peut ne pas s'arrêter si son entrée n'est pas dans L.

Si un langage L est récursivement énumérable par une MT M, alors il est généré par une MT M': plus difficile. Problème: M peut ne pas s'arrêter sur une entrée w si $w \notin L$.

Machines de Turing comme générateurs

Si un langage L est récursivement énumérable par une MT M, alors il est généré par une MT M': plus difficile.

Idée: Une machine M'' génère tous les mots sur l'alphabet.

M analyse si chaque mot généré w est dans L.

Si oui il copie w$ sur une bande de sortie.

Si non: problème, car M peut ne pas s'arrêter !

Solution: générer tous les couples (w,n)

w mot,

n nombre d'étapes autorisées pour l'analyse de w par M

Si $w \in L$, alors il existe n tel que M accepte w en n étapes

Génération de tous les couples (w,n):

similaire à la génération de tous les couples (i,j), $i,j \in N$

ordre croissant de i+j

(0,0) (1,0) (0,1) (2,0) (1,1) (0,2) (3,0) (2,1) (1,2) (0,3) (4,0) (3,1) ...

mot w sur un alphabet à n lettres = codage d'un entier en base n

Ordre canonique: a, b, aa, ab, ba, bb, aaa, aab, aba, abb, ...

Machines de Turing comme générateurs

Soit L un langage infini.
L est récursif ssi il est généré dans l'ordre canonique

Si L est récursif, il existe une MT L qui dit si oui ou non un mot en entrée est dans L.
Pour construire une MT M qui génère L par ordre canonique, il faut une MT G qui génère tous les mots par ordre canonique. Chaque mot généré passe en entrée de L
Si L indique que $w \in L$, alors M recopie w$ sur sa sortie, et demande à G de générer le mot suivant.
Si L indique que $w \notin L$, alors M demande à G de générer le mot suivant.

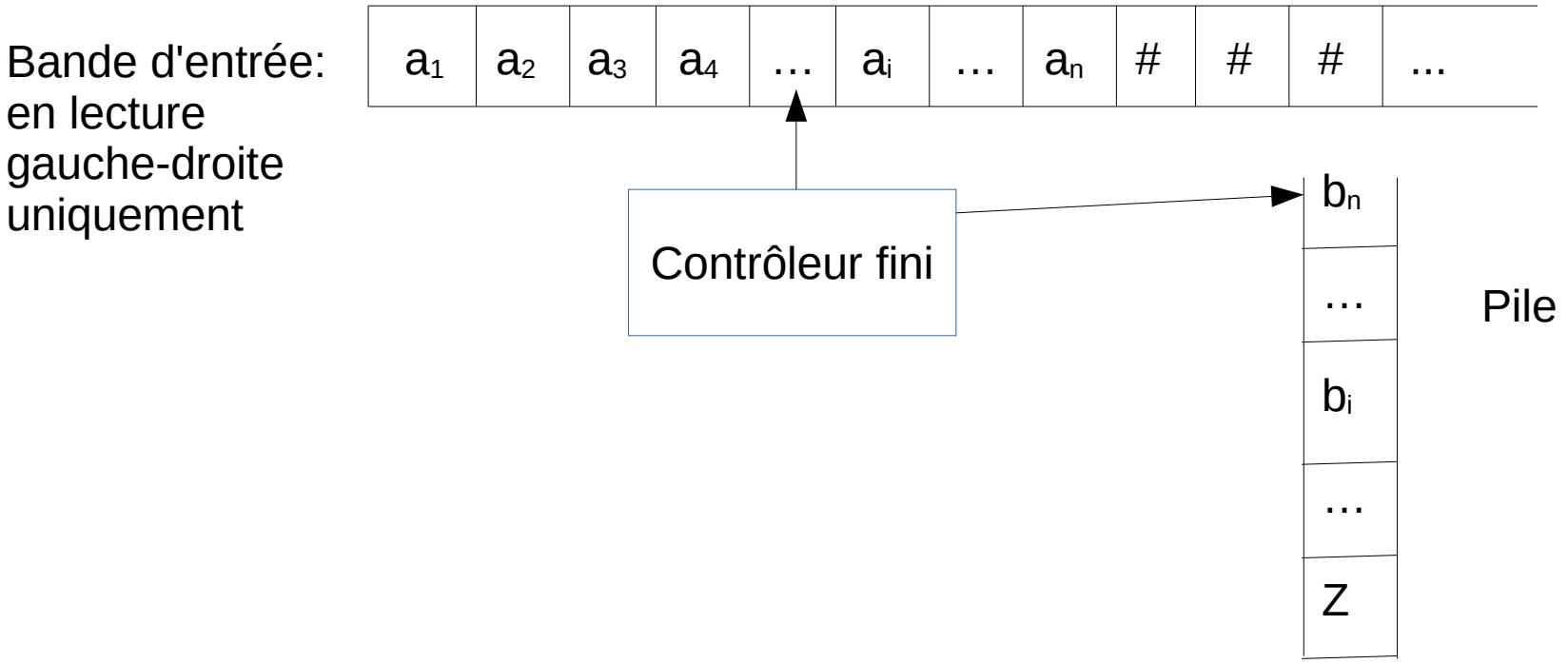
Si L est généré en ordre canonique par une MT G, alors la MT M qui décide si son entrée $w \in L$ ou non consiste à demander à G de générer tous les mots par ordre canonique jusqu'à w si $w \in L$, ou $w' > w$, $w' \in L$ sinon.

Remarque: si L est fini et $w \notin L$, alors G pourrait ne jamais sortir $w' > w$, $w' \in L$.
Pour construire M, il faut donc savoir si L est fini, et si oui, connaître son nombre de mots ou son dernier mot pour l'ordre canonique.

Automates à pile (*PushDown Automata : PDA*)

- Deux façons de les voir

1) MT restreinte avec
   - une bande d'entrée
   - une bande gérée comme une pile
2) Automate fini + pile

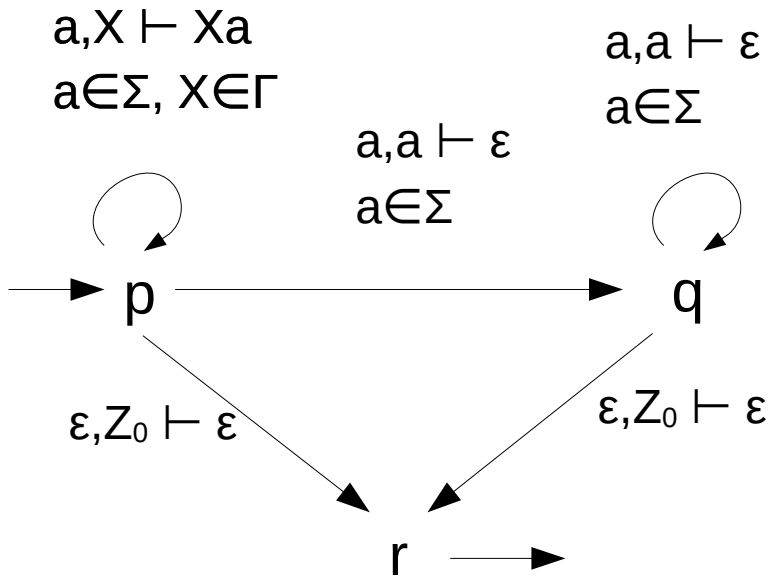


Automates à pile (*PushDown Automata: PDA*)

- Formellement $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$
  - Q ensemble fini d'états
  - Σ alphabet d'entrée
  - Γ alphabet de pile
  - $q_0 \in Q$ l'état initial
  - $Z_0 \in \Gamma$ symbole de fond de pile
  - F états finaux
    - $\delta : Q^*(\Sigma \cup \{\varepsilon\})^*\Gamma^*Q^*\Gamma^*$ transitions
      - $(q,a,A,q',s) \in \delta$:

        Dans l'état q, à la lecture de a sur la bande d'entrée et avec le symbole A en somme et de pile, passer dans l'état q' et remplacer A en sommet de pile par s
      - Remarque: ε-transitions autorisées

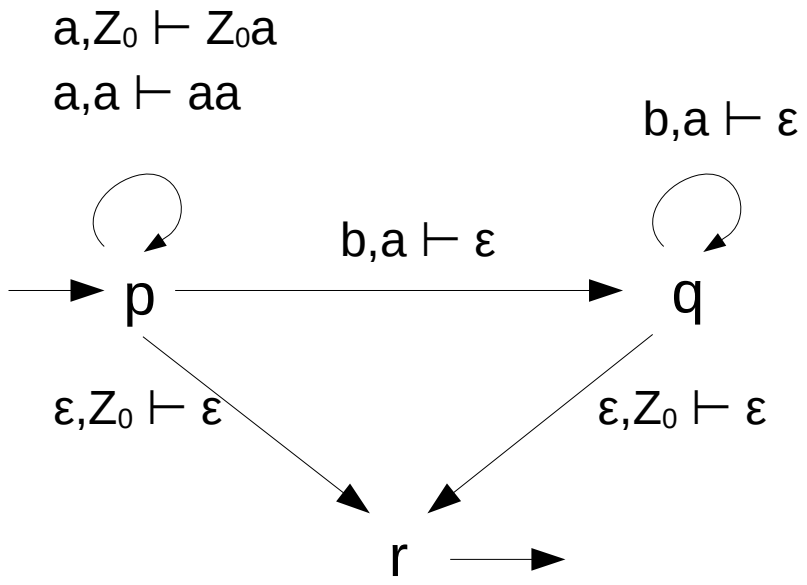
Automates à pile: exemple 1



$L=\{w\overline{w} : w\in\Sigma^*\}$

- p empile les lettres lues (dans l'ordre de lecture)
- la transition de p vers q suppose qu'on est au milieu du mot (non déterminisme)
- q vérifie que les lettres qu'il reste à lire correspondent au contenu de la pile
- pour aller dans l'état final r, il faut que la pile soit vide, et pour accepter, qu'il n'y ait plus rien à lire.

Automates à pile: exemple 2



$L=\{ a^n b^n : n \in N \}$

L'ensemble des langages définis par automates (de Kleene) est strictement inclus dans l'ensemble des langages définis par automates à pile

Automates à pile: une autre définition

- Condition d'acceptation 1: l'entrée w est acceptée si et seulement il existe une lecture de w qui amène dans un état final

  $L=\{ w : (q_0,w,Z_0) \vDash (f,\varepsilon,P), P\in\Gamma^*, f\in F \}$

- Condition d'acceptation 2: l'entrée w est acceptée si et seulement si il existe une lecture de w qui amène sur une pile vide

  $L=\{ w : (q_0,w,Z_0) \vDash (p,\varepsilon,\varepsilon), p\in Q \}$

- Les deux conditions sont équivalentes

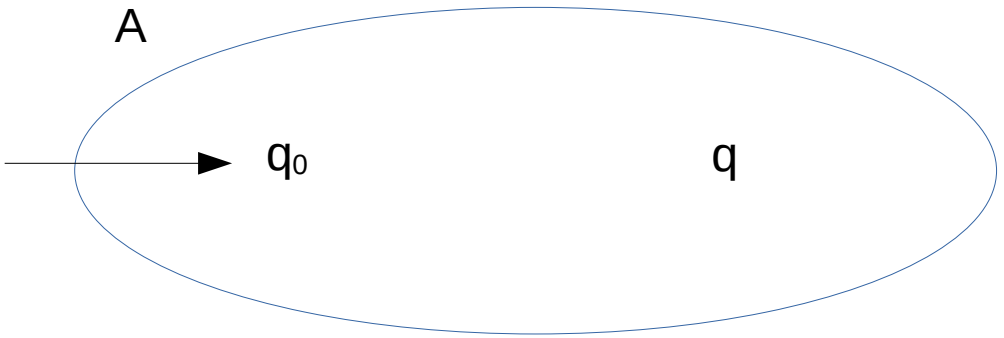
Automates à pile:

de l'acceptation par pile vide à l'acceptation par état final

Soit $A=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,\emptyset)$ un PDA avec acceptation par pile vide.

On construit $B=(Q\cup\{q'_0,f\},\Sigma,\Gamma\cup\{Z'_0\},\delta,q'_0,Z'_0,\{f\})$

Il faut détecter le dépilement du symbole Z_0 de fond de pile, et rediriger vers un nouvel état final f



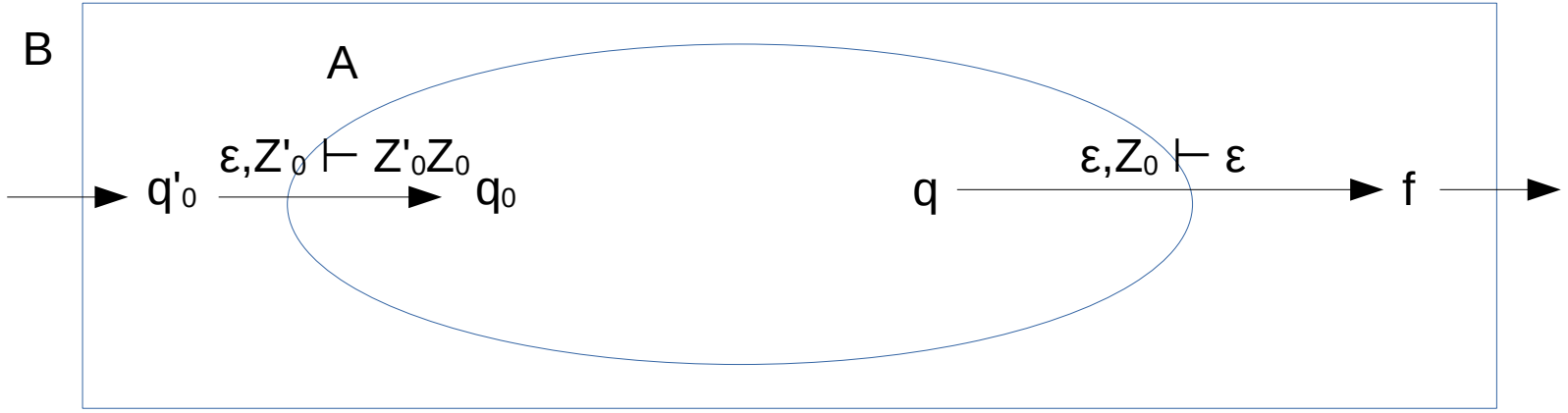
Automates à pile:

de l'acceptation par pile vide à l'acceptation par état final

Soit $A=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,\emptyset)$ un PDA avec acceptation par pile vide.

On construit $B=(Q\cup\{q'_0,f\},\Sigma,\Gamma\cup\{Z'_0\},\delta,q'_0,Z'_0,\{f\})$

Il faut détecter le dépilement du symbole Z_0 de fond de pile, et rediriger vers un nouvel état final f

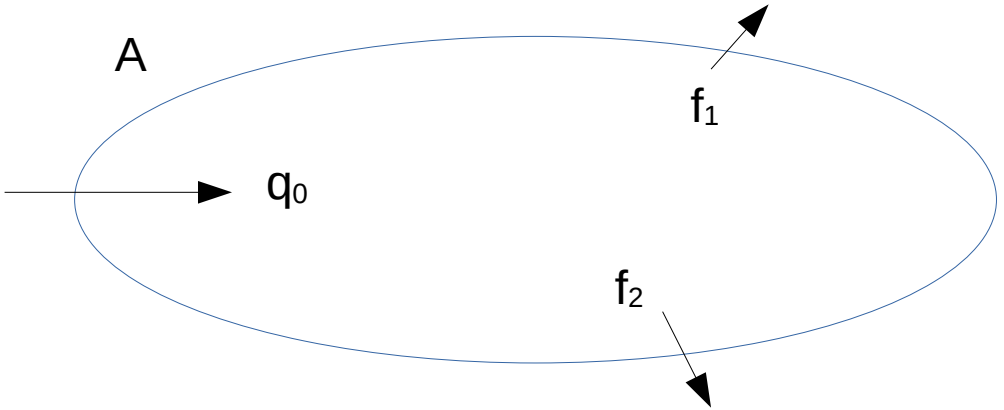


Automates à pile:

de l'acceptation par état final à l'acceptation par pile vide

Soit $A=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ un PDA avec acceptation par état final.
On construit $B=(Q\cup\{q'_0,r\},\Sigma,\Gamma\cup\{Z'_0\},\delta,q'_0,Z'_0,\emptyset)$
Il faut,

- quand on arrive dans un état final, vider la pile : c'est le rôle du nouvel état r
- garantir que seul r est autorisé à vider la pile : un nouveau symbole de fond de pile Z'_0 est empilé au démarrage de B. Seul r est autorisé à dépiler Z'_0

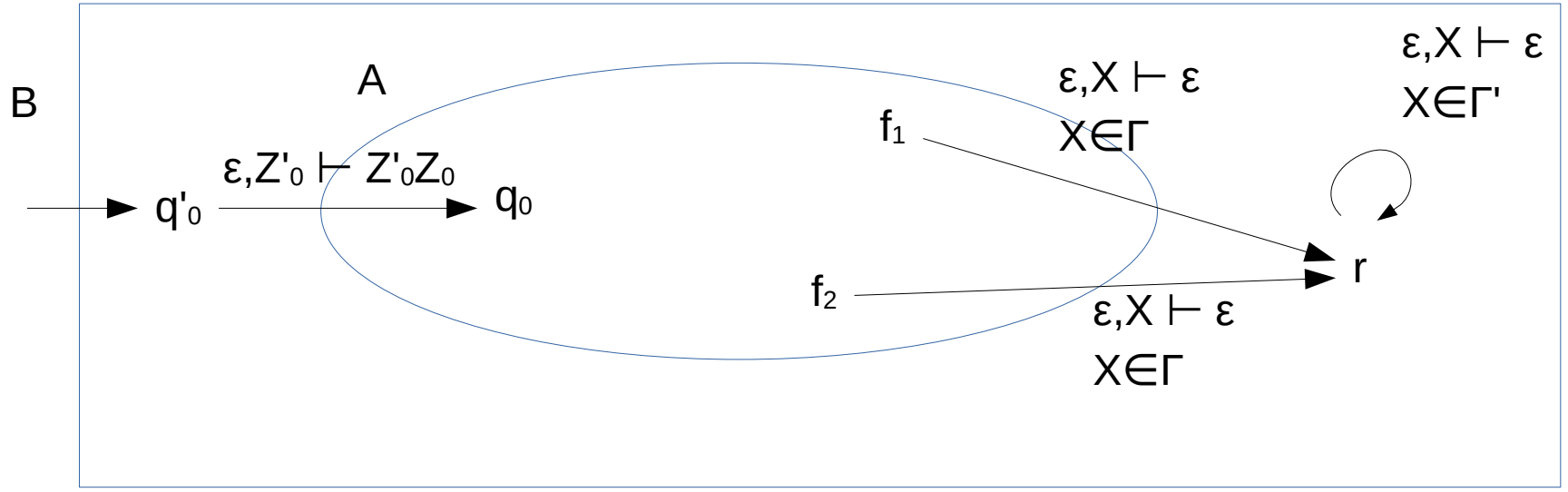


Automates à pile:

de l'acceptation par état final à l'acceptation par pile vide

Soit $A = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ un PDA avec acceptation par état final.
On construit $B =(Q\cup\{q'_0,r\},\Sigma,\Gamma\cup\{Z'_0\},\delta,q'_0,Z'_0,\emptyset)$
Il faut,

- quand on arrive dans un état final, vider la pile : c'est le rôle du nouvel état r
- garantir que seul r est autorisé à vider la pile : un nouveau symbole de fond de pile Z'_0 est empilé au démarrage de B. Seul r est autorisé à dépiler Z'_0



Grammaires

- G=(N,T,S,R) avec
  - N alphabet des « *non terminaux* »
  - T alphabet des « *terminaux* »
  - S un symbole spécial de N, appelé *axiome*
  - R un ensemble fini de *productions* de la forme $\alpha \rightarrow \beta$ avec
    - α un mot sur $N \cup T$ avec au moins un élément de N. α est le *membre gauche* de la production
    - β un mot sur $N \cup T$. β est le *membre droit* de la production
- Langage de G
  - Soit u un mot sur $N \cup T$ avec au moins un élément de N, v un mot de $N \cup T$
  - $u \Rightarrow v$ si u est de la forme $u_1 \alpha u_2$, v est de la forme $v_1 \beta v_2$ et $\alpha \rightarrow \beta \in R$
  - $\Rightarrow^*$ fermeture réflexive et transitive de $\Rightarrow$
  - $L(G)=\{ w \in T^* : S \Rightarrow^* w \}$

Hiérarchie de Chomsky

- Grammaires générales (type 0): pas de restriction sur les productions
  - langages: récursivement énumérables
  - appartenance d'un mot au langage: indécidable
- Grammaires contextuelles (type 1): toute production est de la forme $\alpha A \beta \rightarrow \alpha \gamma \beta$ avec $A \in N$, $\alpha, \beta, \gamma \in (N \cup T)^*$
- Grammaires non contextuelles ou algébriques (type 2) : toute production est de la forme $A \rightarrow \alpha$ avec $A \in N$, $\alpha \in (N \cup T)^*$
  - langages des automates à piles
- Grammaires linéaires : toute production est de la forme $A \rightarrow w_1 B w_2$ ou $A \rightarrow w_3$ avec $A,B \in N$, $w_1, w_2, w_3 \in T^*$
- Grammaires linéaires gauche (resp. droite) (type 3) : toute production est de la forme $A \rightarrow B w_1$ ou $A \rightarrow w_2$ (resp. $A \rightarrow w_1 B$ ou $A \rightarrow w_2$) avec $A,B \in N$, $w_1, w_2 \in T^*$
  - langages des automates finis (de Kleene)

Hiérarchie de Chomsky

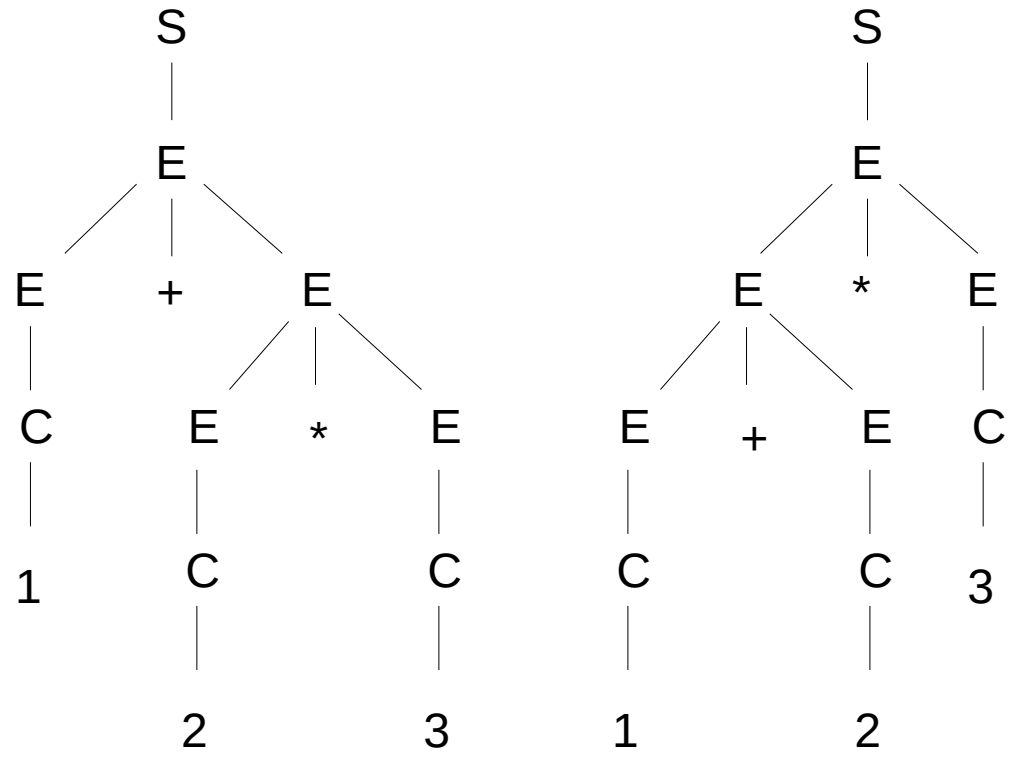
- Grammaires générales (type 0): pas de restriction sur les productions
  - langages: récursivement énumérables
  - appartenance d'un mot au langage: indécidable
- Grammaires contextuelles (type 1): toute production est de la forme $\alpha A\beta \rightarrow \alpha\gamma\beta$ avec $A \in N$, $\alpha,\beta,\gamma \in (N \cup T)^*$
- Grammaires non contextuelles ou algébriques (type 2) : toute production est de la forme $A \rightarrow \alpha$ avec
  $A \in N$, $\alpha \in (N \cup T)^*$
  - langages des automates à piles
- Grammaires linéaires : toute production est de la forme $A \rightarrow w_1Bw_2$ ou $A \rightarrow w_3$ avec $A,B \in N$, $w_1,w_2,w_3 \in T^*$
- Grammaires linéaires gauche (resp. droite) (type 3) : toute production est de la forme $A \rightarrow Bw_1$ ou $A \rightarrow w_2$ (resp. $A \rightarrow w_1B$ ou $A \rightarrow w_2$) avec $A,B \in N$, $w_1,w_2 \in T^*$
  - langages des automates finis (de Kleene)

Grammaires algébriques: arbres de dérivation

- Nœuds internes à étiquettes dans N
- Feuilles à étiquettes dans $N \cup T^*$
- Racine étiquetée par S
- Représentation graphique d'une suite non ordonnée de dérivations
- {S} est un arbre de dérivation
- Si t est un arbre de dérivation, f une feuille de t étiquetée par $X \in N$, $X \rightarrow X_1 \ldots X_n \in R$, alors t dans lequel on ajoute à f la suite ordonnée de fils $X_1, \ldots, X_n$ est un arbre de dérivation

Grammaires algébriques: arbres de dérivation

- Exemple:
  - S→E E→E+E | E-E | E*E | E/E | (E) | C
    C→0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  - 1+2*3

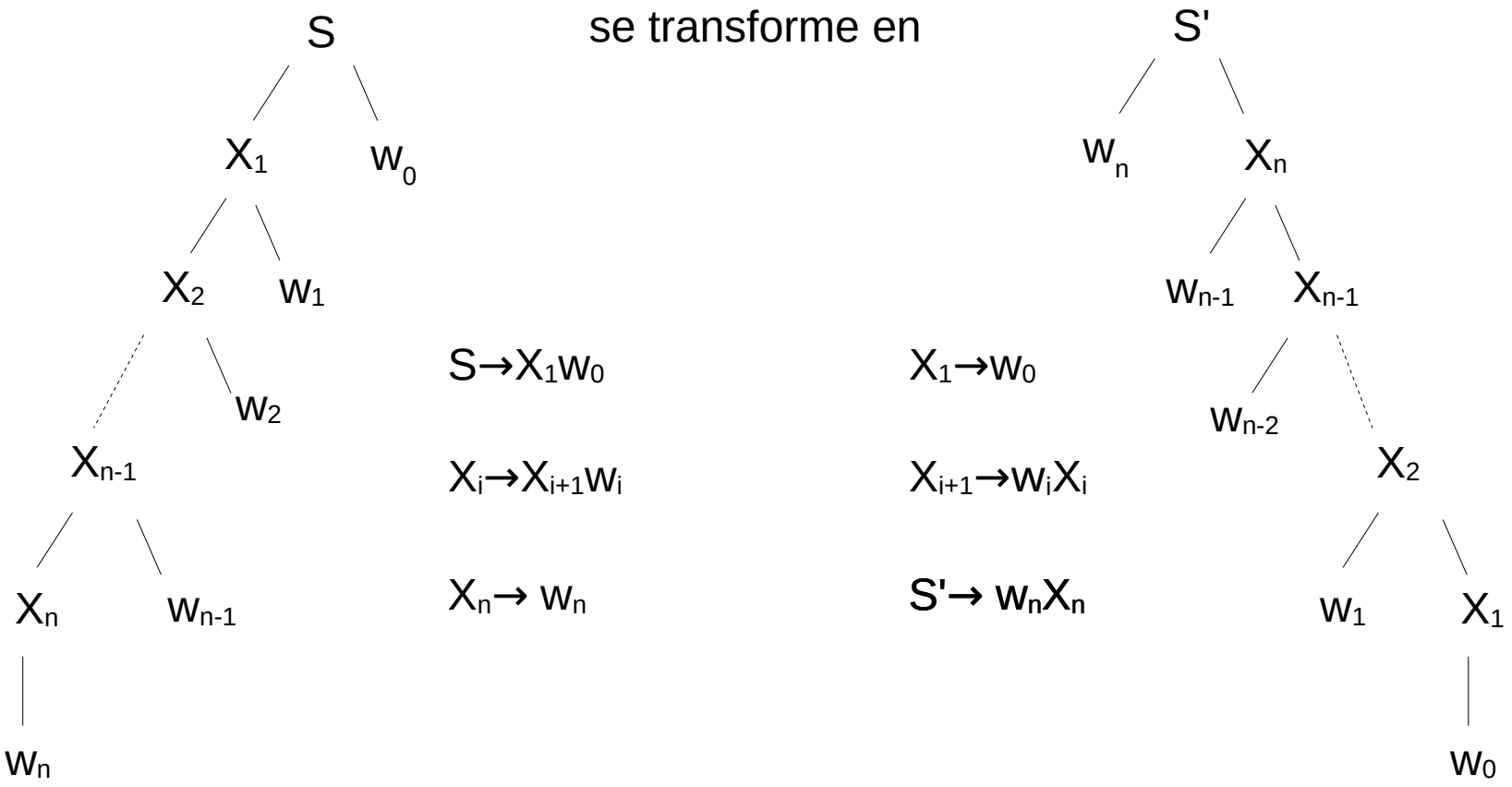


Grammaires de type 3

- L est le langage d'une grammaire linéaire gauche (resp. droite) si et seulement si L est le langage d'une grammaire linéaire droite (resp. gauche)
- Idée de preuve:

Grammaires de type 3

- L est le langage d'une grammaire linéaire gauche (resp. droite) si et seulement si L est le langage d'une grammaire linéaire droite (resp. gauche)
- Idée de preuve:



Les langages de type 3 sont les langages réguliers

- Des langages linéaires droits aux automates
  - On réécrit $X \to a_1 \dots a_n Y$, $a_1 \dots a_n \in T^*$, $n >= 2$ en $X \to a_1 X_1$, $X_i \to a_{i+1} X_{i+1}$, $X_{n-1} \to a_n Y$
  - Automate $(N,T,\delta,\{S\},\{X \in N : X \to \varepsilon\})$
    - $(X,a,Y) \in \delta$ ssi $X \to aY \in R$
- Des automates aux grammaires linéaires droites
  - Construction réciproque (sous hypothèse que l'automate n'ait exactement qu'un état initial)

Grammaires de type 2
Forme normale de Chomsky (FNC)

- Toute production est de la forme $X \to YZ$ ou $X \to a$ ou $S \to \varepsilon$ avec $X,Y,Z \in N$, $a \in T$
- Toute langage de type 2 est le langage d'une grammaire de type 2 en FNC
- Il suffit
  - de transformer la grammaire en supprimant toute production de la forme $X \to \varepsilon$ si $X \neq S$
  - de transformer $X \to X_1 \ldots X_n$, $n>2$, en $X \to X_1 Y_1$, $Y_1 \to X_2 Y_2, \ldots, Y_{n-2} \to X_{n-1} X_n$, en introduisant des nouveaux non-terminaux $Y_1, \ldots, Y_{n-2}$ dans N
  - d'ajouter si nécessaire, pour chaque $a \in T$ qui peut apparaître en membre droit d'une production, un non terminal A et une production $A \to a$ et de faire le remplacement dans les membres droits des productions concernées.

Grammaires de type 2
Forme normale de Greibach (FNG)

- Toute production est de la forme $X \to aX_1 \ldots X_n$ ou $X \to a$ avec $a \in T$, $X_1, \ldots, X_n \in N$.
- Tout langage de type 2 sans ε est le langage d'une grammaire de type 2 en FNG
- Lemme intermédiaire : Supposons que $A \in N$, et que toutes les productions dont A est le membre gauche soient de la forme
  - $A \to A\alpha_1, \ldots, A \to A\alpha_n$
  - $A \to \beta_1, \ldots, A \to \beta_m$

  Alors ces productions se changent, sans modifier le langage, en (B est un nouveau non terminal)
  - $A \to \beta_1 B, \ldots, A \to \beta_m B$
  - $A \to \beta_1, \ldots, A \to \beta_m$
  - $B \to \alpha_1 B, \ldots, B \to \alpha_n B$
  - $B \to \alpha_1, \ldots B \to, \alpha_n$

  Justification : dans l'ancienne grammaire, $A \Rightarrow^* \beta_{i0}\alpha_{i1} \ldots \alpha_{ij}$ si et seulement si, dans la nouvelle, $A \Rightarrow^* \beta_{i0}\alpha_{i1} \ldots \alpha_{ij}$

Grammaires de type 2
Forme normale de Greibach (FNG)

- Tout langage L de type 2 sans ε est le langage d'une grammaire de type 2 en FNG
  - on suppose L donné par une grammaire $G=(N=\{N_1,\ldots,N_m\},T,S,R)$ en FNC
  - on transforme G pour que si $N_i \to N_j\alpha$ est une production, alors $j>i$
    - par induction sur i, on suppose que pour tout $k<i$, $N_k \to N_l\beta$ implique $l>k$
    - dans la production $N_i \to N_j\alpha$, si $j<i$, et $N_j \to N_s\gamma$ (ou $N_j \to a$) alors on remplace $N_i \to N_j\alpha$ par $N_i \to N_s\gamma\alpha$ (ou $N_i \to a\alpha$). On itère jusqu'à ne plus avoir que des productions de la forme $N_i \to N_t\delta$ (ou $N_j \to a$) avec $t \geq i$.
    - On supprime maintenant les productions de la forme $N_i \to N_t\delta$ en utilisant le lemme intermédiaire précédent.

Grammaires de type 2
Forme normale de Greibach (FNG)

- Exemple

Tout langage L de type 2 est définissable par automate à pile

- On suppose L donné par G=(N,T,S,R) en FNG
- M=({q},T,N,δ,q,S,{}) un automate à pile
  - reconnaissance par pile vide
  - $(q,a,X,q,Y) \in \delta$ ssi $X \rightarrow aY \in R$
- On a L(G)=L(M)
  - car M calcule les dérivations les plus à gauche pour les mots de L(G)

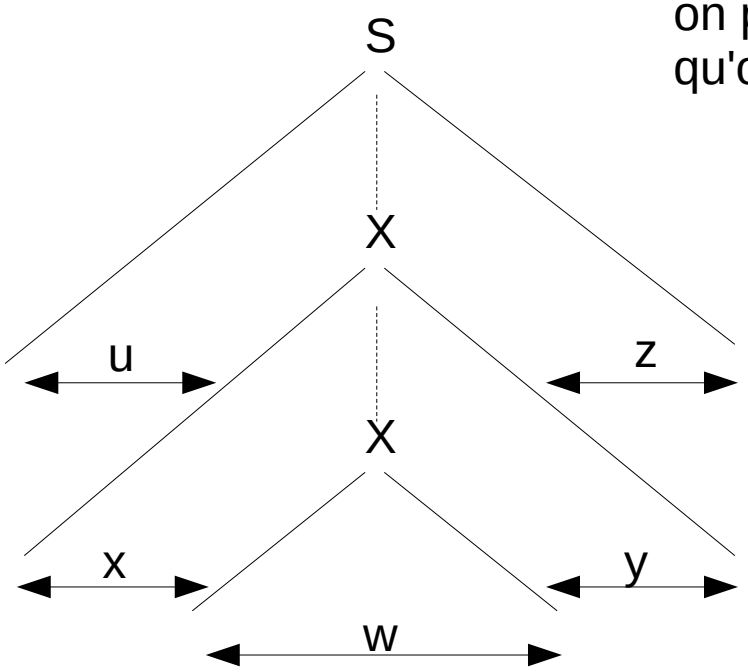
Tout langage L défini par un automate à pile est de type 2

- L est le langage de $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,\{\})$ qui reconnaît par pile vide
- On construit $G = (N,\Sigma,S,R)$ avec
  - $N = \{ [p,X,q] : p,q \in Q, X \in \Gamma \} \cup \{ S \}$
  - R est le plus petit ensemble contenant
    - $S \rightarrow [q_0,Z_0,q]$ pour tout $q \in Q$
    - $[q,X,q_{m+1}] \rightarrow a[q_1,X_1,q_2][q_2,X_2,q_3]...[q_m,X_m,q_{m+1}]$ pour tout $q,q_1,q_2,\ldots,q_{m+1} \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $X,X_1,X_2,\ldots,X_m \in \Gamma$ tels que $(q,a,X,q_1,X_1X_2\ldots X_m) \in \delta$
  - Principe de fonctionnement :
    - $[q,X,q_{m+1}]$ dérive x ssi l'entrée x quand M est dans l'état q supprime X du sommet de pile et va en q_{m+1}
      $[q,X,p] \Rightarrow^* x$ ssi $(q,x,X) \models (p,\varepsilon,\varepsilon)$
    - Preuve par induction sur la longueur de la chaîne de dérivations.

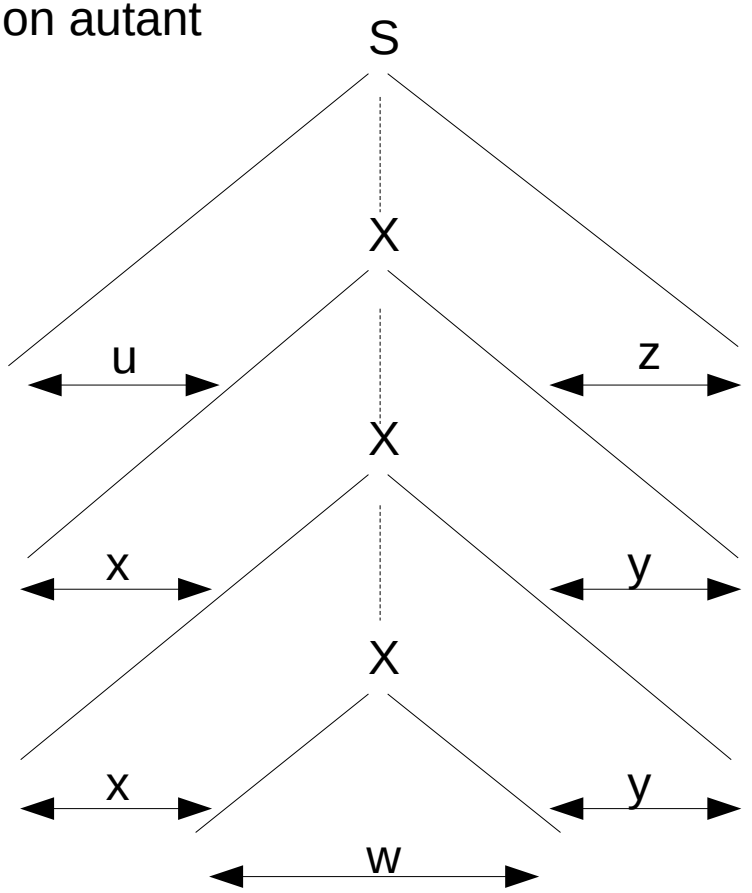
Lemme de la pompe sur les langages algébriques

- Soit L un langage algébrique. Il existe $n \in N$ dépendant de L tel que, si $v \in L$ avec $|v| > n$, alors il existe u,x,w,y,z tels que v = uxwyz avec
  - $|xy| \geq 1$
  - $|xwy| \leq n$
  - $ux^iwy^iz \in L$ pour tout $i \in N$
- On suppose L=L(G) avec G une grammaire de type 2 en FNC
  - tout mot w produit par un arbre de dérivation de hauteur h vérifie $h-1 \leq |w| \leq 2^{h-2}$
  - la hauteur minimale d'un arbre de dérivation pour un mot $w \neq \varepsilon$ est $2+\text{ceil}(\log_2|w|)$
  - la hauteur minimale d'un arbre de dérivation pour un mot $w \neq \varepsilon$ de longueur $2^{|N|}$ est $2+|N|$
    - il contient donc une branche répétant au moins 2 fois un non terminal

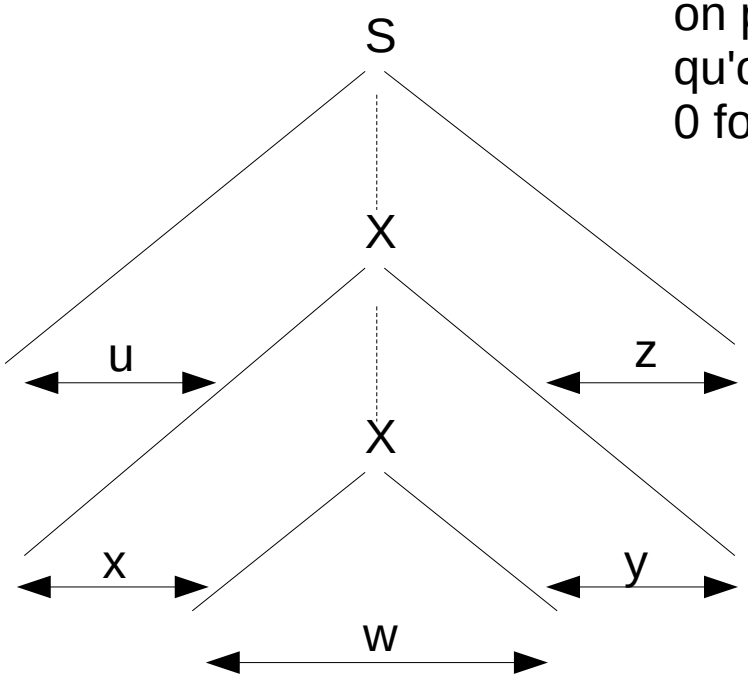
Lemme de la pompe sur les langages algébriques



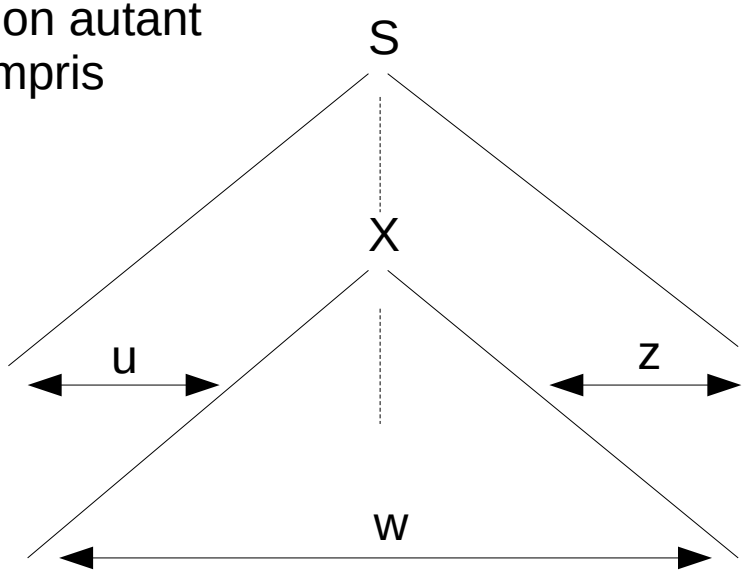
on peut itérer la répétition autant qu'on le souhaite



Lemme de la pompe sur les langages algébriques



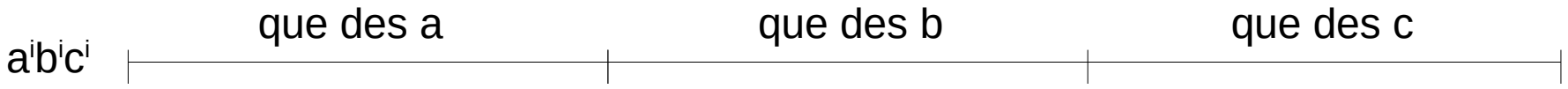
on peut itérer la répétition autant qu'on le souhaite, y compris 0 fois !



Lemme de la pompe: Conséquence

- Il existe des langages non algébriques

Exemple { $a^ib^ic^i$: i>0 }
En effet, supposons qu'il soit algébrique et prenons i suffisamment grand: i>n, où n est la constante du lemme de la pompe



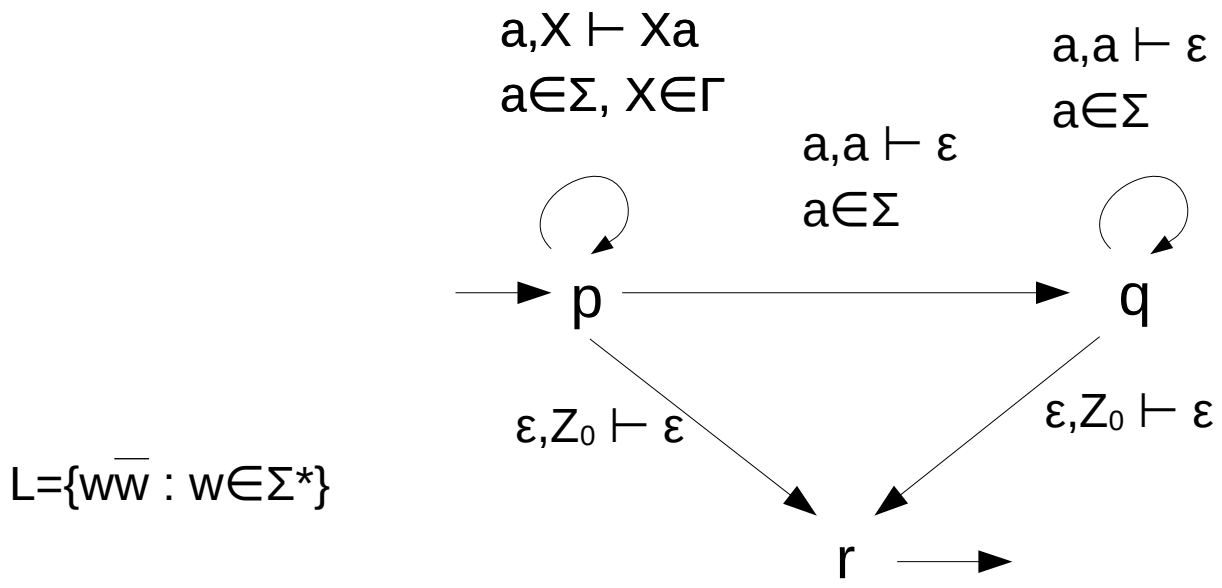
Dans la factorisation de $a^ib^ic^i$=uxwyz du lemme de la pompe, comme |xwy| ≤ n < i, xwy ne contient
- soit que des a, que des b ou que des c
- soit que des a et des b
- soit que des b et des c

Dans tous les cas, itérer x en même temps que y va dissocier le nombre d’occurrences d'une des lettres des deux autres

Quelques propriétés de fermeture des langages algébriques

- Les langages algébriques sont fermés par
  - union: facile
  - concaténation: facile
  - étoile: facile
- mais pas par
  - intersection
    - $L=\{ a^n b^n c^k : n,k>0 \}$ $L'=\{ a^k b^n c^n : n,k>0 \}$ sont tous les deux algébriques mais $L \cap L' = \{ a^n b^n c^n : n>0 \}$ ne l'est pas
  - complément
    - Comme $L \cap L' = C(C(L) \cup C(L'))$ et que les langages algébriques sont fermés par union, s'ils l'étaient aussi par complément ils le seraient également par intersection.

Un langage algébrique non reconnaissable par PDA déterministe



$L=\{w\overline{w} : w\in\Sigma^*\}$

La transition de p vers q est prise quand on passe de w à $\overline{w}$.
Un automate déterministe devrait décider s'il doit commencer à dépiler ou continuer à empiler, c'est-à-dire s'il est à la moitié de la lecture du mot, ce qui n'est pas possible (on peut prendre, par exemple, l'analyse des mots aaaa et aaaaaa)

Machines de Turing « restreintes » automates à 2 piles

Automate à deux piles:
- une bande d'entrée en lecture seule
- deux piles
- un automate de contrôle

- Tout automate à deux piles est équivalent à une MT.

Preuve:
- un automate à deux piles se simule facilement avec une MT à trois bandes
  - une bande d'entrée en lecture seule
  - deux bandes infinies à droite pour simuler les piles
  -
- Simulation d'une MT par un automate à deux piles
  - les deux piles simulent le ruban de la MT coupé en deux. Déplacer la tête vers la gauche, par exemple, consiste à dépiler sur la pile de gauche, et empiler sur celle de droite.

Machines de Turing « restreintes »: automates à 4 compteurs

Compteur = bande infinie à droite avec deux symboles: Z (début de bande), et B
le compteur contient k si la tête est placée sur le kième B
Opérations sur les compteurs: +1, -1, test à 0

Une machine à 4 compteurs peut simuler une MT

Preuve: il suffit de montrer qu'avec deux compteurs on peut simuler une pile. Le reste vient car les MT sont équivalentes aux automates à deux piles

Soit $Z_1,\ldots,Z_{k-1}$ l'alphabet de pile, et $Z_{i1}Z_{i2}\ldots Z_{im}$ le contenu de pile. Il peut être stocké dans un compteur en le codant par une décomposition en base k

$j = i_m+ki_{m-1}+k^2i_{m-2}+\ldots+k^{m-1}i_1$

Opération push Z_r: corresponds à calculer jk+r

Supposons que j soit dans le premier compteur.
On commence par mettre 0 dans le second
Pour chaque symbole B du premier compteur, on en met k dans le second
Le second compteur contient donc jk
On lui ajoute r

Opération pop: symétrique (on divise j par k)
Opération lire sommet de pile: similaire (on calcule j mod k)

Machines de Turing « restreintes »: automates à 2 compteurs

Une machine à 2 compteurs peut simuler une MT

Il suffit de simuler 4 compteurs contenant i,j,k,l avec 2.

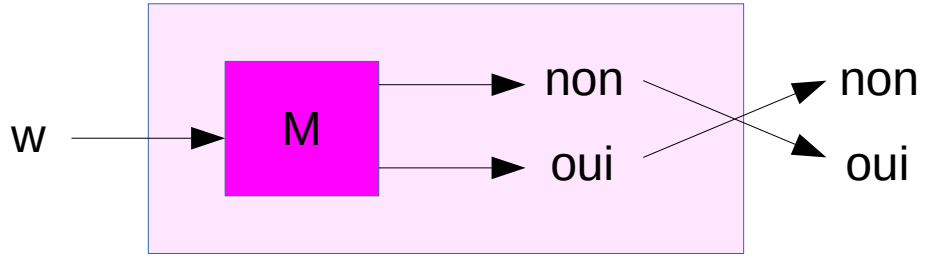
On code les 4 compteurs en un seul avec $2^i3^j5^k7^l$. Comme il s'agit d'un produit de puissance de nombres premiers, I,j,k,l peuvent être retrouvés à partir de $2^i3^j5^k7^l$.

Retirer/ajouter 1 à i,j,k ou l: diviser/multiplier par 2,3,5 ou 7.

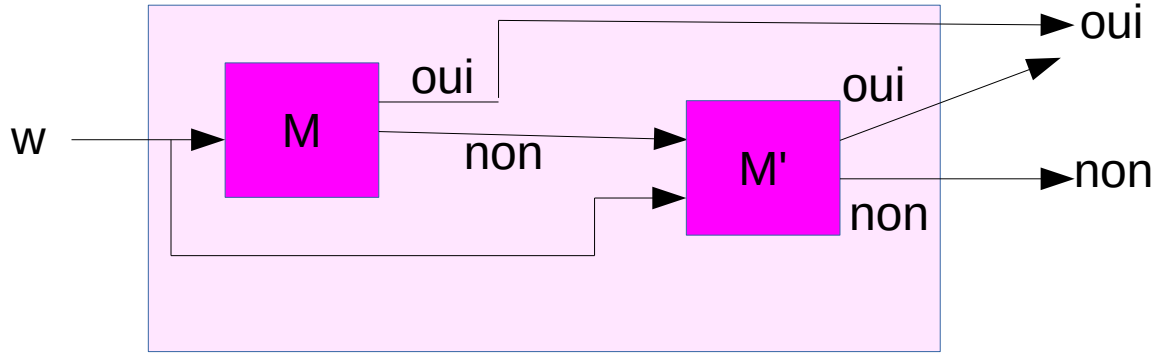
Test à 0 d'un compteur: test de divisibilité par 2,3 5 ou 7

Propriétés des langages récursifs

Le complément d'un langage récursif est récursif

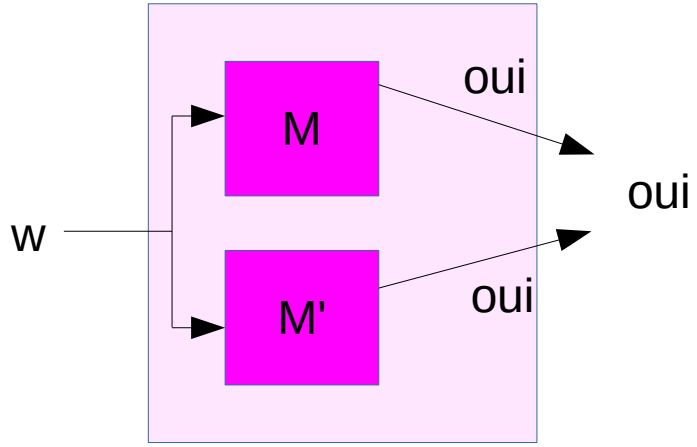


L'union de deux langages récursifs est un langage récursif

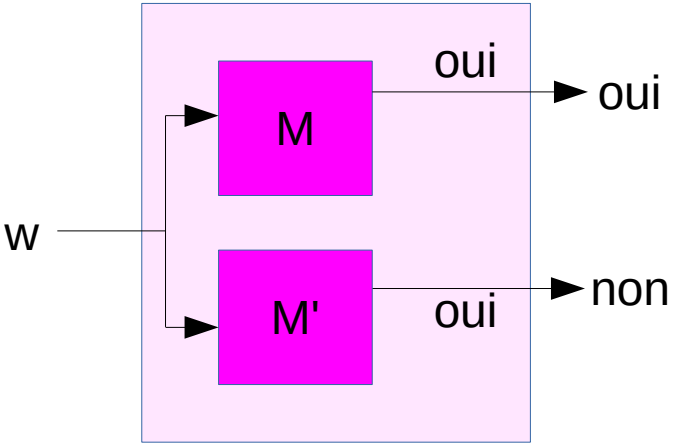


Propriétés des langages r.e.

L'union de deux langages r.e. est r.e.



Si L et C(L) sont r.e., alors L est récursif



Propriétés d'un langage L et de son complémentaire C(L). Soit:

- L et C(L) sont récursifs,
- ni l'un ni l'autre ne sont r.e.,
- l'un des deux est r.e. et non récursif, et alors l'autre n'est pas r.e.

Une MT universelle

- Soit M=(Q,T,I,B,next,i,F) une MT
  - supposons wlog. $Q=\{q_1,\ldots,q_n\}$, T={0,1}, I={0,1,B}, $i=q_1$, $F=q_2$
  - posons $D_1=D$, $D_2=G$, $0=S_1$, $1=S_2$, $B=S_3$
  - Codage d'une transition $t=next(q_i,s_j)=(q_k,s_l,D_m)$: $0^i10^j10^k10^l10^m$
  - Codage du contrôle fini: $111t_111..11t_r111$
    - remarque: les codes des contrôles finis sont en nombre infini dénombrable, donc les MT aussi: $M_0,M_1,...$
- Machine universelle M_u: simule M sur w (quels que soient M et w)
  - une bande pour le contrôle fini de M
  - une bande pour w
  - une bande pour stocker l'état courant de M
  - une ou plusieurs bandes de travail
  - Code de M_u
    - chercher la transition associée à l'état et au symbole d'entrée courants
    - simuler l'exécution de l'action associée
    - accepter ou recommencer si besoin !
- Coût de la simulation : k_M fois le temps de l'exécution de M sur w (k_M constante dépendant de M)
- On peut faire une machine universelle qui simule une MT à r bandes
  - Il faut simuler r bandes sur une seule
  - à chaque transition suivie, il faut parcourir B pour trouver la position de toutes les têtes de la machine simulée
  - coût : $O(k_M*s*t)$ avec s taille max de la bande ($\leq t$) et t temps de calcul de M, donc le coût est en $O(t^2)$

Les MT peuvent être déterminisées:
parcours en largeur de tous les chemins d'exécution

- On construit une MT universelle (déterministe) capable de simuler simultanément toutes les exécutions d'une MT M non déterministe (à une seule bande):
  1) une bande pour la description de M
  2) une bande pour l'entrée w
  3) une bande pour stocker l'ensemble des états courants de M: $q_1Sq_2S...Sq_nS\#$
  4) une bande pour stocker l'ensemble des valeurs de bandes possibles (avec position de la tête) pendant l'exécution de M:
     - $b_1Sb_2S...Sb_nS\#$
     - b_i est la bande correspondant à la ième exécution possible de M (qui est donc dans l'état q_i)
  5) une bande pour faire le calcul de l'ensemble des états suivants de M
  6) une bande pour faire le calcul de l'ensemble des valeurs de bandes possibles pendant l'exécution de M
  7) une bande de travail pour les calculs intermédiaires
- Algorithme
  - on recopie la seconde bande sur la quatrième jusqu'au #
  - on initialise la troisième bande avec $q_1S\#$
  - boucle infinie
    - pour chaque q_i état de la bande 3
      - si q_i est final on s'arrête
      - pour chaque état joignable à partir de q_i
      - recopier b_i à la fin de la bande 6
      - inscrire les états joignables à partir de q_i sur la bande 5, suivis de S
      - simuler l'exécution de M dans l'état q_i en utilisant la bande 6
      - aller à la fin de la bande 6, y inscrire S
    - recopier les bandes 5 et 6 sur 3 et 4

Les MT peuvent être déterminisées:
parcours en profondeur de tous les chemins d'exécution

- On construit une MT universelle (déterministe) capable de simuler simultanément toutes les exécutions d'une MT M non déterministe (à une seule bande. **Hypothèse: M s'arrête toujours**)
- On suppose que les transitions sortantes de chaque état de M sont numérotées 1,2,...
  1) une bande pour la description de M
  2) une bande pour l'entrée w
  3) une bande pour stocker une pile des états de M et des numéros de transitions sortantes en cours d'exploration pour cet état : $(q_1,k_1)S(q_2,k_2)S...S(q_n,0)S\#$
  4) une bande pour stocker la valeur de bande pour ce chemin d'exécution de M:
  5) une bande de travail pour les calculs intermédiaires
- Algorithme
  - on recopie la seconde bande sur la quatrième jusqu'au #
  - on initialise la troisième bande avec $(q_1,0)\#$
  - tant que la pile (troisième bande) n'est pas vide
    - soit (q_n,r) le sommet de pile, qu'on dépile
      - si q_n a une transition sortante r+1 vers un état q_{n+1}
        - empiler $(q_n,r+1),(q_{n+1},0)$
        - simuler l'action de la transition sur la bande
        - si q_{n+1} est final on s'arrête
      - sinon
        - réaliser la simulation arrière de la transition k_{n-1} sortante de q_{n-1}

Il existe des langages non définissables par MT (ie. non r.e.)

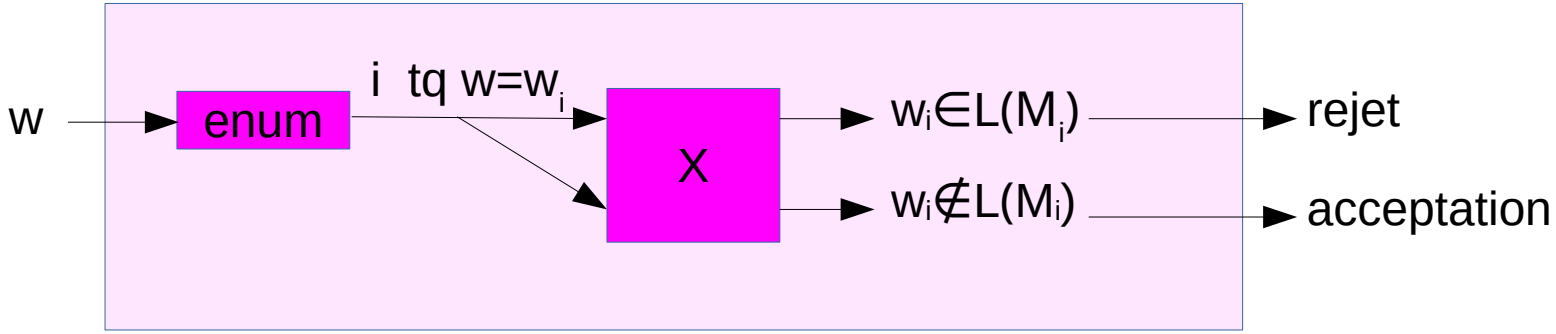
- Colonnes : toutes les MT par ordre militaire
- Lignes : tous les mots finis par ordre militaire
- Matrice :
  - $b_{i,j} = 1$ ssi $w_i \in L(M_j)$

	M_0	M_1	M_2	...	M_i	...
w_0	0	1	0			
w_1	1	0	0			
w_2	0	0	1			
...				...		
w_i					$b_{i,i}$	
...						

- Soit L_d le langage défini par $w_i \in L_d$ ssi $w_i \notin L(M_i)$
- Supposons que L_d soit le langage d'une machine (mettons $L_d = L(M_k)$)
- On a $w_k \in L_d$ ssi $w_k \notin L(M_k)=L_d$: CONTRADICTION !
  Donc L_d n'est le langage d'aucune machine du tableau

Il existe des langages r.e. et non récursifs

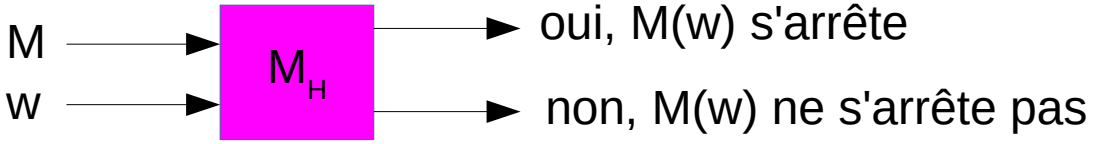
- L_u={ (i,j) : $w_i \in L(M_j)$ }
- L_u est r.e.: il suffit de faire simuler M_j par Mu sur w_i
- L_u n'est pas récursif:
  - par l'absurde, supposons qu'il le soit, et soit X une MT telle que $L(X)=L_u$
  - pour tout w, on peut calculer i tel que $w=w_i$
  - pour tout M, on peut calculer j tel que $M=M_j$
  - on construit une MT Y par :



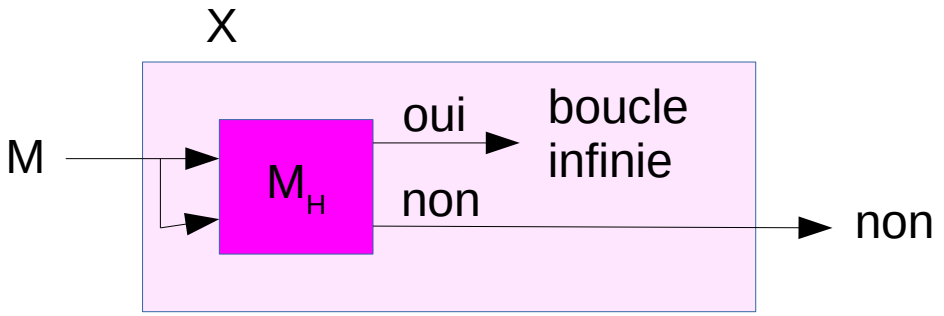
Le langage L(Y) de Y est alors le langage L_d de la diapositive précédente, qui n'est le langage d'aucune MT

Indécidabilité du problème de l'arrêt

- Problème de l'arrêt:
  Existe t-il une MT M_H prenant en entrée une MT M et un mot w, et décidant si oui ou non M s'arrête sur l'entrée w ?
- Par l'absurde, supposons que M_H existe



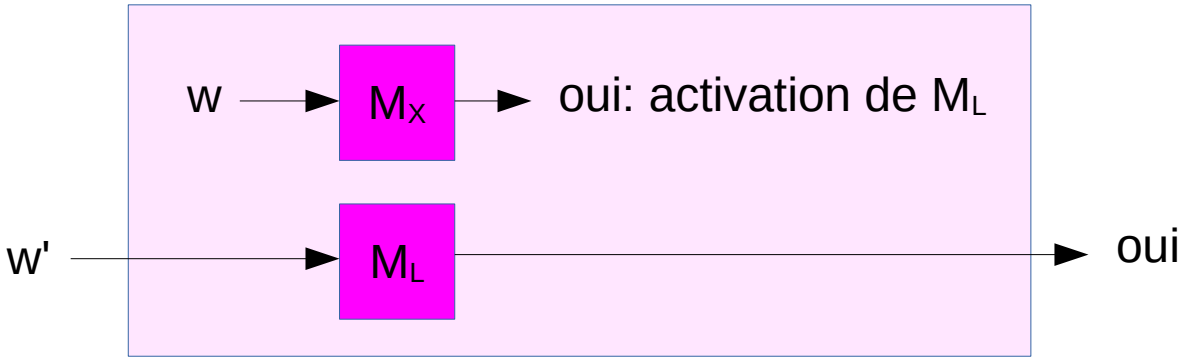
- A partir de M_H, on construit la MT X suivante



- Alors
  - X(M) boucle infiniment ssi M(M) s'arrête
  - X(M) réponds “non” ssi M(M) ne s'arrête jamais
- En appliquant X à son propre code
  - X(X) boucle infiniment ssi X(X) s'arrête !
  - X(X) réponds “non” ssi X(X) ne s'arrête jamais !

Toute propriété P non triviale sur les langages r.e. est indécidable (Rice)

- Propriété P non triviale sur les langages r.e. (ou sur les langages des MT): ensemble de langages r.e. non vide et ne les contenant pas tous
- Théorème de Rice: X∈P est indécidable, X r.e.
  Autre formulation : toute propriété sémantique non triviale est indécidable sur les programmes/MT
  (par opposition aux propriétés syntaxiques, qui elles sont décidables)
- Supposons {}∉P, sinon on prend le complémentaire de P plutôt que P. Comme P est non-triviale, il existe L∈P. On prend w et M_X quelconques. On construit M'(w,M_X,M_L) :



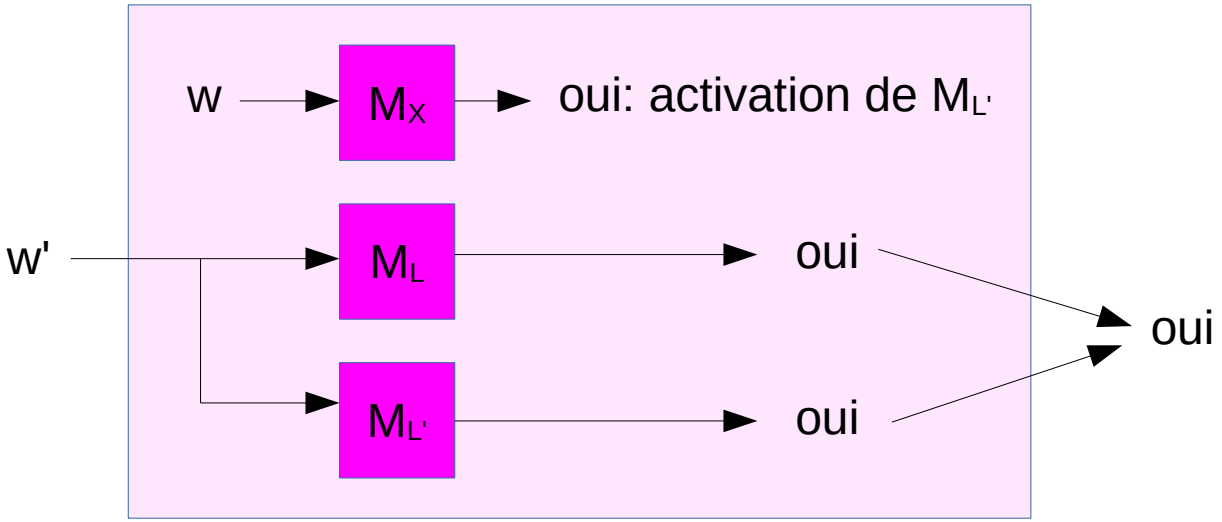
L(M')∈P ssi w∈L(M_X)

Supposons (par l'absurde) qu'on puisse décider si L(M')∈P. Alors L(M_x) est récursif.

- Par exemple, les propriétés suivantes sur les langages r.e. sont indécidables : vide ? fini ? régulier ? sans contexte ?

Si $L \in P$, $L \subseteq L'$, L' r.e. et $L' \notin P$ alors L_P n'est pas r.e.

- On construit M' telle que $L(M') = L'$ si $w \in L(M_x)$, $L(M') = L$ sinon

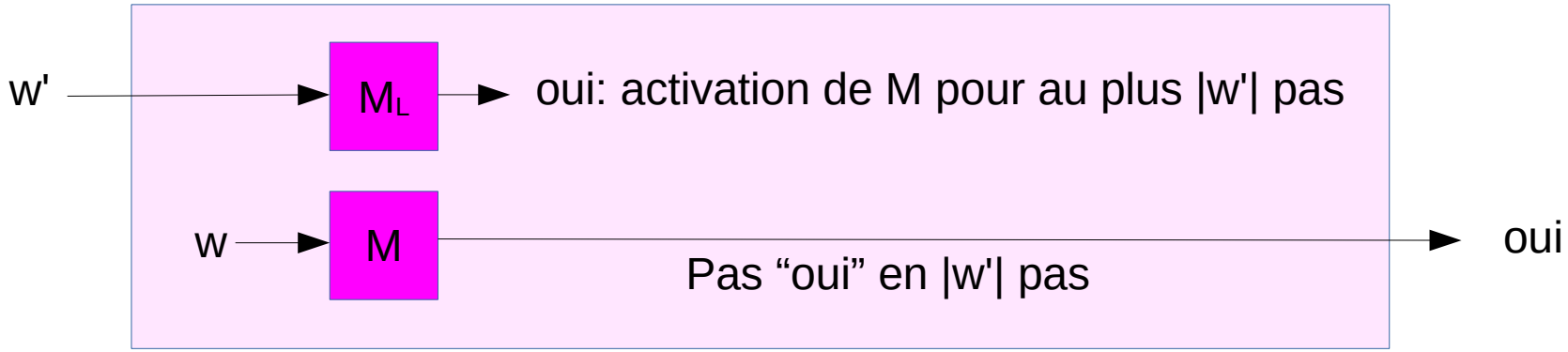


  On a $L(M') \in P$ ssi $w \notin L(M_X)$

- Si L_P est r.e., alors $C(L_u)$ l'est aussi (contradiction, car $C(L_u)$ n'est pas r.e.)

Si L infini, L ∈ P, et L' ∉ P pour tout L' ⊆ L fini, alors L_P n'est pas r.e.

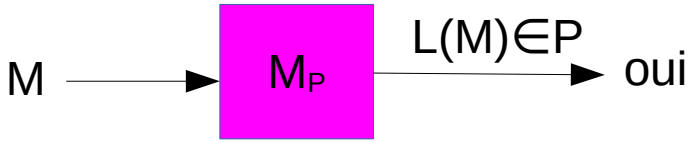
- On construit M'(w,M,L) telle que L(M') soit un sous-ensemble fini de L si w ∈ L(M), L(M') = L sinon.



- Si w ∈ L(M),
  - soit k le nombre de pas de M nécessaires à son acceptation
  - L(M') = { w' ∈ L : |w'|<k } ⊆ L fini et donc L(M') ∉ P
- Sinon L(M') = L ∈ P
- Donc L(M') ∈ P ssi w ∉ L(M)
- En supposant L_P r.e. on a $C(L_u)$ r.e., contradiction

Si L_P est r.e., alors les MT générant les langages finis dans P sont énumérables

- On énumère tous les couples d'entiers (i,j) par ordre croissant de i+j
- Puisque L_P est r.e., il existe une MT M_P



- A chaque (i,j) généré,
  - chaque i (en binaire) est décodé comme un ensemble L_i fini de mots sur {0,1}:
    - 0 = séparateur de mots
    - 10 = code pour 0
    - 11 = code pour 1
  - on exécute j pas de M_P avec M_i=MT codant l'automate fini de L_i en entrée

Caractérisation des propriétés P r.e.

P est r.e. ssi

- si $L \in P$ et $L \subseteq L'$ avec L' r.e., alors $L' \in P$
- si $L \in P$ et L infini, il existe L' fini $\subseteq L$ tel que $L' \in P$
- l'ensemble des langages finis de P est générable (par une MT, sans ordre précis)

Sens de la gauche vers la droite montré dans les diapos précédentes. Pour l'autre sens, on construit une MT M_P telle $L(M_P) = \{ M : L(M) \in P \}$ en utilisant les 3 propriétés :

M_g : générateur des couples d'entiers (i,j) par ordre croissant de i+j

M_f : énumère les langages finis de P, wlog. par ordre croissant de la somme des longueurs des mots

Pour chaque (i,j) fourni par M_g

Exécuter M_f sur i pas

Si M_f a terminé la génération d'un nouveau langage fini L de P

Exécuter M sur j pas et sur L

Si M a accepté L

Accepter M

Corollaires

- Les propriétés suivantes des langages r.e. ne sont pas r.e.:
  - L est vide
  - C(L) est vide
  - L est récursif
  - L n'est pas récursif
  - L est un singleton
  - L est régulier
  - L-L_u n'est pas vide
- Les propriétés suivantes des langages r.e. sont r.e.:
  - L n'est pas vide
  - L contient k mots au moins
  - L'intersection de L et de L_u n'est pas vide

Problème de la Correspondance de Post (PCP, 1946)

- Deux listes de mots $L_1=u_1,\ldots,u_n$; $L_2=v_1,\ldots,v_n$
- PCP: existe t-il une suite d'indices $i_1,\ldots,i_k$ ($k>0$) telle que $u_{i1}u_{i2}\ldots u_{ik}=v_{i1}v_{i2}\ldots v_{ik}$?
- Exemple: L_1=1,10111,10; L_2=111,10,0

- Exemple: L_1=10,011,101; L_2=101,11,011
  -
  -
  -
  -

Problème de la Correspondance de Post (PCP, 1946)

- Deux listes de mots $L_1=u_1,\ldots,u_n$; $L_2=v_1,\ldots,v_n$
- PCP: existe t-il une suite d'indices $i_1,\ldots,i_k$ ($k>0$) telle que $u_{i1}u_{i2}\ldots u_{ik}=v_{i1}v_{i2}\ldots v_{ik}$?
- Exemple: L_1=1,10111,10; L_2=111,10,0

  10111-1-1-10 ;10-111-111-0 (indices 2,1,1,3)
- Exemple: L_1=10,011,101; L_2=101,11,011
  Pas de solution ! En effet,
  - on commence nécessairement par 10-; 101
  - on continue nécessairement par 10-101-;101-011
  - on continue nécessairement par 10-101-101-;101-011-011
  - et encore, indéfiniment

PCP est indécidable

- On montre que s'il l'était, L_u serait récursif.
- Soient $M=(Q,\Gamma,\Sigma,\#,\delta,q_0,F)$ une MT, w un mot sur $\Sigma=\{0,1\}$.
- PCP est utilisé pour construire une exécution de M acceptant w
- On construit deux listes L_A et L_B à partir de M par

Grp	L_A	L_B	
	#	$\#q_0w\#$	
1	X	X	pour tout $X\in\Gamma$
1	#	#	
2	qX	Yp	si $\delta(q,X)=(p,Y,D)$
2	ZqX	pZY	si $\delta(q,X)=(p,Y,G)$
2	q#	Yp#	si $\delta(q,B)=(p,Y,D)$
2	Zq#	pZY#	si $\delta(q,B)=(p,Y,G)$
3	XqY	q	pour tout $q\in F$, $X,Y\in\Gamma$
3	Xq	q	pour tout $q\in F$, $X,Y\in\Gamma$
3	qY	q	pour tout $q\in F$, $X,Y\in\Gamma$
4	q##	#	pour tout $q\in F$

PCP est décidable sur L_A et L_B ssi $w\in L(M)$ est décidable

De plus toute solution représente une exécution de M acceptant w

Les différentes étapes de l'exécution sont séparées par #

PCP est indécidable

- On utilise une version particulière (et équivalente) de PCP dans laquelle le premier indice est choisi: ici, la configuration de départ est #-;#q_0w#-
- Le couple de mots (x,y) est une *solution partielle* si x est un préfixe de y. Si xz=y alors z est le *reste* à calculer pour obtenir une solution
- Le groupe 1 sert à recopier la bande

Grp	L_A	L_B	
	#	#q_0w#	
1	X	X	pour tout X∈Γ
1	#	#	
2	qX	Yp	si δ(q,X)=(p,Y,D)
2	ZqX	pZY	si δ(q,X)=(p,Y,G)
2	q#	Yp#	si δ(q,B)=(p,Y,D)
2	Zq#	pZY#	si δ(q,B)=(p,Y,G)
3	XqY	q	pour tout q∈F, X,Y∈Γ
3	Xq	q	pour tout q∈F, X,Y∈Γ
3	qY	q	pour tout q∈F, X,Y∈Γ
4	q##	#	pour tout q∈F

- Le groupe 2 modélise les transitions
- Le groupe 3 sert à réduire quand un état final a été rencontré
- Le groupe 4 sert à terminer

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

#- ; #$q_1$01#-

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

Situation initiale:
#-
#configuration instantanée de départ

Pour continuer, il faut un domino qui place
en haut $q_1$0
en bas ce qui correspond à l'exécution de la transition
(exécution de transition: groupe 2)

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$0**1#1q_2**-

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

Situation initiale:
#-
#configuration instantanée de départ

Pour continuer, il faut un domino qui place
en haut $q_1$0
en bas ce qui correspond à l'exécution de la transition
(exécution de transition: groupe 2)

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#**1$q_2$1**-

On recopie en haut et en bas, jusqu'à retrouver un état dans la prochaine configuration instantanée du bas

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#**1$q_2$1**#-

On recopie en haut et en bas, jusqu'à retrouver un état dans la prochaine configuration instantanée du bas

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10
2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1**$q_2$1#1**-

On recopie en haut et en bas, jusqu'à retrouver un état dans la prochaine configuration instantanée du bas

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#**10q_1**-

Et on recommence (groupe 2)

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#**10q_1**#-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10
2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#1**0q_1#1**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#**1$q_2$01#**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$0**1#$q_3$10**-

Exemple

$M=(\{q_1,q_2,q_3\},\{0,1,B\},\{0,1\},\delta,q_1,B,\{q_3\})$,
$w=01$

δ	0	1	B
q_1	$(q_2,1,D)$	$(q_2,0,G)$	$(q_2,1,G)$
q_2	$(q_3,0,G)$	$(q_1,0,D)$	$(q_2,0,D)$

Grp	L_A	L_B
	#	$\#q_101\#$
1	0	0
1	1	1
1	#	#
2	q_10	$1q_2$
2	$0q_11$	q_200
2	$1q_11$	q_210
2	$0q_1\#$	$q_201\#$
2	$1q_1\#$	$q_211\#$
2	$0q_20$	q_300
2	$1q_20$	q_310
2	q_21	$0q_1$
2	$q_2\#$	$0q_2\#$
3	$0q_30$	q_3
3	$0q_31$	q_3
3	$1q_30$	q_3
3	$1q_31$	q_3
3	$0q_3$	q_3
3	$1q_3$	q_3
3	q_30	q_3
3	q_31	q_3
4	$q_3\#\#$	#

$\#-$; $\#q_101\#-$
$\#q_10-$; $\#q_101\#1q_2-$
$\#q_101-$; $\#q_101\#1q_21-$
$\#q_101\#-$; $\#q_101\#1q_21\#-$
$\#q_101\#1-$; $\#q_101\#1q_21\#1-$
$\#q_101\#1q_21-$; $\#q_101\#1q_21\#10q_1-$
$\#q_101\#1q_21\#-$; $\#q_101\#1q_21\#10q_1\#-$
$\#q_101\#1q_21\#1-$; $\#q_101\#1q_21\#10q_1\#1-$
$\#q_101\#1q_21\#10q_1\#-$; $\#q_101\#1q_21\#10q_1\#1q_201\#-$
$\#q_101\#1q_21\#10q_1\#1q_20-$; $\#q_101\#1q_21\#10q_1\#1q_201\#q_310-$
$\#q_101\#1q_21\#10q_1\#1q_201-$; $\#q_101\#1q_21\#10q_1\#1q_201\#\mathbf{q_3101}-$

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#10q_1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#**$q_3$101**#-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1**01#q_3**-

Le mot du haut est plus grand que le mot du bas.
Une fois l'état final détecté, on augmente le mot du haut sans changer celui du bas
(groupe 3 une fois par configration instantanée et groupe 1 et on recommence sur la prochaine configuration instantanée)

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10**1#$q_3$0**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101**#$q_3$01**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10
2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#**$q_3$01**#-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0**1#q_3**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#**$q_3$1**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#**$q_3$1#**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#**q_3**-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#**q_3**#-

Exemple

M=({q_1,q_2,q_3},{0,1,B},{0,1},δ,q_1,B,{q_3}),
w=01

δ	0	1	B
q_1	(q_2,1,D)	(q_2,0,G)	(q_2,1,G)
q_2	(q_3,0,G)	(q_1,0,D)	(q_2,0,D)

Grp	L_A	L_B
	#	#$q_1$01#
1	0	0
1	1	1
1	#	#
2	$q_1$0	1q_2
2	0$q_1$1	$q_2$00
2	1$q_1$1	$q_2$10
2	0q_1#	$q_2$01#
2	1q_1#	$q_2$11#
2	0$q_2$0	$q_3$00
2	1$q_2$0	$q_3$10

2	$q_2$1	0q_1
2	q_2#	0q_2#
3	0$q_3$0	q_3
3	0$q_3$1	q_3
3	1$q_3$0	q_3
3	1$q_3$1	q_3
3	0q_3	q_3
3	1q_3	q_3
3	$q_3$0	q_3
3	$q_3$1	q_3
4	q_3##	#

#- ; #$q_1$01#-
#$q_1$0- ; #$q_1$01#1q_2-
#$q_1$01- ; #$q_1$01#1$q_2$1-
#$q_1$01#- ; #$q_1$01#1$q_2$1#-
#$q_1$01#1- ; #$q_1$01#1$q_2$1#1-
#$q_1$01#1$q_2$1- ; #$q_1$01#1$q_2$1#10q_1-
#$q_1$01#1$q_2$1#- ; #$q_1$01#1$q_2$1#10q_1#-
#$q_1$01#1$q_2$1#1- ; #$q_1$01#1$q_2$1#10q_1#1-
#$q_1$01#1$q_2$1#10q_1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$1- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$10- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$0- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q_3-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q_3#- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q_3#-
#$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q_3##- ; #$q_1$01#1$q_2$1#10q_1#1$q_2$01#$q_3$101#$q_3$01#$q_3$1#q_3##-

L'utilisation du groupe 4 permet de terminer
L'alignement des 2 mots

Il existe un alignement des mots haut et bas si et seulement si il existe une exécution acceptante

L'ambiguïté d'une CFG est indécidable

- On montre que si ce problème était décidable, alors PCP le serait aussi
- Soient
  - $L_A = \{ u_1,\ldots,u_n \}$ et $L_B = \{ v_1,\ldots,v_n \}$ deux listes de mots sur Σ
  - $a_1,\ldots,a_n$ des nouvelles lettres (qui ne sont pas dans Σ, et toutes différentes)
  - $X_A = \{ u_{i1}u_{i2}...u_{im}a_{im}...a_{i2}a_{i1} : m>0 \}$, $X_B = \{ v_{i1}v_{i2}...v_{im}a_{im}...a_{i2}a_{i1} : m>0 \}$
  - $G : S \rightarrow S_A \mid S_B$, $S_A \rightarrow u_iS_Aa_i \mid u_ia_i$, $S_B \rightarrow v_iS_Ba_i \mid v_ia_i$ pour tout $1 \leq i \leq n$
    - $L(S_A) = X_A$, $L(S_B) = X_B$, $L(S) = X_A \cup X_B$
- Si PCP a une solution sur (L_A,L_B), alors G est ambiguë
- Si G est ambiguë, alors il existe $w_{i1}...w_{ik}a_{ik}...a_{i1} \in L(S_A) \cap L(S_B)$, et la suite $i_1,...i_k$ donne une solution à PCP sur (L_A,L_B)

$L(G) \cap L(G')$ vide indécidable, G,G' CFG

- Réduction à L(M) vide, $M=(Q,\Gamma,\Sigma,\delta,q_0,B,F)$ MT
- Calcul acceptant de M: $q_0w'_0\#(w_1q_1w'_1)^R\#...\#w_kq_kw'_k$ avec $w_i,w'_i \in \Gamma$, $q_k \in F$,
  - $w_iq_iw'_i\#(w_{i+1}q_{i+1}w'_{i+1})^R$ cohérent avec δ pour tous les i pairs
  - $(w_iq_iw'_i)^R\#w_{i+1}q_{i+1}w'_{i+1}$ cohérent avec δ pour tous les i impairs
- $L=\{uqu'\#(vpv')^R : u,u',v,v' \in \Gamma, p,q \in Q$, cohérent avec $\delta\}$ est reconnaissable par un PDA (donc, un CFL)
  - le PDA empile *uqu'* en vérifiant qu'il a exactement un état
  - il parcourt ensuite $(vpv')^R$ en s'assurant qu'il s'agit de $(uqu')^R$ à l'application d'une transition de δ près
- $L'=\{(uqu')^R\#vpv' : u,u',v,v' \in \Gamma, p,q \in Q$, cohérent avec $\delta\}$ est aussi un CFL (argumentation similaire)
- Soient $L_1=(L\#)^*(\Gamma^*F\Gamma^* \cup \{\varepsilon\})$, $L_2=q_0\Sigma^*\#(L'\#)^*(\Gamma^*F\Gamma^* \cup \{\varepsilon\})$.
  Soient G_1 et G_2 CFG tels que $L(G_1)=L_1$ et $L(G_2)=L_2$.
- L'ensemble des calculs acceptants de M est $L(G_1) \cap L(G_2)$
- $L(G_1) \cap L(G_2)$ est vide ssi L(M) l'est aussi

$L(G)=\Sigma^*$ indécidable, G CFG

- $C(L(G_1)\cap L(G_2))$ est un CFL car $w\in C(L(G_1)\cap L(G_2))$ ssi
  - w n'est pas de la forme $w_1\#...\#w_n$ avec $w_i\in\Gamma^*Q\Gamma^*$ (langage rationnel)
  - l'état apparaissant dans w_1 n'est pas initial (langage rationnel)
  - l'état apparaissant dans w_n n'est pas final (langage rationnel)
  - $w_iq_iw'_i\#(w_{i+1}q_{i+1}w'_{i+1})^R$ non cohérent avec δ pour un i pair (CFL)
  - $(w_iq_iw'_i)^R\#w_{i+1}q_{i+1}w'_{i+1}$ non cohérent avec δ pour un i impair (CFL)

  et l'union de tous ces langages est un CFL
- $C(L(G_1)\cap L(G_2))=\Sigma^*$ ssi L(M) est vide
- Si $L(G)=\Sigma^*$ était décidable pour tout G CFG, alors L(M) est vide le serait aussi (contradiction)
- (On rappelle au passage que L(G) vide est décidable)
  - il suffit de transformer G en supprimant les non-terminaux ne produisant rien

D'autres résultats d'indécidabilité sur les CFL

- Soient G_1, G_2 des CFL, et R un langage rationnel. Les questions suivantes sont indécidables:
  - $L(G_1)=L(G_2)$

    Si ça l'était, en prenant G_2 tq $L(G_2)=\Sigma^*$, on aurait $L(G_1)=\Sigma^*$ décidable
  - $L(G_1)=R$

    Si ça l'était, en prenant R tq $L(R)=\Sigma^*$, on aurait $L(G_1)=\Sigma^*$ décidable
  - $L(G_2)\subseteq L(G_1)$

    Si ça l'était, en prenant G_2 tq $L(G_2)=\Sigma^*$, on aurait $L(G_1)=\Sigma^*$ décidable
  - $R\subseteq L(G_1)$

    Si ça l'était, en prenant R tq $L(R)=\Sigma^*$, on aurait $L(G_1)=\Sigma^*$ décidable
  - $C(L(G_1))$ CFL (par des arguments qu'on ne développera pas)
  - $L(G_1)\cap L(G_2)$ CFL (par des arguments qu'on ne développera pas)
  - $L(G_1)$ rationnel (par des arguments qu'on ne développera pas)

Fonctions calculables

- Une fonction partielle $f:N^k \rightarrow N$ est **calculable** s'il existe une MT prenant en entrée $1^{v1}01^{v2}0...01^{vk}0$ et
  - s'arrêtant avec $1^{f(v1,...,vk)}0$ sur sa bande si $f(v_1,\ldots,v_k)$ est définie
  - ne s'arrêtant pas si $f(v_1,\ldots,v_k)$ n'est pas définie
- Il existe des fonctions non calculables...
  - l'ensemble des fonctions $N^k \rightarrow N$ est indénombrable
  - celui des MT l'est
- … et on peut en montrer !

source:https://commons.wikimedia.org/wiki/File:American_Beaver.jpg

Le castor affairé (Tibor Radó, 1962)

- X(n)={ MT à n+1 états $q_1,\ldots,q_n,q_{n+1}$, $\Gamma=\{0,1\}$, q_1 état initial, q_{n+1} état final sans transition sortante }
  - $|X(n)|=(4(n+1))^{2n}$ quand $n \geq 2$
- Castor à n états: MT de X(n) s'arrêtant sur une bande bi-infinie initialement remplie de 0
  - on a toujours au moins un castor à n états quand $n \geq 2$ (il suffit de produire un 1 et de s'arrêter)
- Score s(c) d'un castor: nombre de 1 sur la bande quand la machine s'arrête
- C(n)=max { s(c): c castor dans X(n) }
  - Remarque: $C(n+1) \geq C(n)$, puisque en ajoutant un état inutile à un castor de X(n) on obtient un castor de X(n+1)
- Exemple

Castor c de X(1) avec s(c)=C(1)=1

δ	0	1
q_1	$(1,q_2,D)$	
q_2		

Castor c de X(2) avec s(c)=C(2)=4

δ	0	1
q_1	$(1,q_2,D)$	$(1,q_2,G)$
q_2	$(1,q_1,G)$	$(1,q_3,D)$
q_3		

source:https://commons.wikimedia.org/wiki/File:American_Beaver.jpg

Le castor affairé (Tibor Radó, 1962)

Castor c de X(3) avec s(c)=C(3)=6

δ	0	1
q_1	$(1,q_2,D)$	$(1,q_4,D)$
q_2	$(0,q_3,D)$	$(1,q_2,D)$
q_3	$(1,q_3,G)$	$(1,q_1,G)$
q_4		

Castor c de X(4) avec s(c)=C(4)=13

δ	0	1
q_1	$(1,q_2,D)$	$(1,q_2,G)$
q_2	$(1,q_1,G)$	$(0,q_3,G)$
q_3	$(1,q_5,D)$	$(1,q_4,G)$
q_4	$(1,q_4,D)$	$(0,q_1,D)$
q_5		

Pour 5 états (de nombreuses années pour prouver):

- C(5)=4098
- 2 MT possibles (en forme normale) : une exécute 11798826 étapes, l'autre 47176870
- Le plus grand nombre d'étapes pour 5 états est 47176870 (preuve par énumération de tous les cas, en COQ, 2024, projet bbchallenge.org)

À partir de 6 états: on ne sait pas ! mais

- pour 6 états, on a un castor c avec $s(c)=3515\times10^{18267}$ (on ne sait pas si $C(6)=3515\times10^{18267}$)
- on sait que $C(2k)>3\uparrow^{k-2}3 > Ack(k-2,k-2)$ pour tout $k \geq 2$

source:https://commons.wikimedia.org/wiki/File:American_Beaver.jpg

Le castor affairé (Tibor Radó, 1962)

- Si M est une MT calculant une fonction (totale) f_M, alors $C(|Q_M|+2n+2) \geq f_M(n)$ pour tout n
  - On construit un castor c de $X(|Q_M|+2n+2)$ à partir de M par
    - on écrit 1^n0 sur la bande
    - on revient sur le premier 1 de la suite
    - on rend le contrôle à M
  - On a bien $s(c) = \max(n,f_M(n))$, donc $C(|Q_M|+2n+2) \geq f_M(n)$
- C n'est pas calculable
  - si elle l'était, il existerait M MT telle que $f_M=C$, donc une MT M' telle que $f_{M'}(n) = C(3n)$
  - on aurait $C(|Q_{M'}|+2n+2) \geq f_{M'}(n) = C(3n)$
  - pour tout $n \geq 2+|Q_{M'}|$, on a donc $C(3n) = C(|Q_{M'}|+2n+2)$, donc C est constante
  - en prenant maintenant M'' MT telle que $f_{M''}(n) = n$, on a $C(|Q_{M''}|+2n+2) \geq n$
  - donc C ne peut pas être constante
- Il n'est pas décidable si un castor c à n états vérifie $s(c)=C(n)$
  - si c'était vrai, alors C(n) serait calculable, puisque c la donne !

Théorèmes d'itération et de récursion (Kleene)

- Théorème Smn (ou d'itération, Kleene 1943):
  il existe une fonction (totale) calculable s prenant en entrée le code d'une machine M et un mot w et telle que, pour tout mot w', $M_u(s(M,w),w') \equiv M(w,w')$
  - s(M,w) est le code de la machine qui insère w# avant w' sur son ruban d'entrée, et qui passe le contrôle à M
  - s est totale et calculable
- Théorème de récursion (ou du point fixe, Kleene 1938) : soit f une fonction totale calculable. Il existe un code de MT M tel que $M_u(f(M)) \equiv M_u(M)$
  - En d'autres termes, si on modifie toutes les MT d'une certaine façon, il existe toujours une MT équivalente à la MT non modifiée
  - Soit $M(x,y) \equiv M_u(f(s(x,x)),y)$. On a $M_u(M) \equiv M$, et $M(M,y) \equiv M_u(f(s(M,M)),y)$. Par le théorème Smn on a $M(M,y) \equiv M_u(s(M,M),y)$, donc les MT d'indices respectifs f(s(M,M)) et s(M,M) sont équivalentes

Applications

- Soit f la fonction prenant en entrée le code d'une MT M et renvoyant le code de la MT M' vérifiant M'(0) = 1 et M'(n) = n*M(n-1) quand n > 0.
  - d'après le Théorème du point fixe, il existe M telle que $M_u(M) \equiv M_u(f(M))$, et donc qui calcule fact(n)
- Soit f la fonction prenant en entrée le code d'une MT M et renvoyant le code de la MT M' qui affiche le code de M
  - d'après le Théorème du point fixe, il existe une MT qui affiche son propre code

Complexité algorithmique

- Classer les fonctions f(x) calculées par MT par
  - le temps (nombre d'étapes de calcul) nécessaire pour calculer f(x)
  - l'espace (le nombre de cases de la bande) nécessaire pour calculer f(x)
    - on ne comptera pas la taille de l'entrée x
  - autres ressources...
- Essayer de classer les fonctions en « difficiles » et « faciles »
  - classes P et NP
  - réductions, NP-complétude, …
- Comme à partir de maintenant on s'intéresse aux ressources consommées par un calcul, et plus à l'existence du calcul, on supposera que les fonctions analysées sont toutes calculables
  - Toute MT s'arrête
  - On peut changer un petit peu la définition des MT pour que l'arrêt se fasse
    - dans un état d'acceptation
    - dans un état de rejet

  $F = \{ q_{accept}, q_{reject} \}$

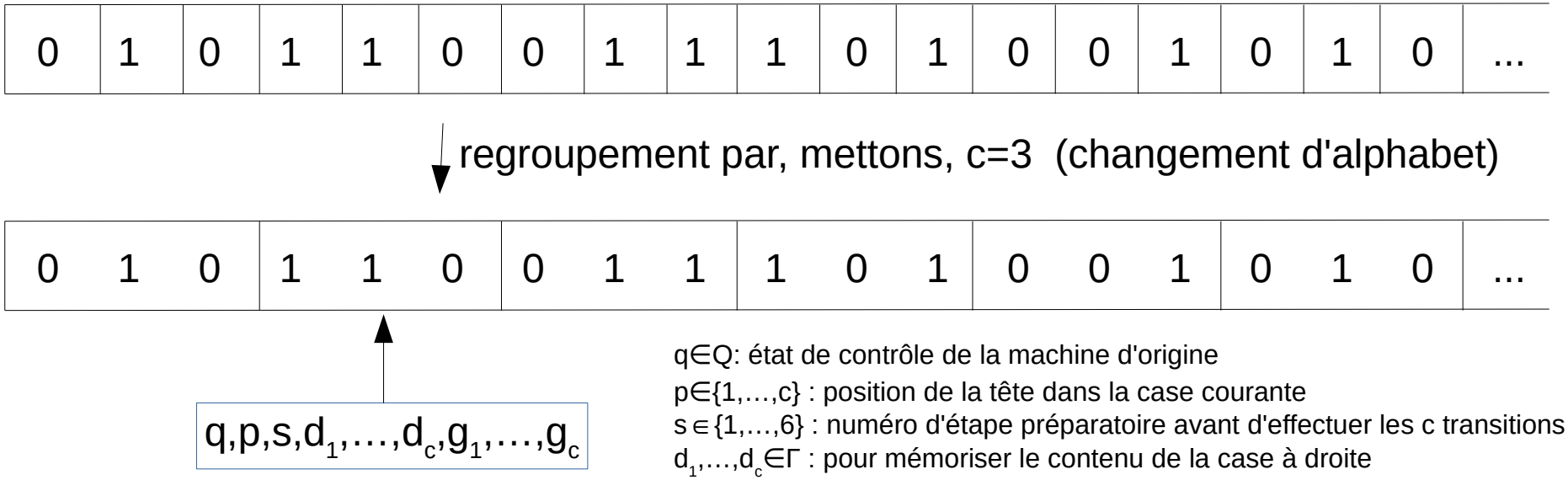
Complexité en temps: peu importe le modèle d'exécution

- n: taille de l'entrée
- Si M est une MT à k bandes et k têtes fonctionnant en temps t(n), on peut simuler M par une MT M' à une seule bande et fonctionnant en temps $O(kt(n)^2)$
  - on stocke la case i du ruban j sur la case ik+j de M'
  - un symbole spécial est utilisé pour localiser les têtes des k rubans
  - pour chaque transition de M, il faut
    - pour chacune des k bandes (total=O(kt(n)))
      - chercher le symbole sous la tête: t(n)
      - revenir au début de la bande : t(n)
    - pour chacune des k bandes (total=0(k(t(n)))
      - simuler le mouvement de la tête sur la bande: t(n)
      - revenir au début: t(n)
  - au total, les t(n) opérations nécessaires au calcul nécessitent un temps t(n)O(kt(n))
- Si M est une machine RAM fonctionnant en temps t(n), on peut la simuler par une MT fonctionnant en temps p(t(n)), où p est un polynôme
  - dans la construction machine RAM → MT, chaque instruction se simule en temps polynomial
- Dans la suite, le modèle de base de MT utilisé pour les complexités en temps est celui à k bandes bi-infinies et k têtes

DTIME(T(n))

- T:N→N
- DTIME(T(n)) : ensemble des langages reconnus par une MT à k>1 bandes et k têtes en temps ≤ T(n) sur les entrées de taille n
- Le temps mis pour lire l'entrée (qu'on suppose toujours lue en entier) compte
- si T≤G, on a DTIME(T(n)) ⊆ DTIME(G(n))
- Théorème (d'accélération linéaire en temps):
  Si L est accepté par une MT à k bandes et k têtes en temps T(n), alors il est accepté par une MT à k bandes et k têtes en temps zT(n) quel que soit z > 0, à condition que k > 1 et $\inf_{n\to\infty} T(n)/n = \infty$
- Conséquence :
  Si $\inf_{n\to\infty} T(n)/n = \infty$ et z > 0, alors DTIME(T(n)) = DTIME(zT(n))
- Dit autrement, si une MT M' s'autorise z fois plus de temps que M, elle ne reconnaîtra pas plus de langages que M !

Théorème d'accélération linéaire en temps



$q \in Q$: état de contrôle de la machine d'origine
$p \in \{1,\dots,c\}$: position de la tête dans la case courante
$s \in \{1,\dots,6\}$: numéro d'étape préparatoire avant d'effectuer les c transitions
$d_1,\dots,d_c \in \Gamma$: pour mémoriser le contenu de la case à droite
$g_1,\dots,g_c \in \Gamma$: pour mémoriser le contenu de la case à gauche

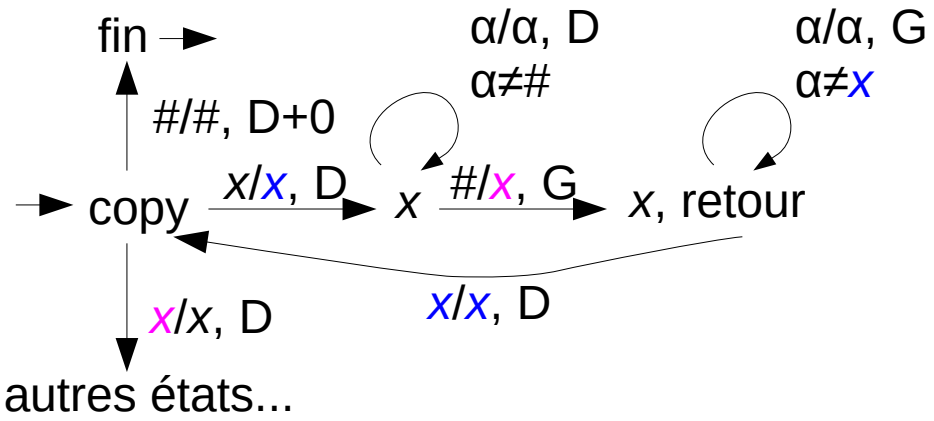
Étapes préparatoires (mémorisées dans s):

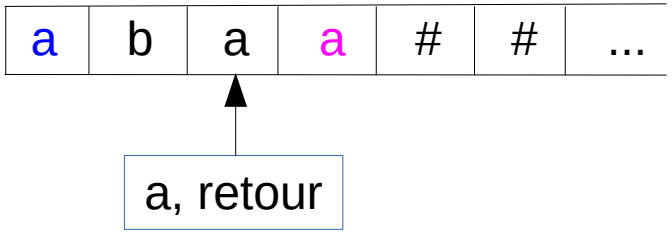
1) aller sur la case de droite, lire son contenu et le mémoriser dans $d_1,\dots,d_c$
2) aller à gauche
3) aller à gauche, lire son contenu et le mémoriser le contenu de la cellule dans $g_1,\dots,g_c$
4) aller à droite (donc, on revient sur la cellule de départ)
5) exécuter les c transitions (peut prendre 2 étapes s'il est nécessaire soit de modifier la case à droite, soit de modifier la case à gauche)

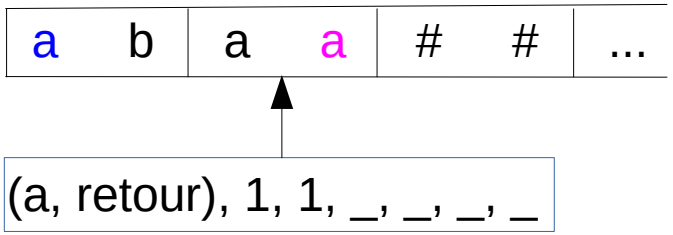
Exemple

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#

x∈A
x,x∉A nouveaux symboles associés à x



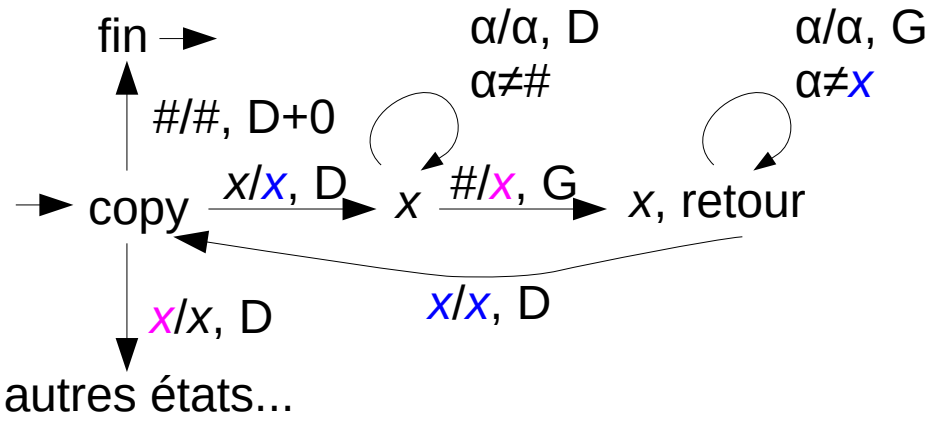


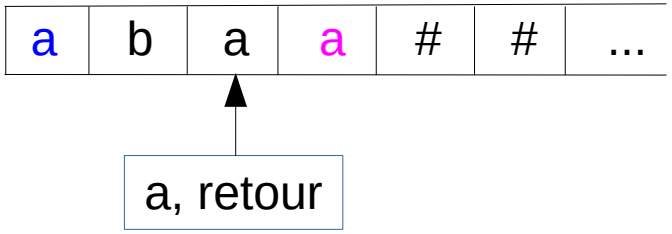


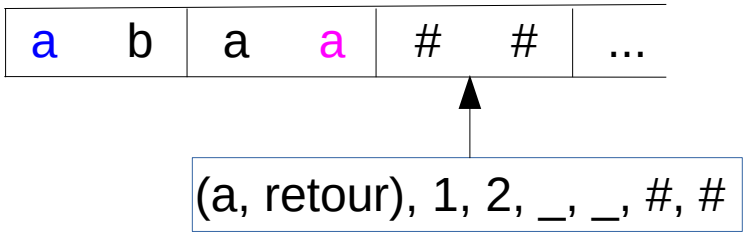
Exemple

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#

x∈A
x,x∉A nouveaux symboles associés à x



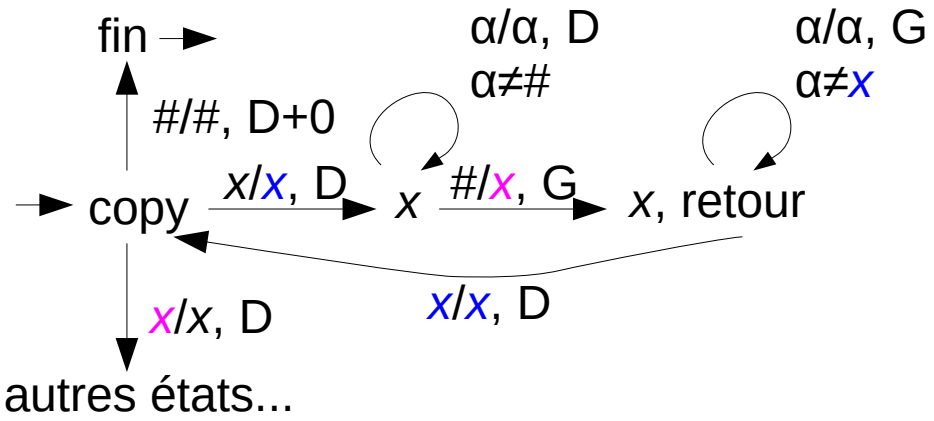


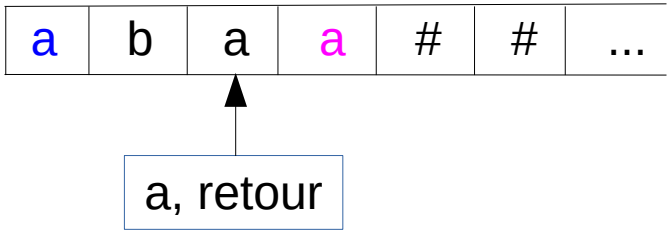


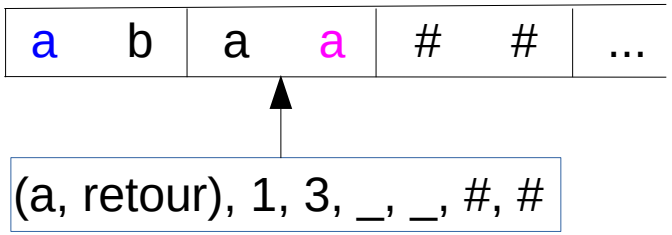
Exemple

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#

x∈A
x,x∉A nouveaux symboles associés à x



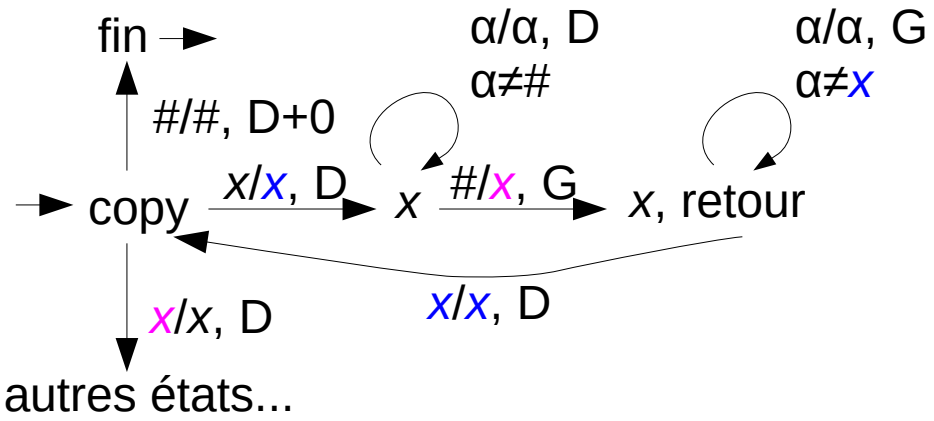


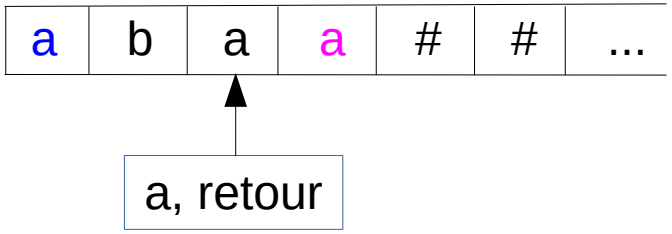


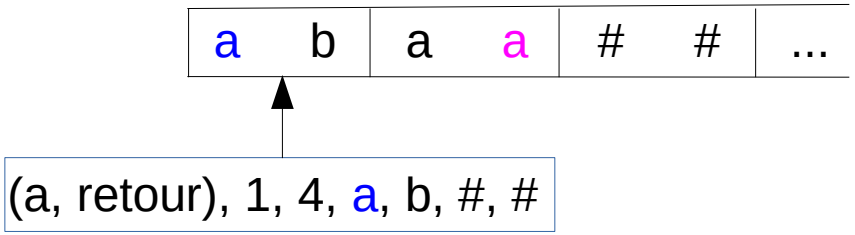
Exemple

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#

x∈A
x,x∉A nouveaux symboles associés à x



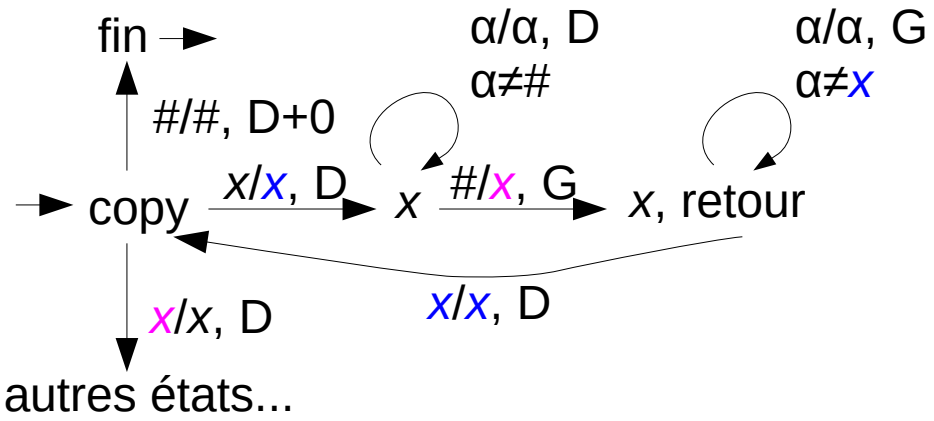


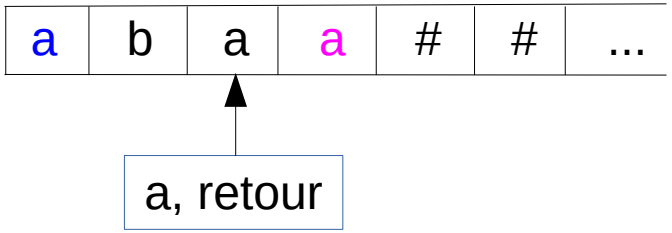


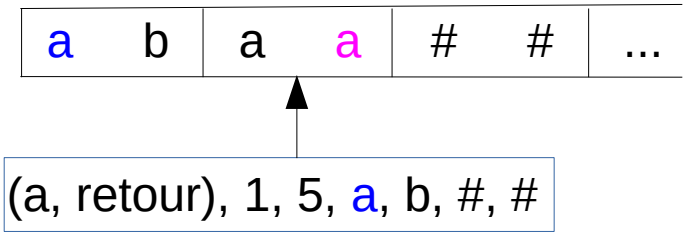
Exemple

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#

x∈A
x,x∉A nouveaux symboles associés à x



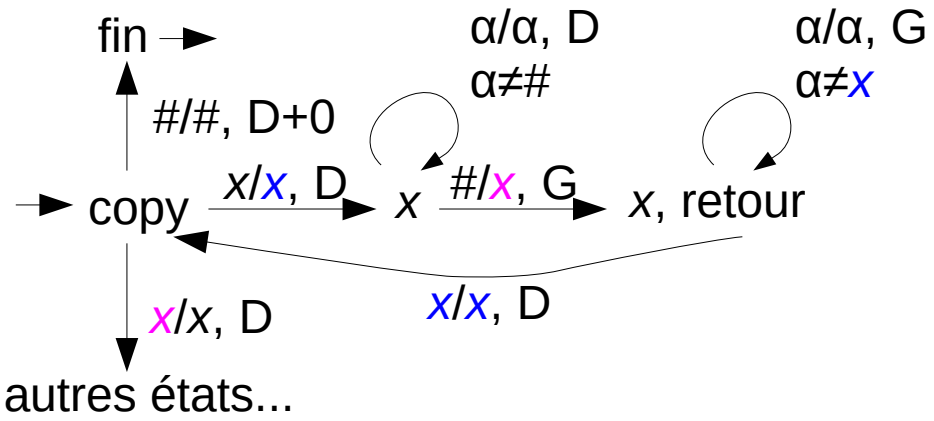


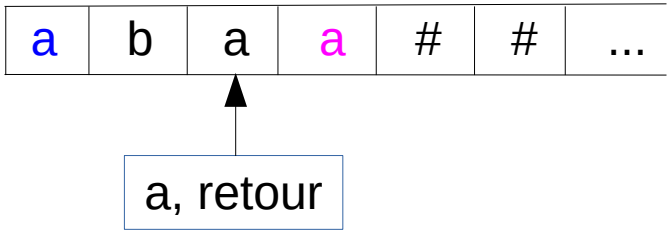


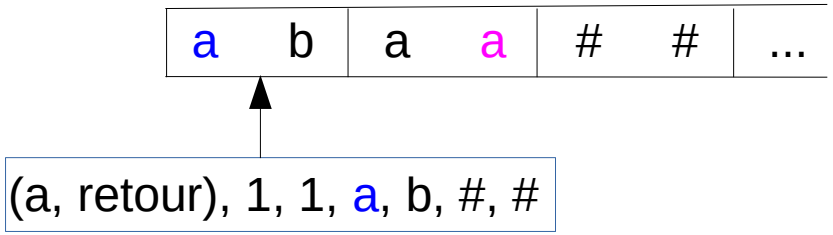
Exemple

copy: copie le mot sous la tête en fin de bande
Entrée: ucopyv#
Sortie: ufinvv#

x∈A
x,x∉A nouveaux symboles associés à x







Remarques

Sur la construction

- il faut au plus 6 mouvements pour en simuler c, car il peut être nécessaire de modifier la case de droite ou celle de gauche
- pour obtenir le théorème, il faut:
  - commencer par construire la bande condensée,
    - n mouvements pour construire la partie condensée
    - $\lceil n/c \rceil$ pour revenir au début de la partie condensée
  - au total, il faut $n+\lceil n/c \rceil+6\lceil T(n)/c \rceil$ mouvements pour réaliser la simulation sur une entrée de taille n
  - comme $\lceil x \rceil \leq x+1$, on a $n+\lceil n/c \rceil+6\lceil T(n)/c \rceil \leq n+n/c+6T(n)/c+7$
    $=T(n)[n/T(n)+n/(cT(n))+6/c+7/T(n)]$
  - en supposant $\inf_{n\to\infty} T(n)/n = \infty$, pour tout y il existe x tel que $T(n)/n \geq y$ pour tout $n \geq x$
    il vient donc, quand $n \geq x$, $T(n)[n/T(n)+n/(cT(n))+6/c+7/T(n)] \leq T(n)[1/y+1/(yc)+6/c+7/y]$
- il suffit maintenant, pour avoir le théorème d'accélération en temps,
  - de choisir y et c pour que $z \geq [1/y+1/(yc)+6/c+7/y]$ pour tout $n \geq x$
  - il faut aussi construire la machine pour qu'elle traite les cas $n < x$ (qui sont en nombre fini) en dur
- pour avoir le corollaire, il suffit d'appliquer le théorème quand $k > 1$. Si la MT n'a qu'une bande on peut toujours la simuler par une MT à 2 bandes

Théorème de hiérarchie pour DTIME

- Rappel : si une MT M' s'autorise z fois plus de temps que M, elle ne reconnaîtra pas plus de langages que M !
- Mais si z est une fonction et non une constante ?
- Théorème de hiérarchie :
  Pour toute fonction récursive T(n), il existe un langage récursif $L \notin DTIME(T(n))$

Théorème de hiérarchie pour DTIME

Les MT à k bandes et k têtes sont énumérables

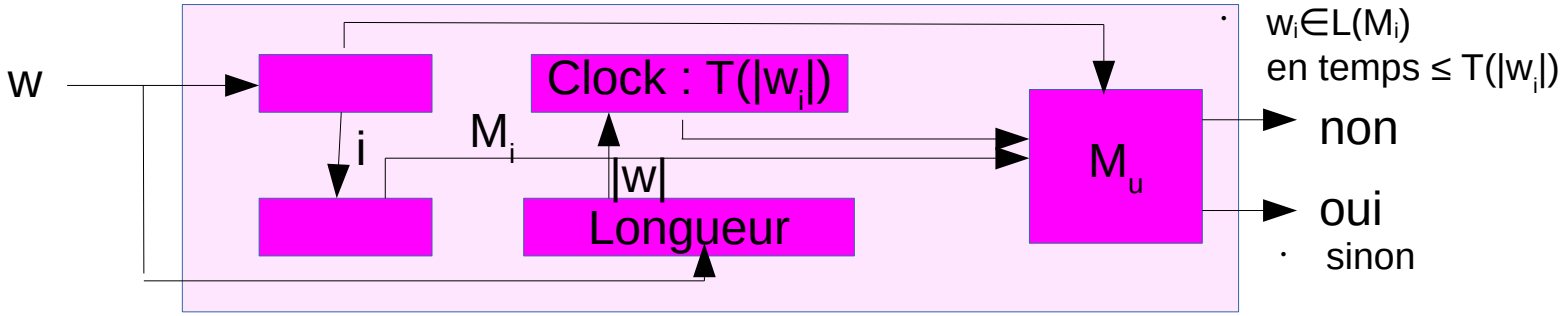
$b_{i,j}$ vaut vrai ssi $w_i \in L(M_j)$

T(n) une fonction récursive

Soit L = { w_i : w_i non reconnu par M_i en temps ≤ T($|w_i|$) }

	M_0	M_1	M_2	...	M_i	...
w_0	$b_{0,0}$	$b_{0,1}$	$b_{0,2}$			
w_1	$b_{1,0}$	$b_{1,1}$	$b_{1,2}$			
w_2	$b_{2,0}$	$b_{2,1}$	$b_{2,2}$			
...						
w_j					$b_{j,i}$	
...						

L est récursif:



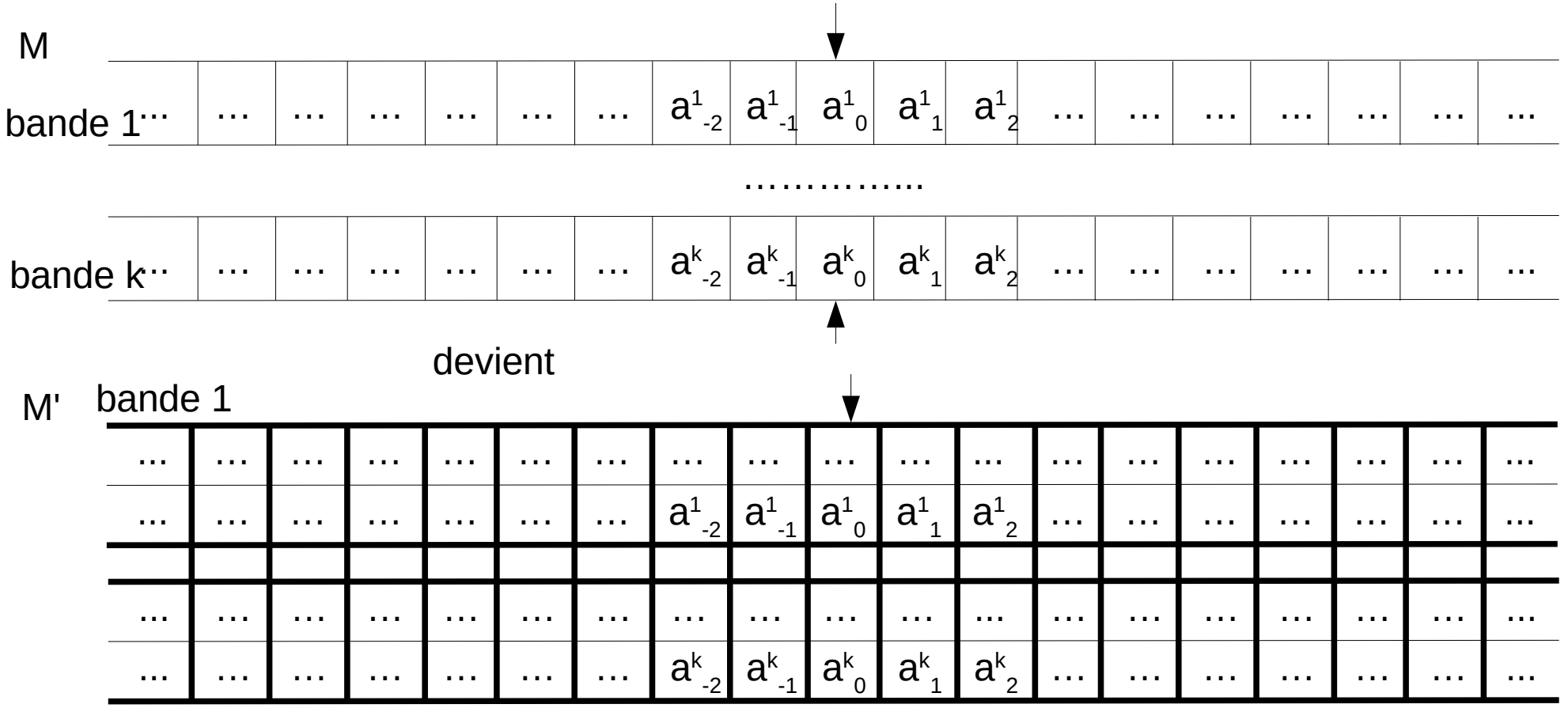
L n'est reconnu par aucune MT M_i en temps ≤ T(n): par l'absurde, supposons qu'il le soit

si $w_i \in L$, alors w_i non reconnu par M_i en temps ≤ T($|w_i|$) : contradiction

si $w_i \notin L$, alors comme $L = L(M_i)$, w_i non reconnu par M_i en temps ≤ T($|w_i|$) : contradiction

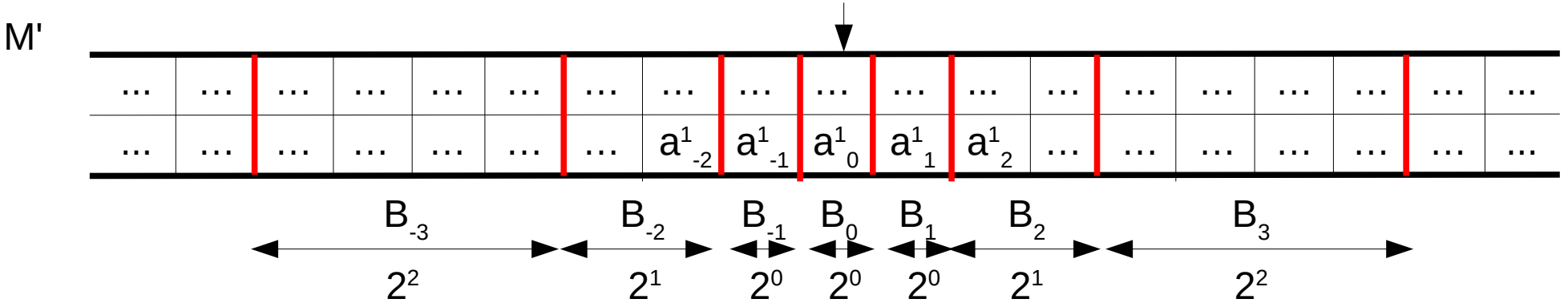
Réduction du nombre de bandes

- Si L est le langage d'une MT M à k bandes et k têtes et accepté en temps T(n), alors L est le langage d'une MT M' à 2 bandes, qui l'accepte en T(n)logT(n)
  - La première bande de M' contient 2 pistes pour chacune des k bandes de M
    - La première piste est vide
    - La seconde piste contient initialement la bande de M correspondante
  - La seconde est une bande de travail



Réduction du nombre de bandes

Focus sur la piste de M' contenant une bande particulière (par exemple la première) de M



La piste est divisée en blocs B_i de tailles 2^{i-1} (sauf la taille de B_0 qui est 1)

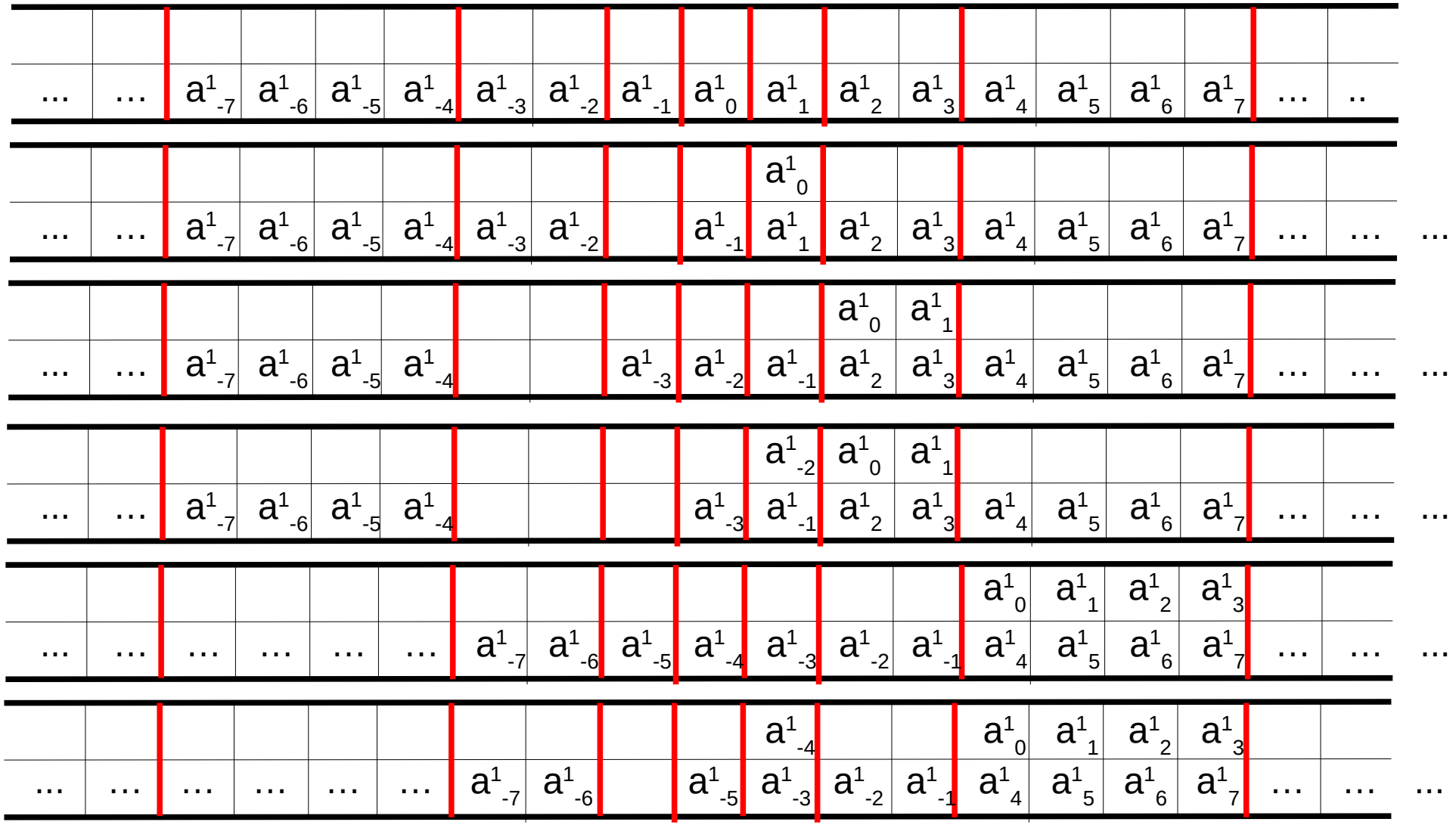
La partie supérieure de B_0 est toujours vide

On s'arrange pour que le caractère sous la tête i de M se trouve toujours dans B_0

On ré-arrange pour cela les B_i

Réduction du nombre de bandes
Ex: 5 mouvements de tête à gauche

M'



Réduction du nombre de bandes Calcul de complexité

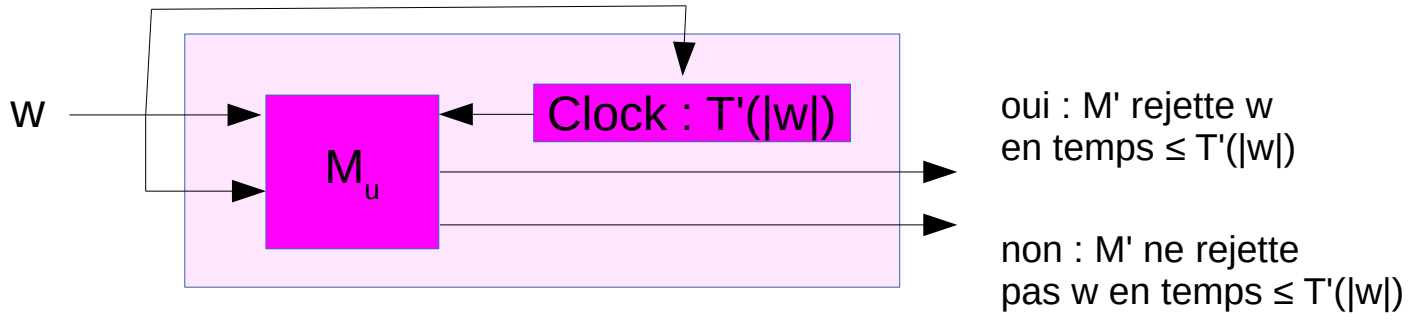
- B_i-opération: ré-arrangement des blocs B_i et B_{-i}
- B_1-opération: tous les mouvements
- B_2-opération: tous les 2 mouvements
- B_3-opération: tous les 4 mouvements
- …
- B_k-opération: tous les 2^{k-1} mouvements
- chaque B_i-opération a un coût de $c2^i$ mouvements
- la première B_i-opération n'est pas faite avant le 2^{i-1} mouvement de M
- Si M fait T(n) mouvements, M' en fait donc $\Sigma_{i=1}^{\log T(n)} c2^i T(n)/2^{i-1} < 4cT(n)\log_2 T(n)$
- Par application du Théorème d'accélération linéaire en temps, M' est équivalente à M'' qui réalise $T(n)\log_2 T(n)$ mouvements quand M en fait T(n)

La hiérarchie pour DTIME est donc infinie, mais avec quelle granularité ?

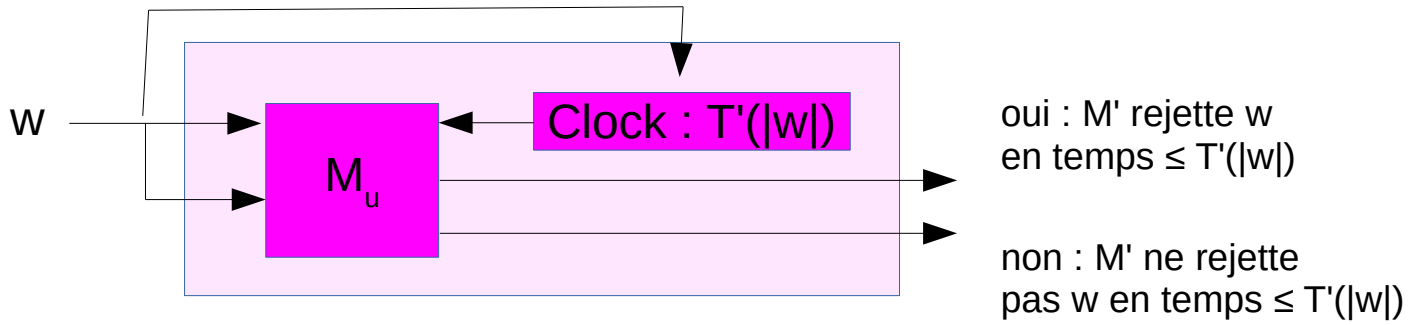
- Fonction
  - **constructible en temps**: il existe une MT M telle que pour chaque entier n il existe une entrée w telle que M(w) s'arrête en T(n) mouvements
  - **entièrement constructible en temps**: M s'arrête en T(n) mouvements sur toute entrée de taille n
- Si T'(n) est entièrement constructible en temps et $\inf_{n\to\infty} (T(n)\log T(n))/T'(n)=0$, alors il existe L tel que $L\in DTIME(T'(n))$ et $L\notin DTIME(T(n))$

Granularité du théorème de hiérarchie pour DTIME

- Soit M=(Q,T,I,B,next,i,F) une MT à x bandes et x têtes
  - supposons wlog. $Q=\{q_1,\ldots,q_n\}$, $T=\{0,1\}$, $I=\{0,1,B\}$, $i=q_1$, $F=q_2$
  - posons $D_1=D$, $D_2=G$, $0=S_1$, $1=S_2$, $B=S_3$
  - Codage d'une transition $t=next(q_i,s_{j1},\ldots,s_{jx})=(q_k,s_{l1},\ldots,s_{lx},D_{m1,\ldots},D_{mx})$: $0^i10^{j1}\ldots10^{jx}010^k10^{l1}1\ldots10^{lx}10^{m1}1\ldots10^{mx}$
  - Codage du contrôle fini:$1*111t_111..11t_r111$
  - La taille du codage du contrôle fini peut donc être arbitrairement grande
- Dans la MT M suivante w est le codage précédent d'une MT M' s'exécutant en temps T(n):



Granularité du théorème de hiérarchie pour DTIME



- Supposons M' en temps T(n)
- Si M a moins de bandes que M', elle peut simuler M' en temps T(n) log T(n)
- Si de plus les alphabets de bandes sont différents, M peut simuler M' en temps cT(n) log T(n)
  - Comme $\inf_{n\to\infty}$ (T(n)log T(n))/T'(n) = 0 il existe un codage w de M' suffisamment long pour que cT(|w|)log T(|w|) ≤ T'(|w|)
- On a w ∈ L(M) ssi w ∉ L(M'), donc L(M) ≠ L(M')
- Donc L(M) ∈ DTIME(T'(n))-DTIME(T(n))

Le théorème de lacune Version Borodin 1972

- (φ,Φ) est une **mesure de complexité** si
  - φ énumère toutes les fonctions récursives partielles (donc, toutes les machines de Turing)
  - $\varphi_i(n)$ est défini exactement quand $\Phi_i(n)$ l'est
    - $\varphi_i(n)$ est la valeur de la $i^{ème}$ MT sur l'entrée n
    - $\Phi_i(n)$ est la mesure de complexité du calcul de $\varphi_i(n)$ [on prend $\Phi_i(n)=\infty$ quand $\varphi_i(n)$ n'est pas définie]
  - $\Phi_i(n) = m$ est une fonction calculable
- Théorème de lacune (version Borodin)
  Soient
  - (φ,Φ) une mesure de complexité et
  - g une fonction récursive totale non décroissante vérifiant $g(x) \geq x$.

  Alors il existe une fonction récursive totale non décroissante f telle que, pour n suffisamment grand,
  - $\Phi_i(n) \leq f(n)$ ou
  - $\Phi_i(n) > g \circ f(n)$
- Il suffit de prendre
  - $f(0) = 1$
  - $f(n+1) = k$ avec k vérifiant
    - $k \geq f(n)$
    - pour tout $i \leq n$, $\Phi_i(n) \leq k$ ou $g(k) < \Phi_i(n)$

Le théorème de lacune Version Borodin 1972

- Théorème de lacune (version Borodin)
  Soient
  - (φ,Φ) une mesure de complexité et
  - g une fonction récursive totale non décroissante vérifiant $g(x) \geq x$.

  Alors il existe une fonction récursive totale non décroissante f telle que, pour n suffisamment grand,
  - $\Phi_i(n) \leq f(n)$ ou
  - $\Phi_i(n) > g \circ f(n)$
- Il suffit de prendre
  - f(0)=1
  - f(n+1)=k avec k vérifiant
    - $k \geq f(n)$
    - pour tout $i \leq n$, $\Phi_i(n) \leq k$ ou $g(k) < \Phi_i(n)$
- f est bien définie et vérifie les conditions de l'énoncé car
  - k existe pour tout n, puisque pour chaque $i<n$, ou bien $\Phi_i(n)$ est défini, ou bien il ne l'est pas et on peut toujours prendre un k tel que $k \geq f(n)$ et g(k) fini (puisque g est récursive)
  - k (pour $n>0$) se trouve par une MT M (donc f est récursive), car (φ,Φ) étant une mesure de complexité et g étant récursive, les prédicats $\Phi_i(n) \leq k$ et $\Phi_i(n) > g(k)$ sont décidables. M réalise l'algorithme suivant
    - pour chaque valeur de $k \geq f(n-1)$, en les examinant de 1 en 1 et par ordre croissant
      - si pour chaque $i<n$, $\Phi_i(n) \leq k$ ou $\Phi_i(n) > g(k)$ alors on arrête le calcul, la valeur de k est trouvée
      - sinon on passe à la valeur de k suivante
    - comme on sait que k existe, la boucle s'arrête

Le théorème de lacune
gap theorem (Trakhtenbrot 64, Borodin 72)

- Il a d'abord été découvert par Trakhtenbrot en 1964, publié en russe, puis redécouvert par Borodin
- Dans la granularité du théorème de hiérarchie, il nécessaire que T'(n) soit entièrement constructible en temps
- Application à DTIME (avec $g = 2^n$)
  Il existe une fonction $f(n) \geq n$ telle que $DTIME(f(n)) = DTIME(2^{f(n)})$
- En d'autres termes, la hiérarchie DTIME s'écrase sans la notion d'"entièrement constructible"

Théorème *speed-up* (Blum)

- Version générale
  Soient
  - (φ,Φ) une mesure de complexité et
  - g une fonction récursive totale à deux paramètres

  Alors il existe une fonction récursive totale f telle que, pour chaque MT M_i calculant f, il existe M_j calculant aussi f vérifiant, pour n suffisamment grand,
  - $g(n,\Phi_j(n)) \leq \Phi_i(n)$
- Version “mesure de complexité en espace”

  Soit g(n) une fonction récursive totale.
  Il existe une fonction récursive totale f telle que, pour chaque MT M_i calculant f, il existe M_j calculant aussi f vérifiant, pour n suffisamment grand,
  - $g(S_j(n)) \leq S_i(n)$

  On note $S_i(n)$ la mesure de l'espace mémoire nécessaire par M_i au calcul de f(n) par M_i.

Preuve

- On suppose *wlog*. $g(n) \geq n^2$ constructible en espace
- Soit $h(1) = 2$, $h(n) = g(h(n-1))$: h est aussi constructible en espace
- On construit L vérifiant

  1) si $L(M_i)=L$ alors $S_i(n) \geq h(n-i)$ pour n suffisamment grand

  2) pour chaque k il existe M_j tel que $L = L(M_j)$ et $S_j(n) \leq h(n-k)$

  alors, si $L=L(M_i)$, on a $S_i(n) \geq h(n-i) = g(h(n-i-1))$ et comme $h(n-i-1) \geq S_j(n)$ alors $S_i(n) \geq g(S_i(n))$

- Définition de $L \subseteq 0^*$
  - Dans l'ensemble $\{ M_0, M_1, \ldots \}$ on *annule* des machines
    - soit $\sigma(n)$ le plus petit indice $j \leq n-1$ de machine non annulée vérifiant $S_j(n) < h(n-j)$
    - quand on considère n, si $\sigma(n)$ existe, on annule $M_{\sigma(n)}$, et on pose $0^n \in L$ si $0^n \notin L(M_{\sigma(n)})$
      [donc, une machine annulée ne peut pas avoir L pour langage]
  - Alors L vérifie 1)
    - supposons $L(M_i) = L$ et $S_i(n) < h(n-i)$. Alors il existe n' tel que M_i soit annulée pendant l'examen de $0^{n'}$, donc $L(M_i) \neq L$

Théorème de l'union

- Soit { $f_i(n)$: i=1,2,3,… } une famille r.e. de fonctions r.e. Si, pour chaque i et n, $f_i(n) \leq f_{i+1}(n)$, il existe une fonction récursive S(n) vérifiant $DSPACE(S(n)) = \cup_{i \geq 1} DSPACE(f_i(n))$
- Il faut que $S(n) \geq f_i(n)$ presque partout
  - il suffirait pour cela de définir $S(n) = f_n(n)$
- Il faut aussi qu'il n'existe pas de MT M_j telle que pour tout i, $f_i(n) \leq S_j(n) \leq S(n)$ presque partout
  - $S(n) = f_n(n)$ ne permet pas de satisfaire cette condition
- Par un algorithme, on définit S(n) vérifiant

  1) Pour n suffisamment grand, $S(n) \geq f_i(n)$ pour tout i

  2) S'il existe j tel que pour tout i, $S_j(n) > f_i(n)$ infiniment souvent, alors $S_j(n) > S(n)$ infiniment souvent

  Ces deux conditions garantissent que $DSPACE(S(n)) = \cup_{i \geq 1} DSPACE(f_i(n))$

- Algorithme de calcul de S(n) :
  - il choisit $S(n) \leq$, pour une infinité de n, à chaque $S_j(n)$ qui est infiniment souvent $\geq$ chaque $f_i(n)$
  - pour cela, il maintient une liste L de couples (j,k), j indice de MT, k indice de fonction f, telle que « $f_k(n) \geq S_j(n)$ presque partout » soit une hypothèse de calcul. A chaque étape du calcul, la liste est révisée.
  - si l'algorithme se rend compte que l'hypothèse de calcul « $f_k(n) \geq S_j(n)$ presque partout » peut être fausse (parce qu'elle est fausse pour l valeur de n actuelle),
    - il la remplace par « $f_n(n) \geq S_j(n)$ presque partout »
    - il définit $S(n) = f_k(n)$, garantissant ainsi que $S(n) < S_j(n)$

Théorème de l'union

$L:=\emptyset$

Pour n=1, 2, ...

Si $f_k(n) \geq S_j(n)$ pour tout $(j,k)\in L$

$L+=(n,n)$

$S(n):=f_n(n)$

sinon

Parmi tous les $(j,k)\in L$, choisir celui avec k le plus petit possible, puis avec j le plus petit possible

$S(n):=f_k(n)$

substituer (j,k) par (j,n) dans L

$L+=(n,n)$

DSPACE(S(n))

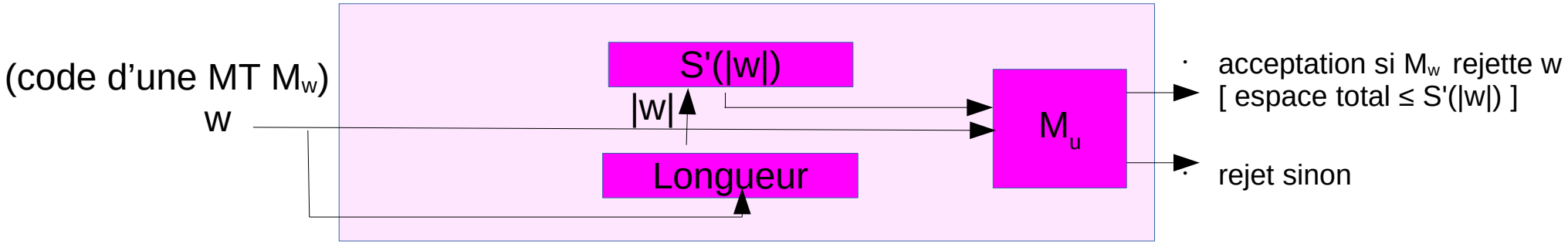
- S:N→N
- DSPACE(S(n)) : ensemble des langages reconnus par une MT à k>1 bandes et k têtes en espace ≤ S(n) sur les entrées de taille n.
- On compte uniquement l'espace pour les calculs, pas celui pour stocker l'entrée.
- Si L est accepté en espace S(n) par une MT M, alors pour tout c>0, L est aussi accepté en espace cS(n) par une MT M'
  - Il suffit de construire M' qui regroupe, sur une seule de ses cellules, plusieurs des cellules de M
- Si L est accepté en espace S(n) par une MT M à k bandes et k têtes, alors il est accepté en espace S(n) par une MT M' à une seule bande
  - les k bandes sont mises en parallèle sur une seule bande avec les techniques déjà vues, et on simule les k têtes avec une technique déjà vue
- Comme pour DTIME, on travaille dans DSPACE à constante multiplicative près

Théorème de hiérarchie pour DSPACE

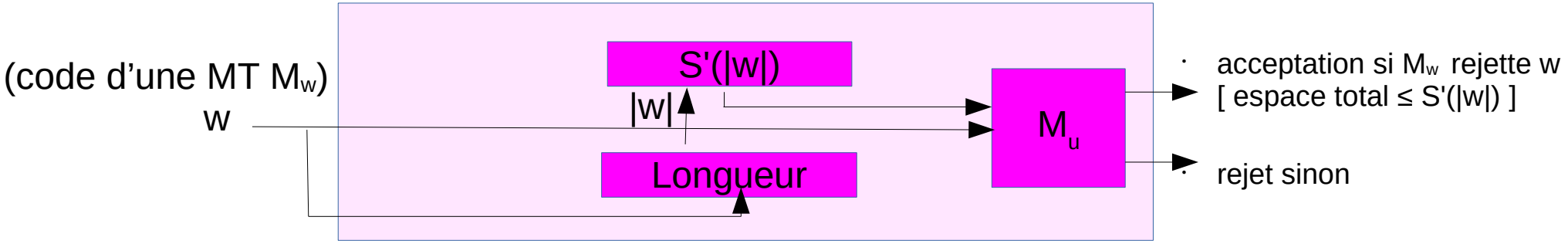
- Fonction
  - **constructible en espace**: il existe une MT M telle que pour chaque entier n il existe une entrée w telle que M(w) utilise S(n) cellules
  - **entièrement constructible en espace**: M utilise S(n) cellules sur toute entrée de taille n
- Si S'(n) est entièrement constructible en espace et $\inf_{n\to\infty} S(n)/S'(n)=0$, alors il existe L tel que $L\in DSPACE(S'(n))$ et $L\notin DSPACE(S(n))$
- Dans le résultat précédent, l'hypothèse « S'(n) entièrement constructible » peut être remplacée par « S'(n) constructible »

Théorème de hiérarchie pour DSPACE

- Soit M=(Q,T,I,B,next,i,F) une MT à x bandes et x têtes
  - supposons wlog. $Q=\{q_1,\ldots,q_n\}$, $T=\{0,1\}$, $I=\{0,1,B\}$, $i=q_1$, $F=q_2$
  - posons $D_1=D$, $D_2=G$, $0=S_1$, $1=S_2$, $B=S_3$
  - Codage d'une transition $t=next(q_i,s_{j1},\ldots,s_{jx})=(q_k,s_{l1},\ldots,s_{lx},D_{m1},\ldots,D_{mx})$: $0^i10^{j1}\ldots10^{jx}010^k10^{l1}1\ldots10^{lx}10^{m1}1\ldots10^{mx}$
  - Codage du contrôle fini:$1*111t_111..11t_r111$
  - La taille du codage du contrôle fini peut donc être arbitrairement grande
- Dans la MT M suivante w est le codage précédent d'une MT M_w



Théorème de hiérarchie pour DSPACE



M commence par marquer sa bande sur une longueur de S'(|w|)
pour définir l'espace d'exécution autorisé
On suppose que M_u décide du problème de l'arrêt dans l'espace autorisé
M simule M_w dans l'espace autorisé
On a $L(M) \in DSPACE(S'(n))$
On montre $L(M) \notin DSPACE(S(n))$
Par l'absurde, supposons $L(M) \in DSPACE(S(n))$. Il existe w tel que $L(M_w) = L(M)$
Comme il existe une infinité de code pour M_w et que $\inf_{n\to\infty} S(n)/S'(n) = 0$
on peut supposer |w| suffisamment grande pour que $S(|w|) < S'(|w|)$
On a alors w accepté par M (espace total $\leq S'(|w|)$) ssi w est rejeté par M_w : contradiction

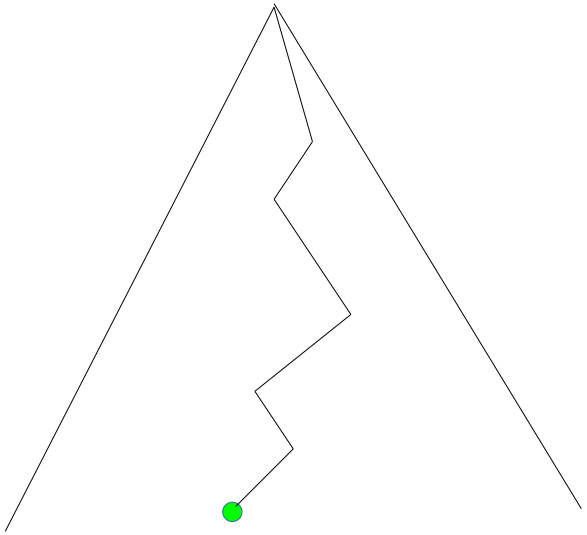
Espace connu et entièrement constructible → arrêt décidable

- Si L est accepté par une MT M en espace S(n) entièrement constructible alors il l'est par une MT M' en espace S(n) qui s'arrête sur toute entrée
- Supposons que M aie s états et un alphabet de taille t
- Si M accepte un mot de longueur n, alors c'est en au plus $sS(n)t^{S(n)}$ mouvements
  - sinon on répète deux fois la même configuration (bande, état, position de tête)
- On a $(2st)^{S(n)} \geq sS(n)t^{S(n)}$
- M’ se réserve une bande supplémentaire de taille S(n) avec un alphabet de taille 2st pour compter le nombre de mouvements en base 2st réalisés lors de la simulation de l’exécution de M sur son entrée de taille n
- M' s'arrête quand le compteur dépasse la borne $sS(n)t^{S(n)}$

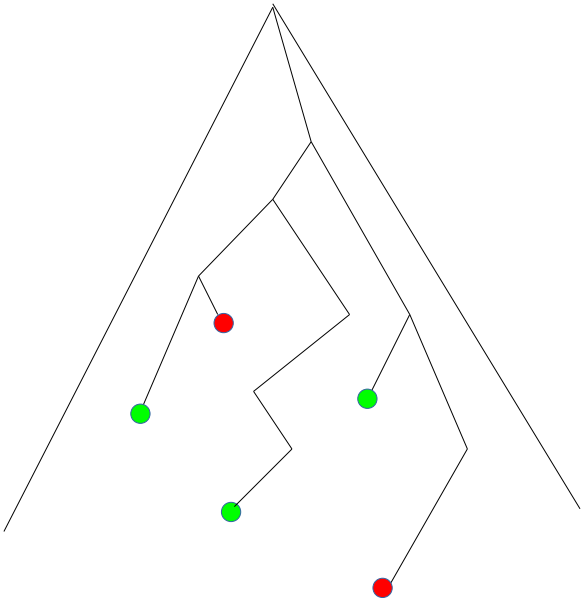
MT non déterministes

- La fonction de transition est remplacée par une relation

MT déterministe
Pour tout mot w, une seule exécution possible dans l'arbre de toutes les exécutions
Acceptation: cette exécution s'arrête sur acceptation
Rejet: cette exécution s'arrête sur rejet



MT non déterministe:
Pour tout mot w, plusieurs exécutions possibles, dont certaines peuvent accepter et d'autres non
Acceptation: il existe une exécution acceptante
Rejet: toute exécution rejette



Rappel: on suppose que toute MT s'arrête

MT non déterministes: Temps de calcul & espace mémoire

- La fonction de transition est remplacée par une relation

MT déterministe

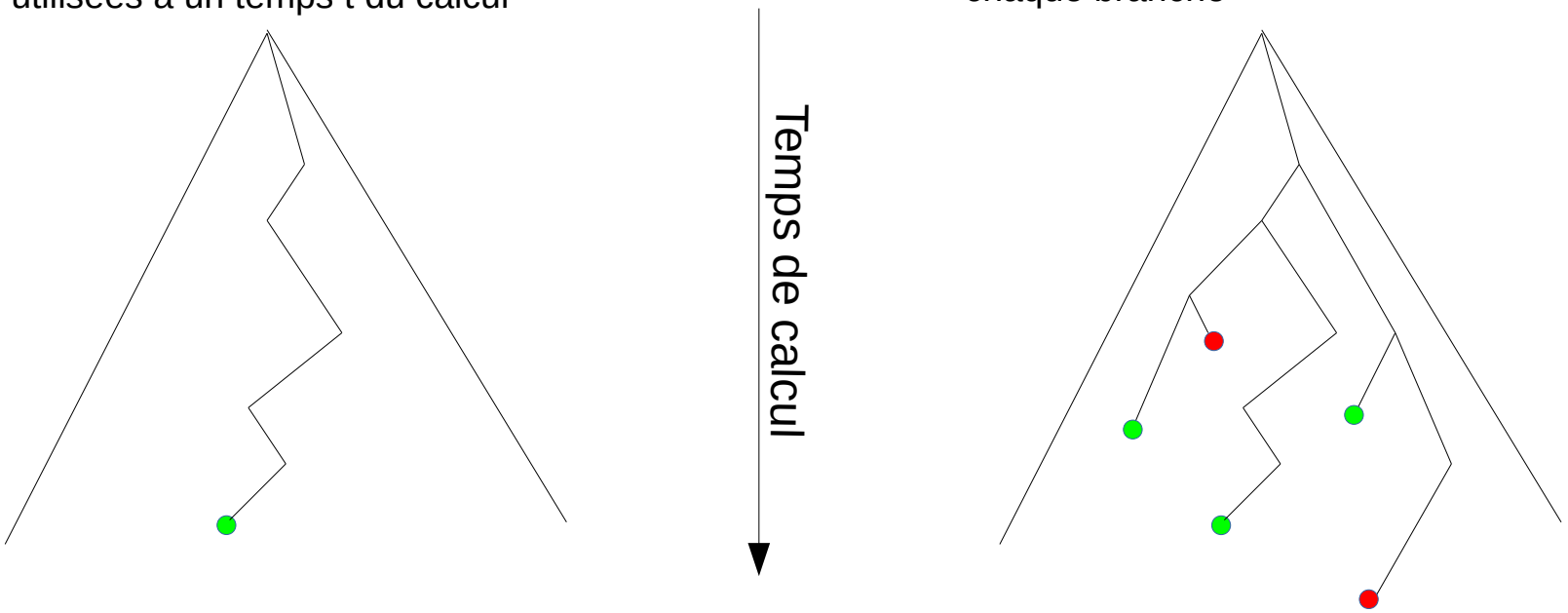
Temps de calcul: nombre d'étapes avant que le calcul ne s'arrête

Espace mémoire: plus grand nombre de cellules de la bande utilisées à un temps t du calcul

MT non déterministe:

Temps de calcul: max des temps de calcul le long de chaque branche

Espace mémoire: max des espaces utilisés le long de chaque branche



Rappel: on suppose que toute MT s'arrête

Machine universelle non déterministe

- Les MT M non déterministes peuvent se simuler par une MT universelle M_{ud} déterministe
  - coût en temps exponentiel, car M_{ud} doit explorer tous les choix faits par M
- Elles peuvent aussi se simuler par une MT universelle M_{un} non déterministe
  - M_{un} peut accepter ssi M accepte: il suffit que M_{un} passe dans un état d'acceptation ssi M le fait
  - M_{un} peut rejeter ssi M accepte, mais ça n'est pas simple !
    - il ne suffit pas que M_{un} passe
      - dans un état d'acceptation quand M passe dans un état de rejet
      - dans un état de rejet quand M passe dans un état d'acceptation
    - pour prendre la négation on ne sait pas faire autrement qu'examiner **tous** les chemins
  - Coût en temps: $O(k_M * \text{nombre d'étapes de M})$ avec k_M constante dépendant de M

Mesures de complexité pour les MTND

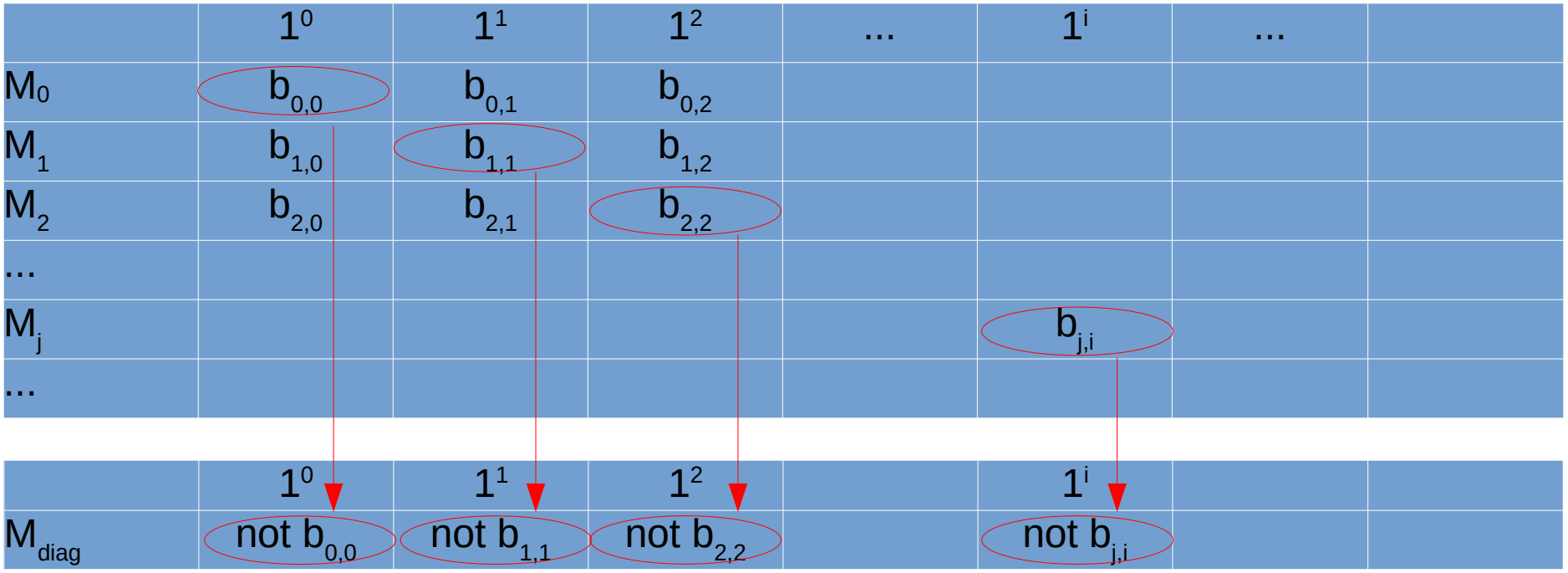
- NTIME(T(n)):
  langages des MTND utilisant un temps majoré par T(n) sur une entrée de taille n, sur toutes les branches de calcul
  - NTIME(T(n)) est clôt par union, intersection
  - la clôture de NTIME(T(n)) par complément est un problème ouvert
- NSPACE(S(n)):
  langages des MTND utilisant un espace majoré par T(n) sur une entrée de taille n, sur toutes les branches de calcul

Théorème de hiérarchie pour NTIME

- Soient f,g :N→N avec g croissante et constructible en temps, et f(n+1) ∈ o(g(n)). Alors NTIME(f(n)) ⊊ NTIME(g(n)).
- La preuve du théorème analogue pour DTIME utilise l'argument de diagonalisation, donc la possibilité pour une MT de répondre l'opposé d'une autre MT de manière cohérente et efficacement
  - ne fonctionne plus avec des MT non déterministes, car, à cause de la condition d'acceptation des MTND, pour répondre l'opposé d'une MTND on ne sait pas faire autrement qu'explorer toutes les branches de calcul

Rappel: f(n) ∈ o(g(n)) ssi pour tout k>0 on a f(n) ≤ kg(n) pour tous les n suffisamment grands

Diagonalisation en temps déterministe



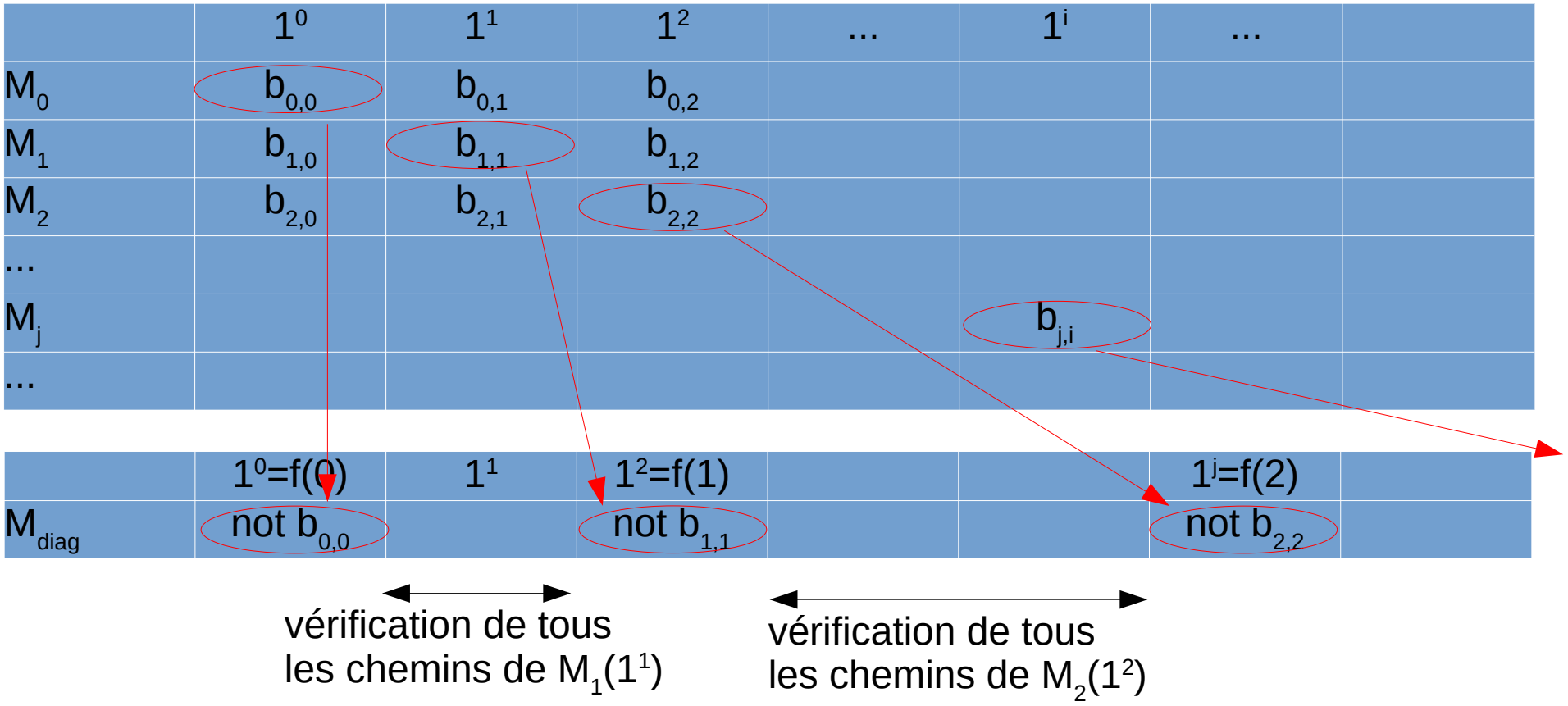
On construit une MT M' qui simule les M_i déterministes sur 1^i et retourne l'opposé

Comme prendre l'opposé se fait en temps constant, le temps mis par M' pour analyser M_i celui de M

Comme on ne sait pas si on peut prendre le complémentaire en temps suffisamment court sur des MTND, il faut trouver une autre solution pour diagonaliser

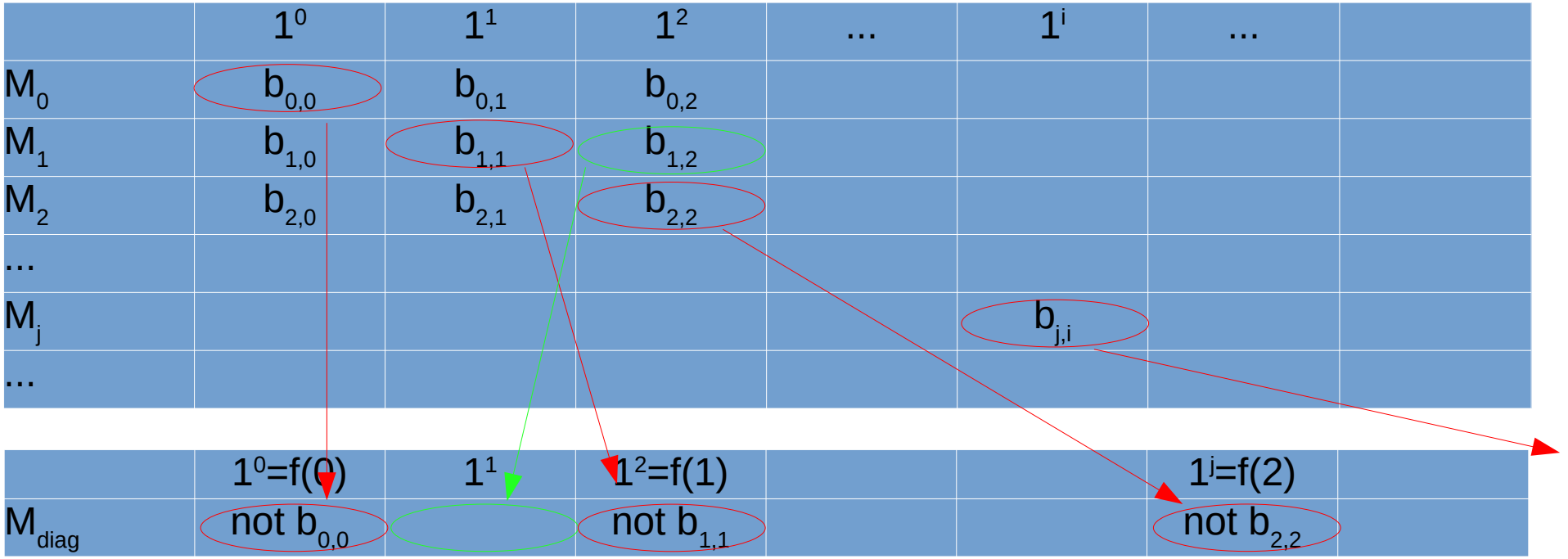
Diagonalisation retardée: justification du retard

En supposant que f(n) croisse suffisamment vite pour que f(n)-f(n-1) soit inférieur au temps de vérification de tous les chemins d'exécution de $M_n(1^n)$



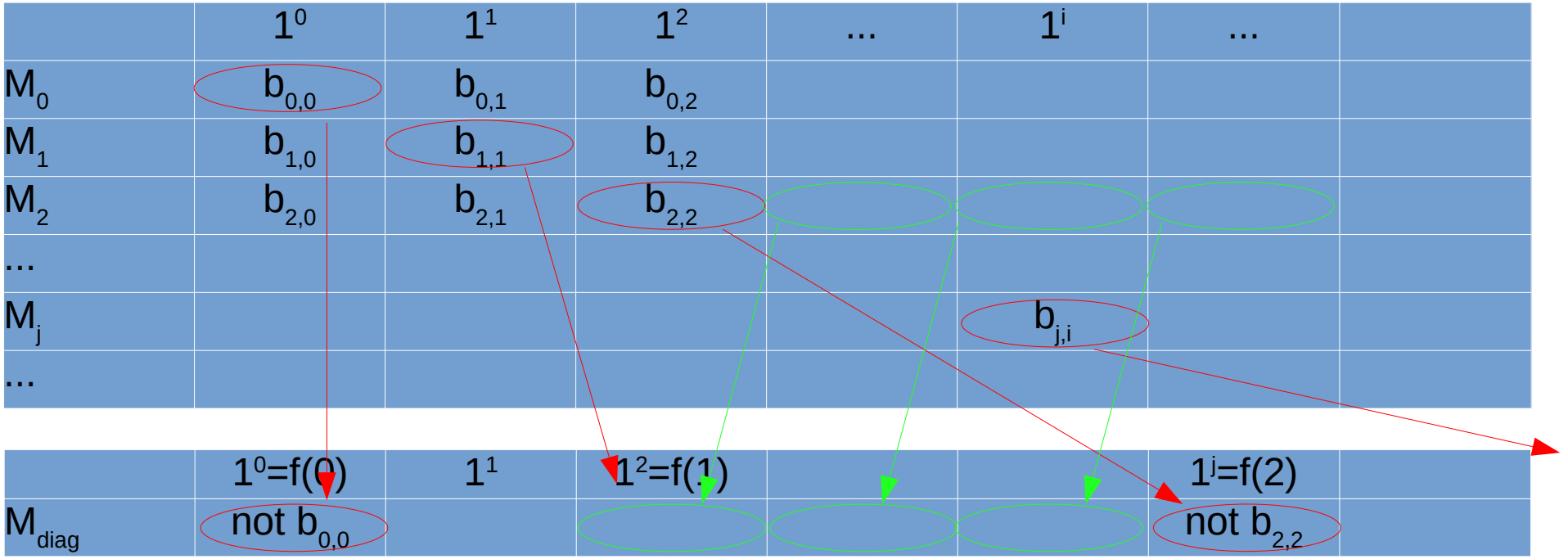
Diagonalisation retardée: obtention de la contradiction

On utilise les espaces vides de M_{diag} ! Calcul de $M_{diag}(1^2)$



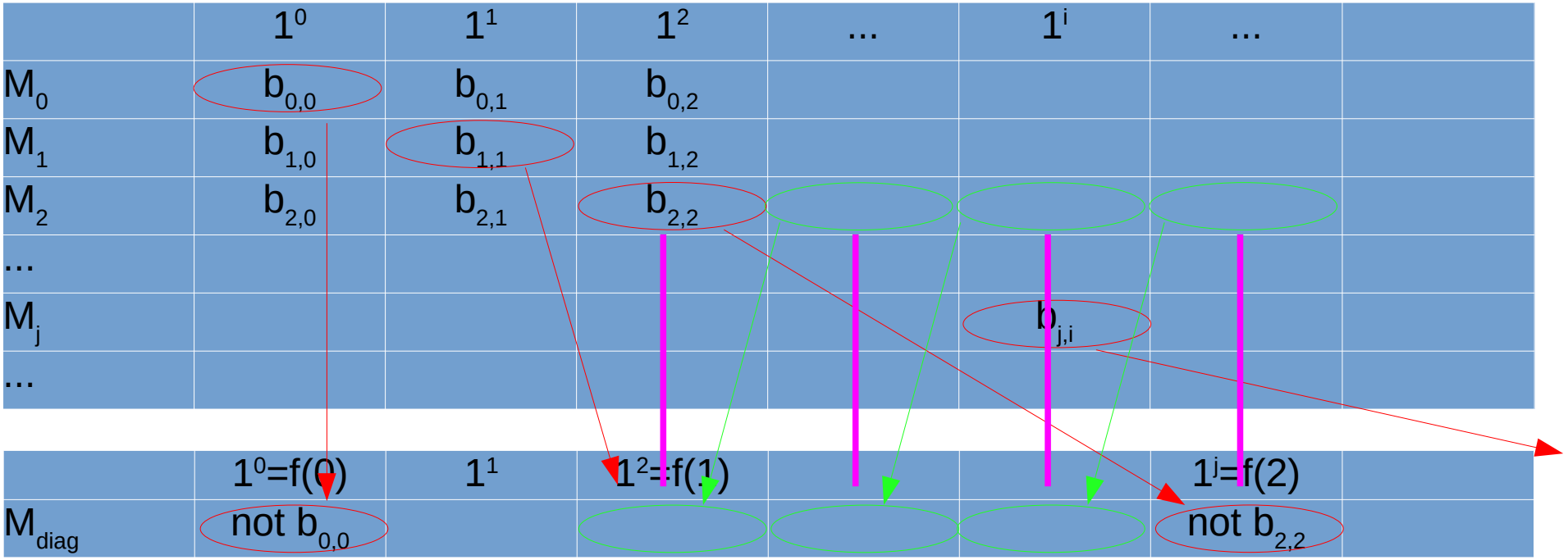
Diagonalisation retardée: obtention de la contradiction

On utilise les espaces vides de M_{diag} ! Calcul de $M_{diag}(1^{f(2)})$



Diagonalisation retardée: obtention de la contradiction

On utilise les espaces vides de M_{diag} ! Calcul de $M_{diag}(1^{f(2)})$



Si M_{diag} est M_2, alors les lignes M_2 et M_{diag} sont égales: contradiction !

Théorème de hiérarchie pour NTIME

- Soient f,g :N→N avec g croissante et constructible en temps, et f(n+1)∈o(g(n)). Alors NTIME(f(n)) ⊊ NTIME(g(n)).
- On construit L ∈ NTIME(g(n)) - NTIME(f(n)) par une MTND M
  - L est constitué de couples (N,i) avec
    - N une MTND codée par la méthode déjà utilisée, en supposant qu'on puisse préfixer par autant de 1 qu'on souhaite (donc, il existe un code pour N aussi long qu'on souhaite)
    - i un entier codé en unaire
- Sur l'entrée (N,i), le comportement de M est
  - si $i > 2^{g(|(N,0)|)}$, on simule N sur l'entrée (N,0), de manière déterministe pendant $2^{g(|(N,0)|)}$ étapes pour pouvoir calculer le complémentaire de sa sortie
    - ici on suppose pour simplifier qu'il y a au plus deux choix possibles en sortant de chaque transition
    - le temps déterministe $2^{g(|(N,0)|)}$ est donc suffisant pour explorer toutes les exécutions possibles de N sur (N,0)
    - comme $|(N,i)| > i > 2^{g(|(N,0)|)}$ le temps d'exécution de M dans ce cas est O(n) avec n taille de l'entrée
  - sinon, on simule N sur l'entrée (N,i+1), de manière non déterministe pendant un temps g(|(N,i)|)
    - La simulation non déterministe prend un temps O(g(|(N,i)|))
- Le temps de calcul de M est en O(g(n)) avec n taille de l'entrée, donc L ∈ NTIME(g(n)).

Théorème de hiérarchie pour NTIME

- Sur l'entrée (N,i), le comportement de M est
  - si $i > 2^{g(|(N,0)|)}$, on simule N sur l'entrée (N,0), de manière déterministe pendant $2^{g(|(N,0)|)}$ étapes pour pouvoir calculer le complémentaire de sa sortie
    - ici on suppose pour simplifier qu'il y a au plus deux choix possibles en sortant de chaque transition
    - le temps déterministe $2^{g(|(N,0)|)}$ est donc suffisant pour explorer toutes les exécutions possibles de N sur (N,0)
    - comme $|(N,i)| > i > 2^{g(|(N,0)|)}$ le temps d'exécution de M dans ce cas est O(n) avec n taille de l'entrée
  - sinon, on simule N sur l'entrée (N,i+1), de manière non déterministe pendant un temps g(|(N,i)|)
    - La simulation non déterministe prend un temps O(g(|(N,i)|))
- Supposons $L \in NTIME(f(n))$:
  - il existe une MTND V telle que L(V)=L et V s'exécute en kf(|N,i|)) pour une constante k
  - comme $f(n+1) \in o(g(n))$ on a $kk_N f(n+1) \leq g(n)$ pour tous les n suffisamment grands, avec k_N une constante ne dépendant que du code de N choisi
  - comme le code de N peut être choisi arbitrairement grand on a $kk_N f(|(N,i+1)|) \leq g(|(N,i)|)$ pour tout i
  - donc, la simulation de N sur l'entrée (N,i+1) de manière non déterministe pendant un temps g(|(N,i)|) termine
- En prenant N=V on a
  - $V((N,0)) = N((N,0)) = N((N,1)) = \ldots = N((N,2^{g(|(N,0)|)}+1))$ = négation de N((N,0)) : contradiction !

Relations entre les mesures en temps et en espace

- Si $L \in DTIME(f(n))$ alors $L \in DSPACE(f(n))$
  - le temps passé dans l'exécution limite le nombre de cellules nécessaires à l'exécution
- Si $L \in DSPACE(f(n))$ et $f(n)$ entièrement constructible en espace alors il existe une constante c_L telle que
  $L \in DTIME(c_L^{f(n)})$
  - donné par la construction « Espace connu → arrêt décidable »
- Si $L \in NTIME(f(n))$ alors il existe une constante c_L telle que $L \in DTIME(c_L^{f(n)})$
  - donné par la construction qui simule une MT non déterministe par une MT déterministe
- Si $L \in NTIME(f(n))$ alors $L \in DSPACE(f(n))$
  - en temps f(n) on explore au plus f(n) cellules
  - on simule la MT non déterministe par une MT déterministe qui réalise un parcours en profondeur des configurations possibles
  - la taille de la bande de la machine simulée est ≤ f(n) [idem pour le simulateur]
  - la taille de la pile du simulateur est ≤ f(n)

Relations entre les mesures en temps et en espace

- Si $L \in NSPACE(f(n))$ alors $L \in DTIME(O(2^{f(n)}))$
  - On simule la MT non déterministe par une MT déterministe qui réalise un parcours en profondeur des configurations possibles
  - Comme la taille de la bande est ≤ f(n), il y a $|Q| f(n) 2^{f(n)} \in O(2^{f(n)})$ configurations possibles (en supposant $\Gamma = \{ 0,1 \}$)

Théorème de Savitch

- Si $L \in NSPACE(S(n))$, $S(n) \geq \log_2(n)$ et $S(n)$ est entièrement constructible en espace, alors $L \in DSPACE(S(n)^2)$
- La MT déterministe M teste l'existence d'une exécution acceptante de la MT non déterministe M' par récurrence :
  - il existe un chemin de s vers t en k étapes dans M' s'il existe une configuration x de M', un chemin de s vers x en k/2 étapes et un de x vers t en k/2 étapes
  - le nombre d'appels récursifs à réaliser est $\log_2 k$
  - on a $O(2^{S(n)})$ configurations possibles de M'
  - chaque configuration occupe une place en $O(S(n))$
  - au pire, le nombre d'étapes à considérer est $O(2^{S(n)})$
  - le test d'existence d'une exécution acceptante de M' est donc dans $DSPACE(S(n)^2)$
- Aucun équivalent du théorème de Savitch n'est connu pour le temps

Un lemme de translation pour NSPACE

- Soient $S_1(n)$, $S_2(n)$, $f(n)$ des fonctions entièrement constructibles en espace vérifiant $S_2(n) \geq n$ et $f(n) \geq n$.
  Si $NSPACE(S_1(n)) \subseteq NSPACE(S_2(n))$ alors $NSPACE(S_1(f(n))) \subseteq NSPACE(S_2(f(n)))$
- Soit M_1 une MTND, $L_1 = L(M_1) \in NSPACE(S_1(f(n)))$
- Soit $L_2 = \{ x\$^i : M_1$ accepte x en $S_1(|x|+i) \}$ accepté par un MTND M_2 (donc dans $NSPACE(S_1(n))$)

  On peut toujours choisir i vérifiant $f(|x|)=|x|+i$ pour avoir $S_1(|x|+i) = S_1(f(|x|))$
  Donc, pour chaque $x \in L(M_1)$, au moins un i tel que $x\$^i \in L(M_2)$
  L_2 s'accepte en $NSPACE(S_1(n))$, donc en $NSPACE(S_2(n))$ -> MTND M_3
- On construit une MTND M_4 pour L_1 dans $NSPACE(S_2(f(n)))$
  - entrée $x\$^i$
  - on marque « autorisées » $S_2(f(n))$ cellules de la bande
  - M_4 simule M_3 dans la zone autorisée, tant que le curseur est sur x
  - quand le curseur passe dans la zone $\$^i$, M_4 simule un compteur, de taille $\log_2(f(|x|)-|x|)$, pour se repérer dans cette zone
  - L'espace occupé par le compteur est $\log_2(f(|x|)-|x|) \leq f(|x|)-|x| \leq f(|x|) \leq S_2(f(|x|))$
  - M_4 reconnaît x ssi il existe i ($\in [0,f(|x|)[$) tel que $x\$^i \in L_2$
    il existe i tel que $x\$^i \in L_2$
    $x \in L_1$

Théorème de hiérarchie pour NSPACE

- Si $\varepsilon>0$ et $r \geq 0$, alors
  $NSPACE(n^r) \subsetneq NSPACE(n^{r+\varepsilon})$
- Preuve : par le calcul, en corollaire du lemme de translation pour NSPACE et du théorème de Savitch

Des classes de problèmes

- La classe des problèmes solvables en temps déterministe polynomial

  $P = \cup_{i \geq 1} DTIME(n^i)$

  - Exemple : déterminer si un graphe est connexe
- La classe des problèmes solvables en temps non-déterministe polynomial

  $NP = \cup_{i \geq 1} NTIME(n^i)$

  - Exemple : trouver un circuit hamiltonien (passant exactement une fois par chaque nœud) dans un graphe
- La classe des problèmes solvables en espace déterministe polynomial

  $PSPACE = \cup_{i \geq 1} DSPACE(n^i)$
- La classe des problèmes solvables en espace non-déterministe polynomial
  $NSPACE = \cup_{i \geq 1} NSPACE(n^i)$
- La classe des problèmes solvables en temps déterministe exponentiel
  $EXP = \cup_{i \geq 1} DTIME(2^{(n^i)})$
  - Exemple : déterminer le complémentaire d'un automate (de Kleene)
- La classe des problèmes solvables en temps non-déterministe exponentiel
  $NEXP = \cup_{i \geq 1} NTIME(2^{(n^i)})$

Ce qu'on sait

- PSPACE = NSPACE
  - car, par le théorème de Savitch, $NSPACE(n^i) \subseteq DSPACE(n^{2i})$
- $DSPACE(\log n) \subseteq P \subseteq NP \subseteq PSPACE$
- $P \subseteq NP \subseteq EXP \subseteq NEXP$
- $P \subsetneq EXP$
  - pour tout i, $n^i < n^{\log n}$ pour n suffisamment grand, donc $P \subseteq DTIME(n^{\log n})$,
    et $DTIME(n^{\log n}) \subsetneq DTIME(2^n)$ par application du théorème de hiérarchie en temps déterministe
- $NP \subsetneq NEXP$
  - même argumentation
- P est close par union, intersection, complémentaire
  - union : on simule M MT déterministe, si elle répond oui on réponds oui, sinon on réponds la même chose que M' MT déterministe
  - intersection : similaire
  - complément : on simule M et on réponds l'inverse, le temps est inchangé
- NP est close par union et intersection
  - même argumentation que pour P, sauf pour le complémentaire qu'on ne sait pas prendre efficacement

Ce qu'on ne sait pas

- dans DSPACE(log n) $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE,
  - on sait qu'au moins une est stricte, car DSPACE(log n) $\subsetneq$ PSPACE (théorème de hiérarchie en espace)
  - la question P = NP fait partie des 7 problèmes du millénaire
    - http://www.claymath.org/millennium-problems
    - 1.000.000$ et célébrité à la clef !
    - une preuve constructive fournirait des solutions efficaces à des problèmes en mathématiques, informatique, ingénierie, économie, électronique, …
- si NP est close par complémentaire
  - on définit la classe coNP des problèmes dont le complémentaire $\in$ NP

Exemples de problèmes NP

- Circuit hamiltonien
  - trouver un chemin passant exactement une fois par chaque nœud d'un graphe
- Clique
  - trouver une partie d'un graphe telle que les nœuds sont tous deux à deux reliés
- SAT
  - trouver une interprétation des variables libres d'une formule de logique booléenne $\Phi(x_1,\ldots,x_n)$ qui la satisfasse
- Factoriser un entier x en $x=x_1x_2$
  - c'est-à-dire, trouver des suites de bits pour x_1 et x_2 tels que le produit fasse x
- Plein d'autres ! et très utiles !

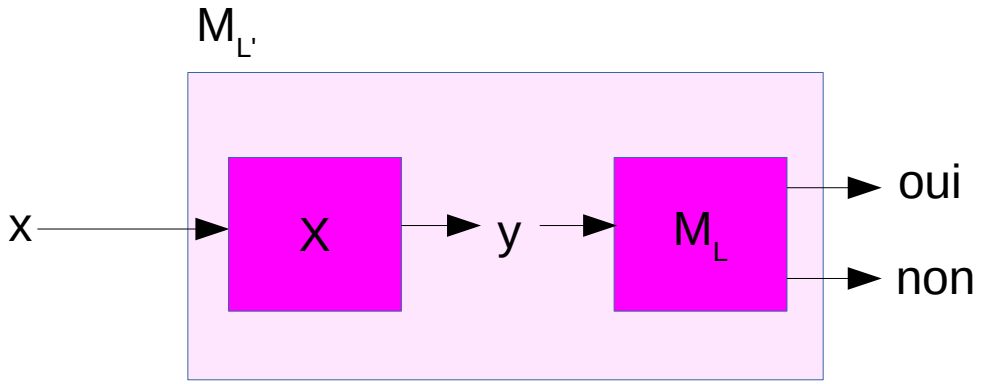
Si P = NP alors EXP = NEXP

(on ne sait pas si la réciproque est vraie)

- Soit L ∈ NEXP.
  - L est reconnu par une MTND M répondant en 2^{n^k}
- Soit L' = { $(x,1^{2^{(n^k)}})$: x ∈ L }
  - Taille d'un mot de L' = $|x|+1+2^{n^k}$
- Soit M' la MTND réalisant
  - le contrôle que son entrée est un couple (x,y)
    - temps linéaire en fonction de la taille de l'entrée (x,y)
  - dont la seconde composante est de la forme $1^{2^{(n^k)}}$
    - temps linéaire en fonction de la taille de l'entrée (x,y)
  - et exécutant M(x) pour vérifier x ∈ L
    - temps linéaire en fonction de la taille de l'entrée (x,y)
- Comme M' s'exécute en temps linéaire en fonction de la taille de l'entrée, L' ∈ NP
- Donc par hypothèse L' ∈ P : il existe une MT det. M'' reconnaissant L' en temps $O(n^{k'})$
- On en déduit une MT dét. pour L fonctionnant en temps exponentiel de son entrée x :
  - elle simule M'' sur $(x,1^{2^{(n^k)}})$

Réduction en temps polynomial

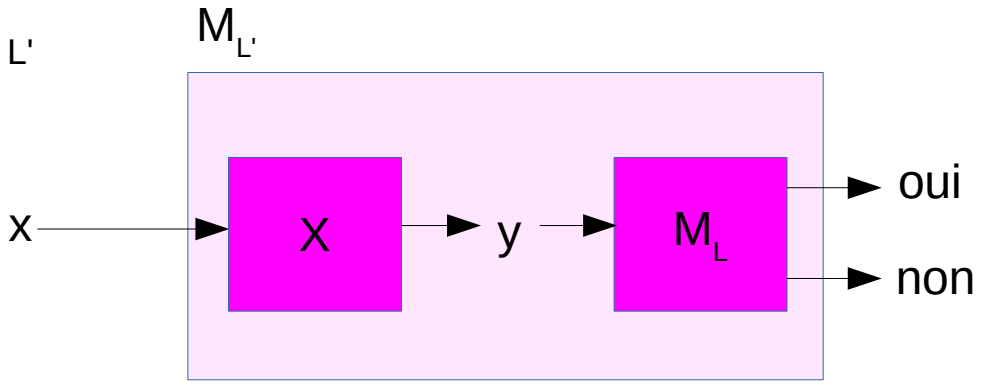
- L' se réduit en temps polynomial à L s'il existe une MT X
  - prenant une entrée x
  - produisant en temps polynomial une sortie y
  - telle que $y \in L$ ssi $x \in L'$



- Si L est dans P, alors L' aussi
- Si L est dans NP, alors L' aussi

Réduction en espace logarithmique

- Transducteur en espace logarithmique: machine s'arrêtant toujours avec
  - une bande de travail de taille log(n), avec n taille de l'entrée
  - une bande de sortie sur laquelle la tête ne revient jamais en arrière
- L' se réduit en espace logarithmique à L s'il existe un transducteur en espace logarithmique X
  - prenant une entrée x
  - produisant une sortie y
  - telle que y ∈ L ssi x ∈ L'



- Si L est dans P, alors L' aussi
- Si L est dans NSPACE($\log^k n$), alors L' aussi
- Si L est dans DSPACE($\log^k n$), alors L' aussi

Réduction en espace logarithmique

- Si L est dans P, alors L' aussi
  - On suppose pour simplifier que X produit sa sortie sur une bande différente, en lecture seule et sans retour arrière
  - Comme X ne boucle pas, elle produit au plus autant de caractères sur sa sortie que d'états différents de M_X
  - Il y en a $O((\log n)(\Sigma^{\log n})n|Q_X|)$, donc moins de $O(n^3)$
- Si L est dans NSPACE($\log^k n$), alors L' aussi
  - Similaire au cas suivant
- Si L est dans DSPACE($\log^k n$), alors L' aussi
  - Difficulté : il faut stocker y dans un espace logarithmique, or y a taille $O(n^c)$
  - Solution : nourrir M_L au fur et à mesure de la production de la sortie de X
    - $M_{L'}$ mémorise la position de la tête de M_L sur y : taille de la mémoire=c log n
    - pour reproduire si nécessaire le ième caractère de y, $M_{L'}$ relance X après avoir mémorisé sa configuration (espace logarithmique nécessaire)

Langages complets et durs

- R une catégorie de réductions
- C une classe de langages
- L un langage
- L est **C-complet** pour R
  - $L \in C$
  - Tout langage de C se réduit par R en L
- L est **C-dur** pour R
  - Tout langage de C se réduit par R en L
- Exemple : Si L est NP-complet pour la réduction polynomiale en temps, alors $L \in P$ implique $P = NP$
- Exemple : Si L est NP-complet pour la réduction polynomiale en temps, et $L' \in NP$, alors si L se réduit en temps polynomial en L', L' est aussi NP complet.

SAT : un problème NP-complet

Expressions booléennes :

- variables $x_1, \ldots, x_n$ à valeur dans $\mathbb{B}$;
- connecteurs propositionnels $\neg, \wedge, \vee$.

Forme normale conjonctive (FNC)

- ψ est de la forme $\psi \equiv \wedge_{i \in I} \vee_{j \in J} \psi_{i,j}$
- chaque $\psi_{i,j}$ est de la forme
  - x_k
  - $\neg x_k$

Chaque expression booléenne admet une FNC

Satisfaisabilité de ψ :

Trouver une valuation de $x_1, \ldots, x_n$ qui rende ψ vraie.

Theorem

La satisfaisabilité SAT des expressions booléennes est un problème NP-complet.

SAT : un problème NP-complet : preuve

$SAT(\psi) \in NP$ car :

- Vérifier qu'une valuation pour $x_1, \ldots, x_n$ permette de satisfaire ψ est linéaire en la taille de ψ ;
- Pour trouver une valuation, il faut en choisir une de manière non-déterministe et vérifier qu'elle satisfait ψ.

Complétude

- une réduction, en temps polynomial, de tout problème NP vers *SAT*
- idée : coder les exécutions acceptantes de toute MT non déterministe $M \in NTIME(p(n))$ par une expression booléenne ψ_M
  - le temps d'execution $p(n) > 0$ de M est un polynôme

$$T = \{0, \ldots, p(n)\}$$

  - la taille nécessaire de bande de M est donc au plus de $p(n)$ cases

$$E = \{0, \ldots, p(n) - 1\}$$

SAT : un problème NP-complet : preuve de complétude

Variables de ψ_M

Pour tout $a \in \Gamma$, $i \in T$, $j \in E$,

$c_{a,i,j}$: au temps i, la lettre de la bande à la position j est un a ;

$p_{i,j}$: au temps i, la tête de lecture est à la position j ;

$e_{q,i}$: au temps i, M est dans l'état q.

Nombre de variables : polynomial !

$$|\Gamma||T||E| + |T||E| + |Q||T| =$$
$$|\Gamma|(p(n)+1)p(n) + (p(n)+1)p(n) + |Q|(p(n)+1)$$

SAT : un problème NP-complet : preuve de complétude

Encodage des propriétés de M

- A tout moment, chaque case de la bande contient exactement un caractère
  - au plus un

$$\bigwedge_{\substack{i\in T\\ j\in E\\ a\in\Gamma}} \left(c_{a,i,j} \rightarrow \bigwedge_{\substack{b\in\Gamma\\ b\neq a}} \neg c_{b,i,j}\right)$$

  - au moins un

$$\bigwedge_{\substack{i\in T\\ j\in E}} \bigvee_{a\in\Gamma} c_{a,i,j}$$

- A tout moment la tête est exactement sur une position de la bande
  - au plus une

$$\bigwedge_{\substack{i\in T\\ j\in E}} \left(p_{i,j} \rightarrow \bigwedge_{\substack{j'\in E\\ j'\neq j}} \neg p_{i,j'}\right)$$

  - au moins une

$$\bigwedge_{i\in T} \bigvee_{j\in E} p_{i,j}$$

SAT : un problème NP-complet : preuve de complétude - Encodage des propriétés de M

- A tout moment M est exactement dans un état
  - au plus un

$$\wedge_{\substack{i\in T\\ q\in Q}} (e_{q,i} \rightarrow \wedge_{\substack{q'\in Q\\ q'\neq q}} \neg e_{q',i})$$

  - au moins un

$$\wedge_{i\in T} \vee_{q\in Q} e_{q,i}$$

- Au temps 0, M est dans son état initial q_0

$$e_{q_0,0}$$

- Si on atteint un état final on y reste jusqu'à écoulement complet du temps

$$\wedge_{i\in T} \vee_{q\in F} e_{q,i} \rightarrow \wedge_{\substack{j\in T\\ i<j}} e_{q,j}$$

- Au temps $p(n)$, M est dans un état final

$$\vee_{q\in F} e_{q,p(n)}$$

SAT : un problème NP-complet : preuve de complétude - Encodage des propriétés de M

- Au temps 0, la tête est en début de bande

$$p_{0,0}$$

- Au temps 0, le contenu de la bande est $a_0 a_1 \dots a_k \# \dots \#$

$$(\wedge_{r\in\{0,\dots,k\}} c_{a_r,0,r}) \wedge (\wedge_{r\in E-\{0,\dots,k\}} c_{\#,0,r})$$

- Entre les temps t et $t+1$, seul le caractère sous la tête au temps t peut avoir changé

$$\wedge_{\substack{t\in T-\{p(n)\},j\in E,a\in\Gamma \\ j'\in T,j'\neq j}} (p_{t,j} \wedge c_{a,t,j'}) \rightarrow c_{a,t+1,j'}$$

- Et son changement est en cohérence avec δ

$$\wedge_{\substack{t\in T-\{p(n)\},j\in E,a\in\Gamma \\ b\in\Gamma,q\in Q}} (c_{a,t,j} \wedge p_{t,j} \wedge e_{q,t}) \rightarrow$$

$$\vee_{(q,a,q',b,D)\in\delta} (c_{b,t+1,j} \wedge p_{t+1,j+1} \wedge e_{q',t+1})$$

$$\vee_{\substack{(q,a,q',b,G)\in\delta \\ j\neq 0}} (c_{b,t+1,j} \wedge p_{t+1,j-1} \wedge e_{q',t+1})$$

SAT : un problème NP-complet : preuve de complétude - Encodage des propriétés de M

- ψ_M est la conjonction de toutes ces formules ;
- ψ_M a une taille polynomiale $p'(|M|)$;
- ψ_M se calcule en temps polynomial $p''(|M|)$;
- le nombre de solutions de ψ_M est égal au nombre de chemins acceptants de M ;
- chaque solution de ψ_M donne une sortie de M.

donc tout problème $\in NP$ donné par une MT non déterministe M se réduit en temps polynomial à SAT.

Bibliothèque de symboles

$$\in \alpha \notin \in \infty$$

$$N \rightarrow B(Q \cup \{q'_0, r\}, \Sigma, \Gamma \cup \{Z'_0\}, \delta, q'_0, Z'_0, F) \alpha \rightarrow \beta \varepsilon \models \subseteq \subsetneq \cap \cup$$

$$\equiv \lceil \rceil \varphi \Phi \pi \circ \varnothing \wp \text{℗} \stackrel{?}{=} \neq \text{PNP}$$