

Jean Goubault-Larrecq

Calculabilité

2. Machines I/O à k bandes de travail

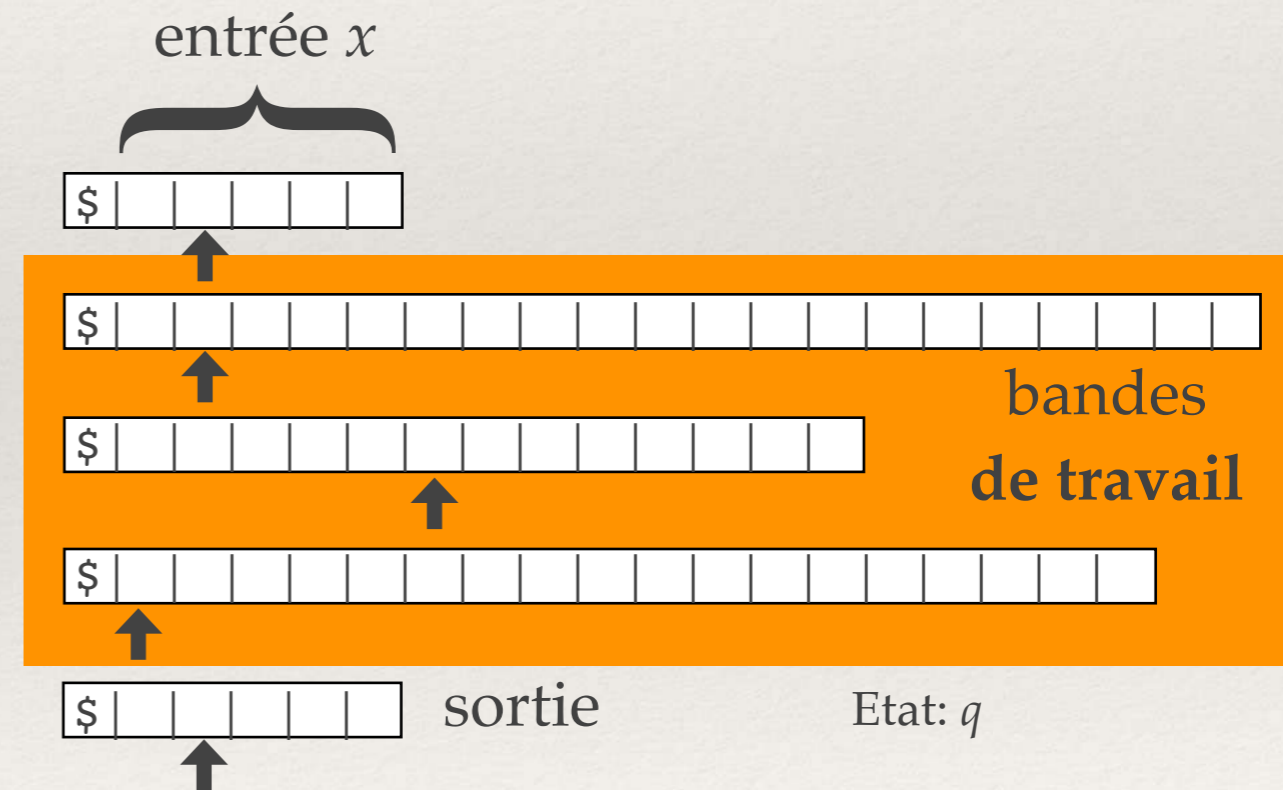
Aujourd'hui

- ❖ Machines I/O à k bandes
- ❖ Equivalence avec les machines de Turing à une bande
- ❖ Notions de complexité en temps, en espace
- ❖ Basé sur le poly d'Hubert Comon:
<http://www.lsv.fr/~comon/Calculabilite1/cours20.1.pdf>

Machines I/O à k bandes de travail

Machines I/O à k bandes de travail

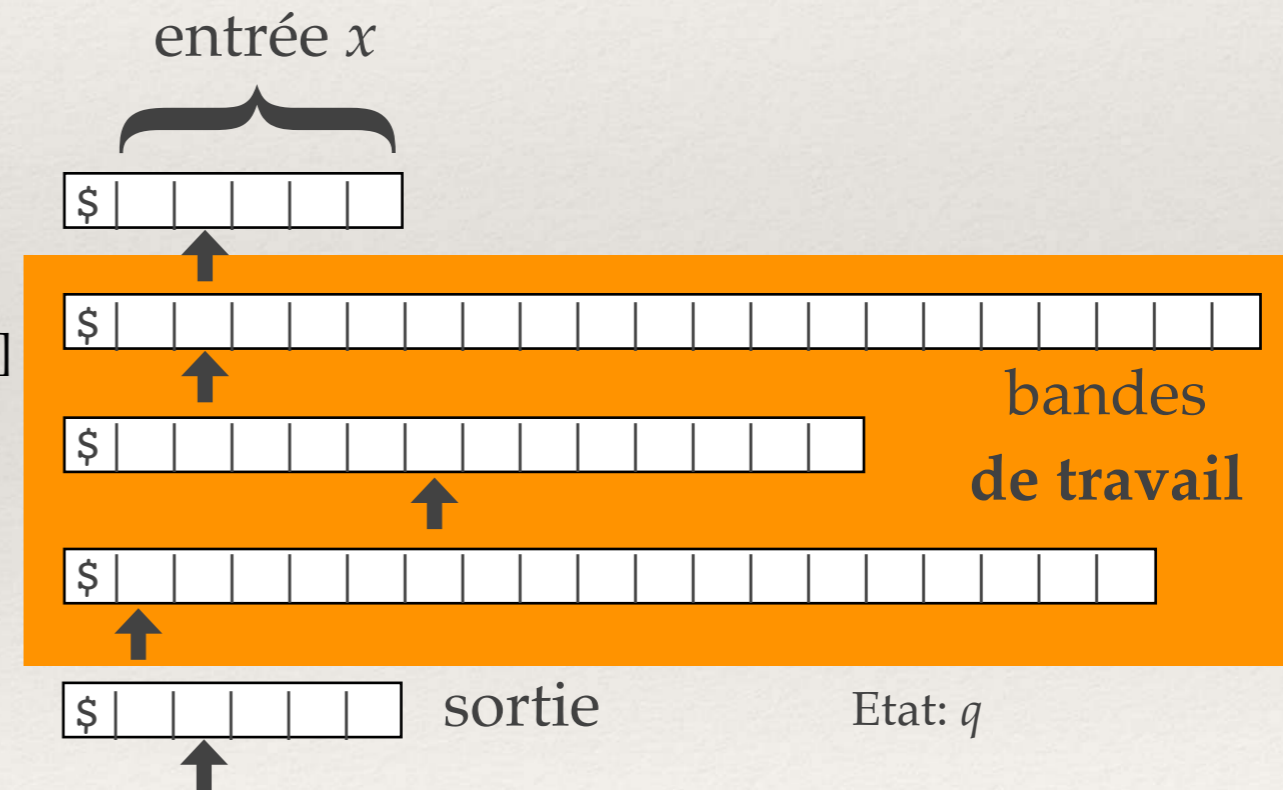
- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en **lecture seule**
 - 1 bande de sortie en **écriture seule**
 - $k \geq 1$ bandes **de travail**



Machines I/O à k bandes de travail

- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en lecture seule
 - 1 bande de sortie en écriture seule
 - $k \geq 1$ bandes de travail

- ❖ $\delta : Q \times \Sigma^{k+1} \rightarrow Q^+ \times \Sigma^{k+1} \times \text{Dir}^{k+2}$
[contraintes: écrit \$ uniquement en début de bandes
+ ne va jamais à droite du premier B sur la bande d'entrée]

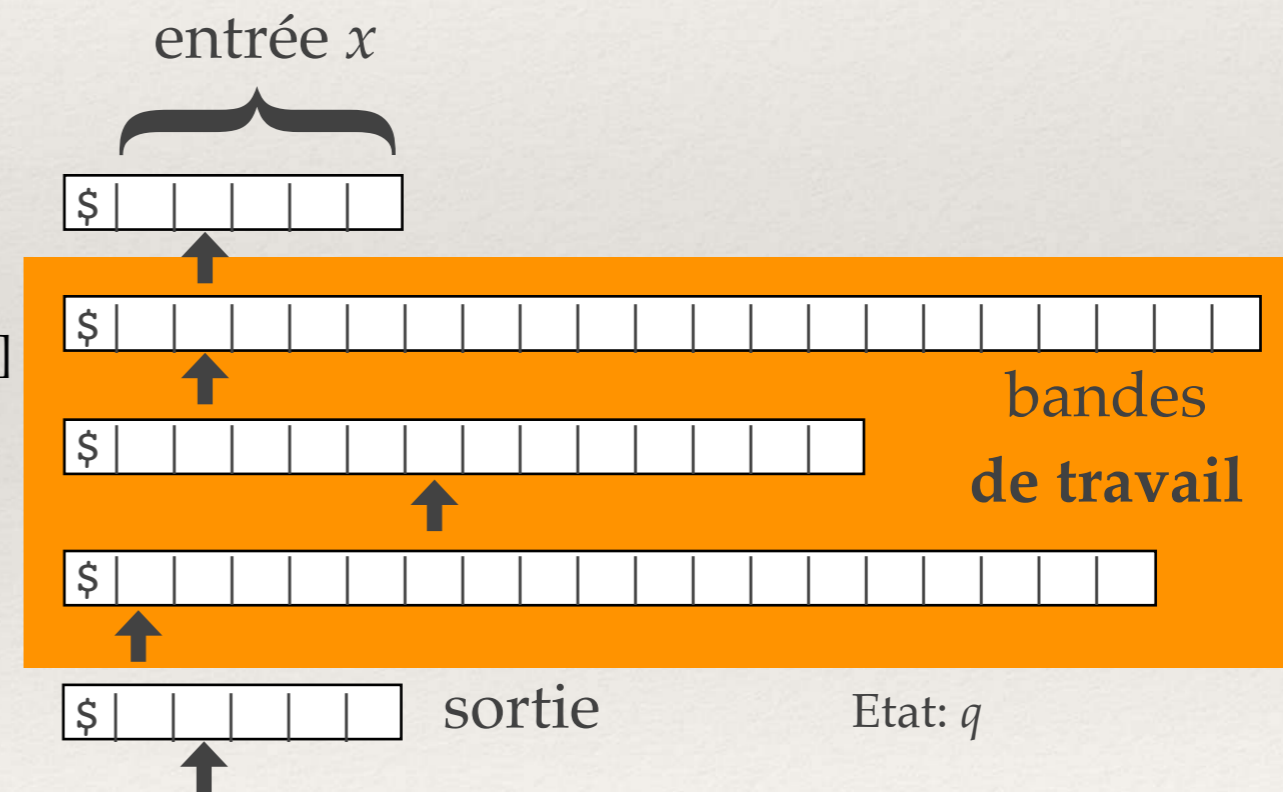


Machines I/O à k bandes de travail

- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en lecture seule
 - 1 bande de sortie en écriture seule
 - $k \geq 1$ bandes de travail

- ❖ $\delta : Q \times \Sigma^{k+1} \rightarrow Q^+ \times \Sigma^{k+1} \times \text{Dir}^{k+2}$
[contraintes: écrit \$ uniquement en début de bandes
+ ne va jamais à droite du premier B sur la bande d'entrée]

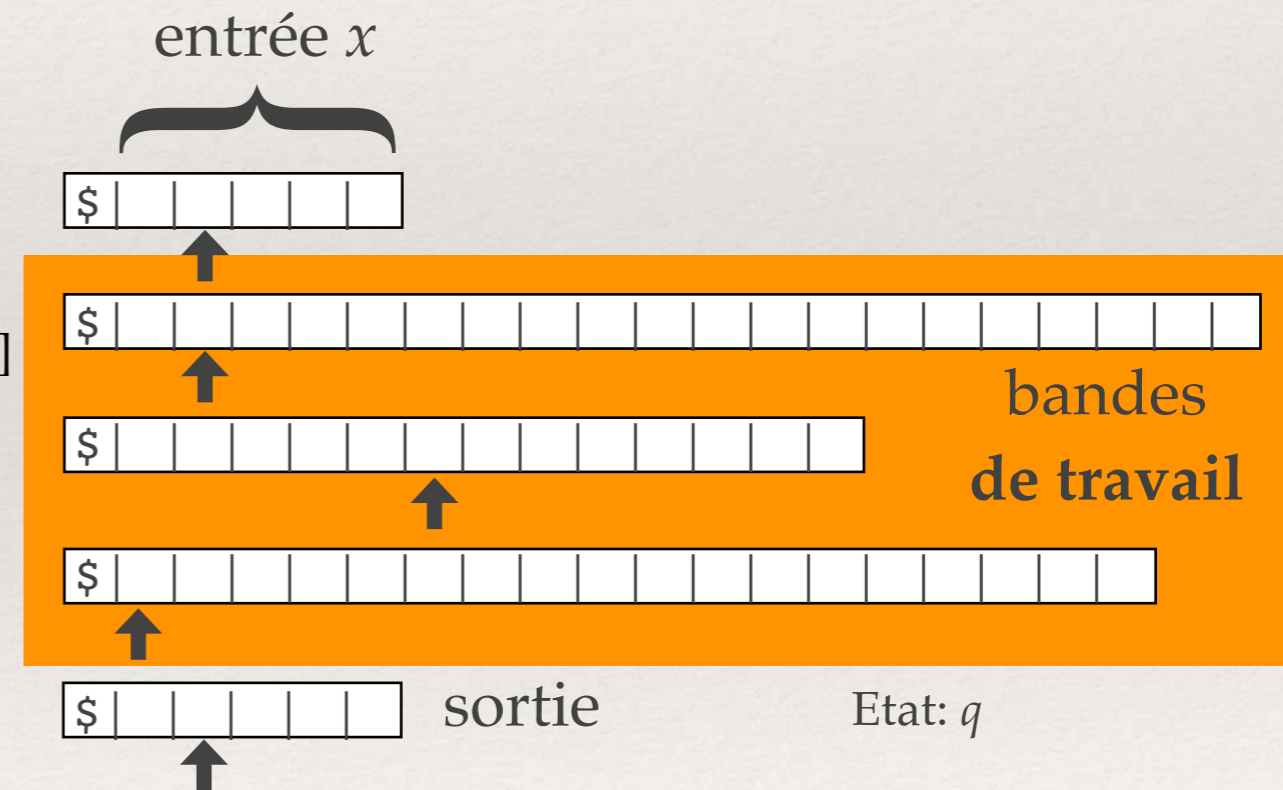
- ❖ Dans le poly: $k=1$



Machines I/O à k bandes de travail

- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en **lecture seule**
 - 1 bande de sortie en **écriture seule**
 - $k \geq 1$ bandes **de travail**

- ❖ $\delta : Q \times \Sigma^{k+1} \rightarrow Q^+ \times \Sigma^{k+1} \times \text{Dir}^{k+2}$
[contraintes: écrit \$ uniquement en début de bandes
+ ne va jamais à droite du premier B sur la bande d'entrée]



- ❖ Dans le poly: $k=1$
- ❖ Initialement, $\$x$ sur bande d'entrée, autres bandes vides (juste \$)

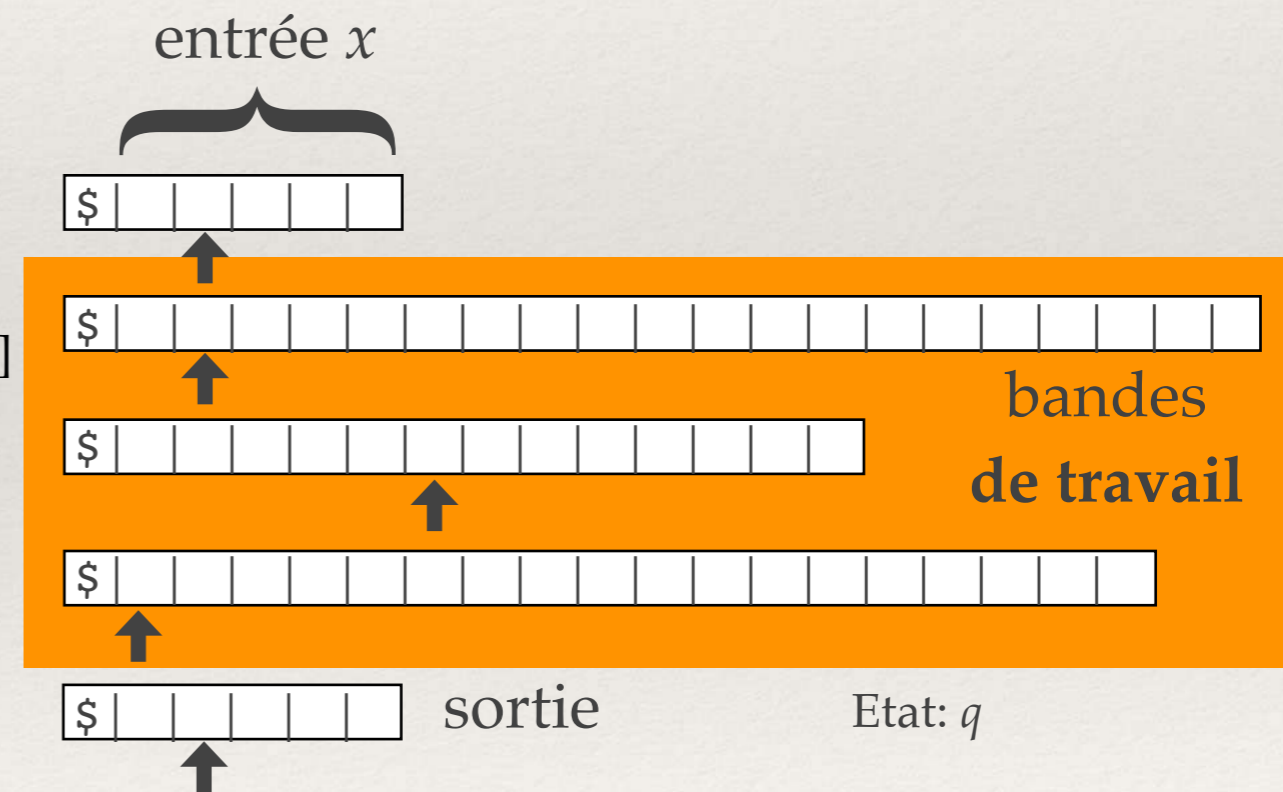
Machines I/O à k bandes de travail

- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en lecture seule
 - 1 bande de sortie en écriture seule
 - $k \geq 1$ bandes de travail

- ❖ $\delta : Q \times \Sigma^{k+1} \rightarrow Q^+ \times \Sigma^{k+1} \times \text{Dir}^{k+2}$
 [contraintes: écrit \$ uniquement en début de bandes
 + ne va jamais à droite du premier B sur la bande d'entrée]

pas $k+2$: on ne lit pas sur la bande de sortie

- ❖ Dans le poly: $k=1$
- ❖ Initialement, $\$x$ sur bande d'entrée, autres bandes vides (juste \$)



Machines I/O à k bandes de travail

- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en **lecture seule**
 - 1 bande de sortie en **écriture seule**
 - $k \geq 1$ bandes **de travail**

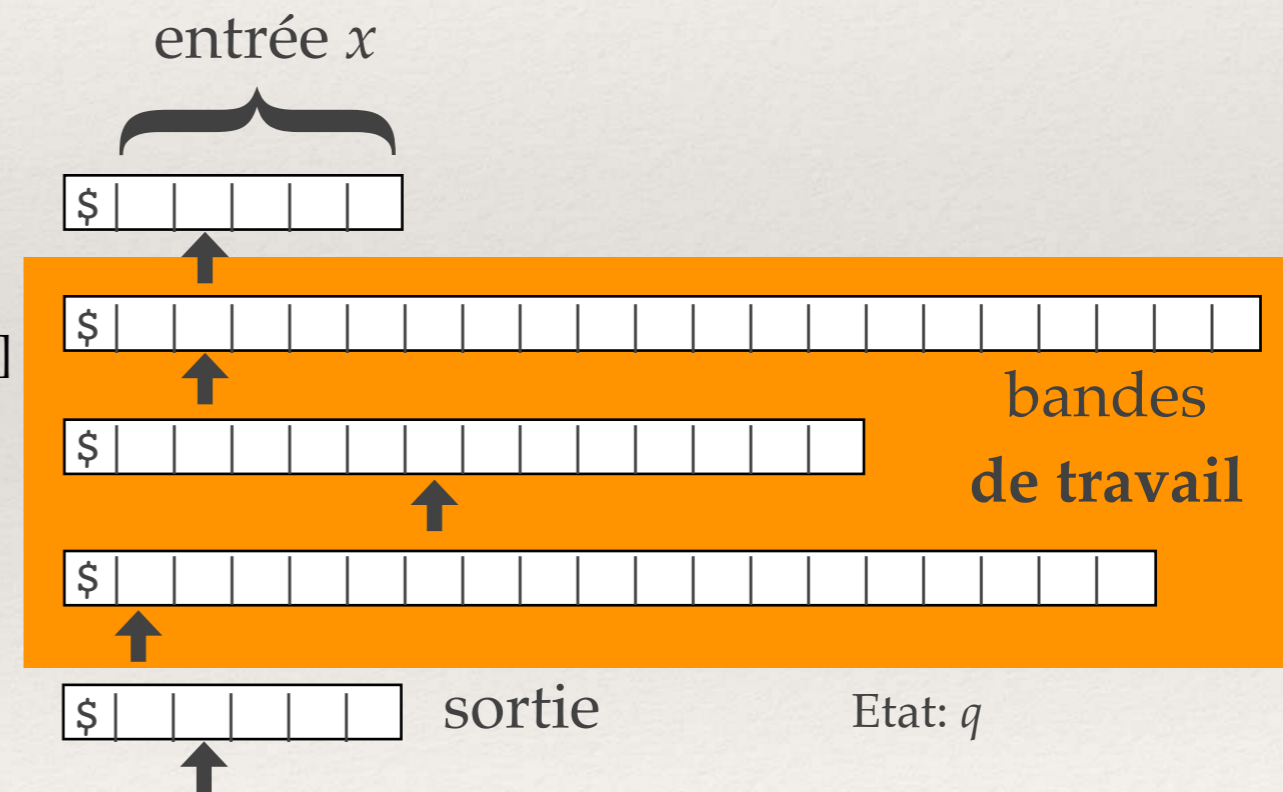
- ❖ $\delta : Q \times \Sigma^{k+1} \rightarrow Q^+ \times \Sigma^{k+1} \times \text{Dir}^{k+2}$
 [contraintes: écrit \$ uniquement en début de bandes
 + ne va jamais à droite du premier B sur la bande d'entrée]

pas $k+2$: on ne lit pas sur la bande de sortie

pas $k+2$: on n'écrit pas sur la bande d'entrée

- ❖ Dans le poly: $k=1$

- ❖ Initialement, $\$x$ sur bande d'entrée, autres bandes vides (juste \$)



Machines I/O à k bandes de travail

- ❖ $k+2$ bandes, dont
 - 1 bande d'entrée en lecture seule
 - 1 bande de sortie en écriture seule
 - $k \geq 1$ bandes de travail

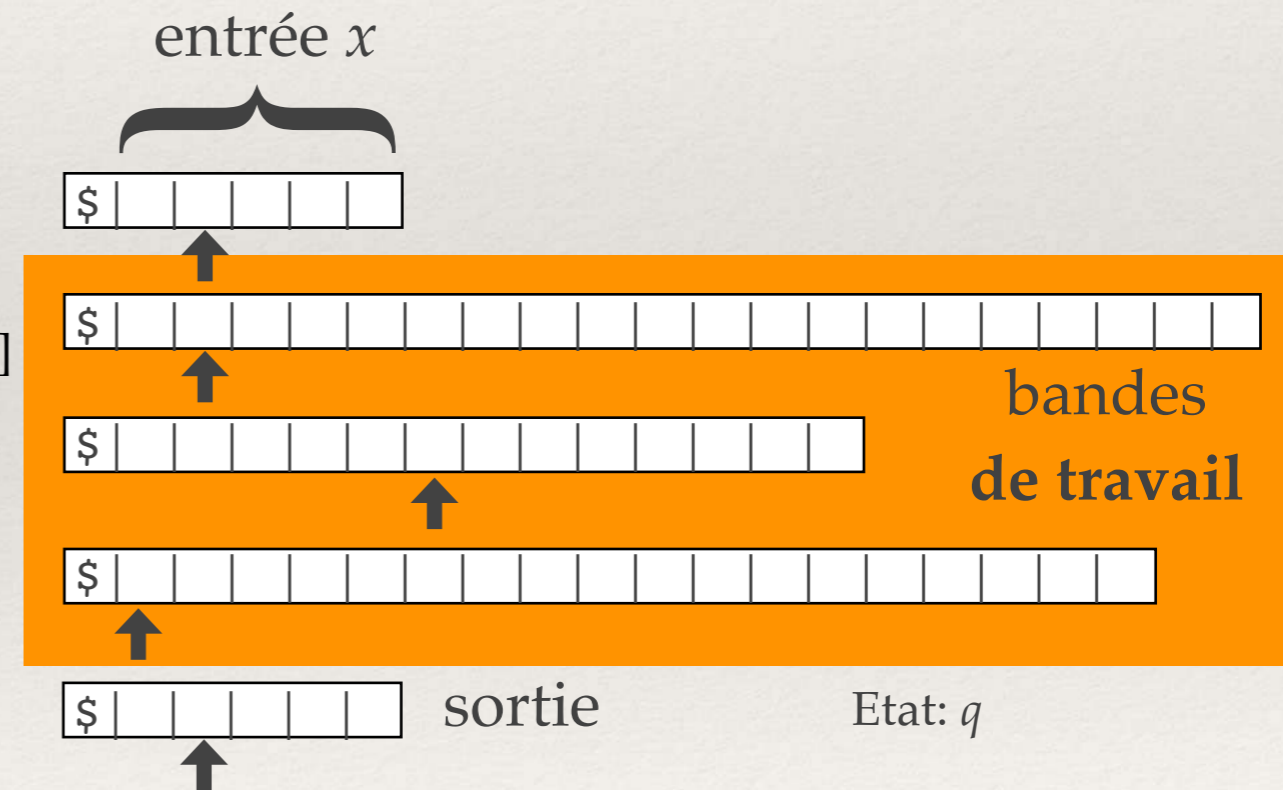
- ❖ $\delta : Q \times \Sigma^{k+1} \rightarrow Q^+ \times \Sigma^{k+1} \times \text{Dir}^{k+2}$
 [contraintes: écrit \$ uniquement en début de bandes
 + ne va jamais à droite du premier B sur la bande d'entrée]

pas $k+2$: on ne lit pas sur la bande de sortie

pas $k+2$: on n'écrit pas sur la bande d'entrée

- ❖ Dans le poly: $k=1$
- ❖ Initialement, $\$x$ sur bande d'entrée, autres bandes vides (juste \$)

bande de sortie optionnelle si on cherche juste à décider des langages

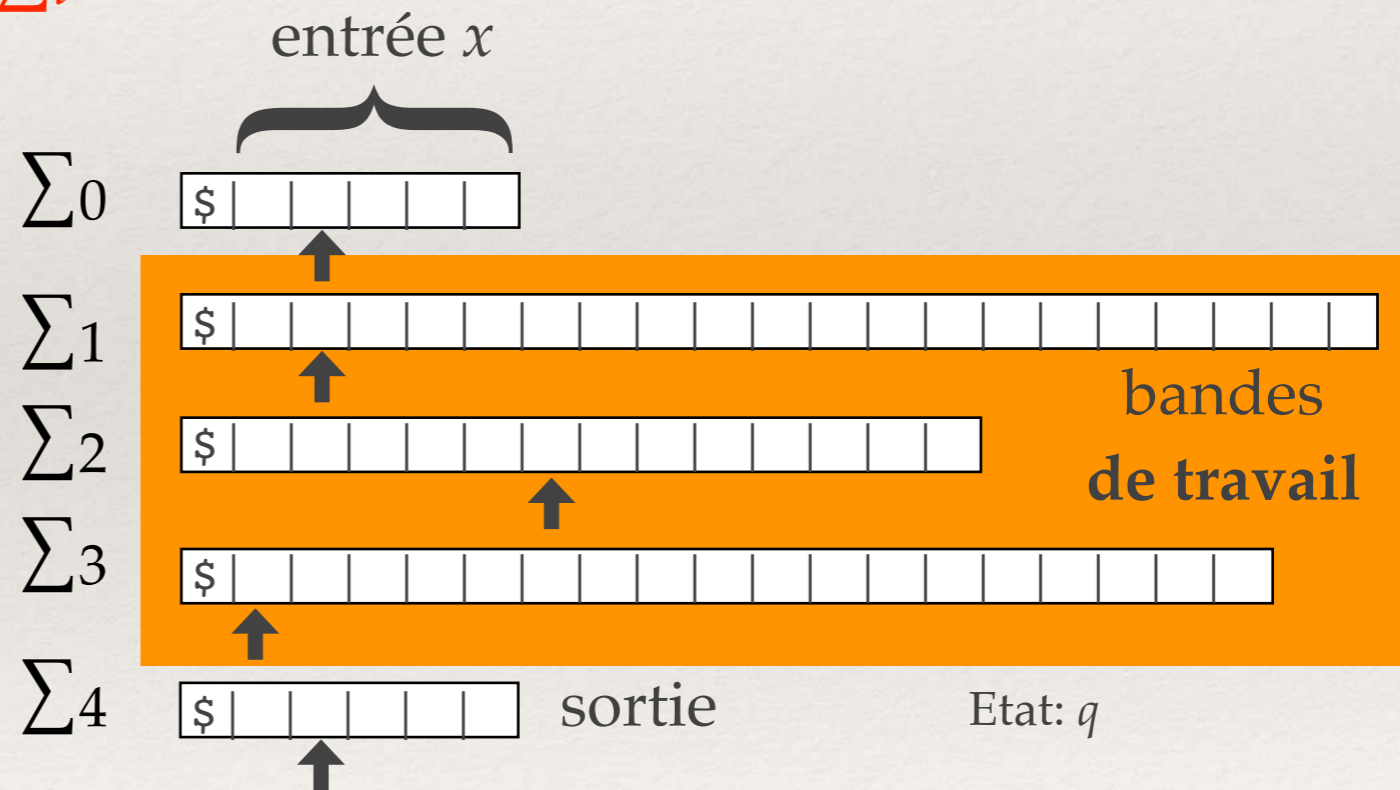


Machines I/O à k bandes de travail

❖ On peut encore raffiner: on n'a pas besoin que l'alphabet soit le même sur toutes les bandes

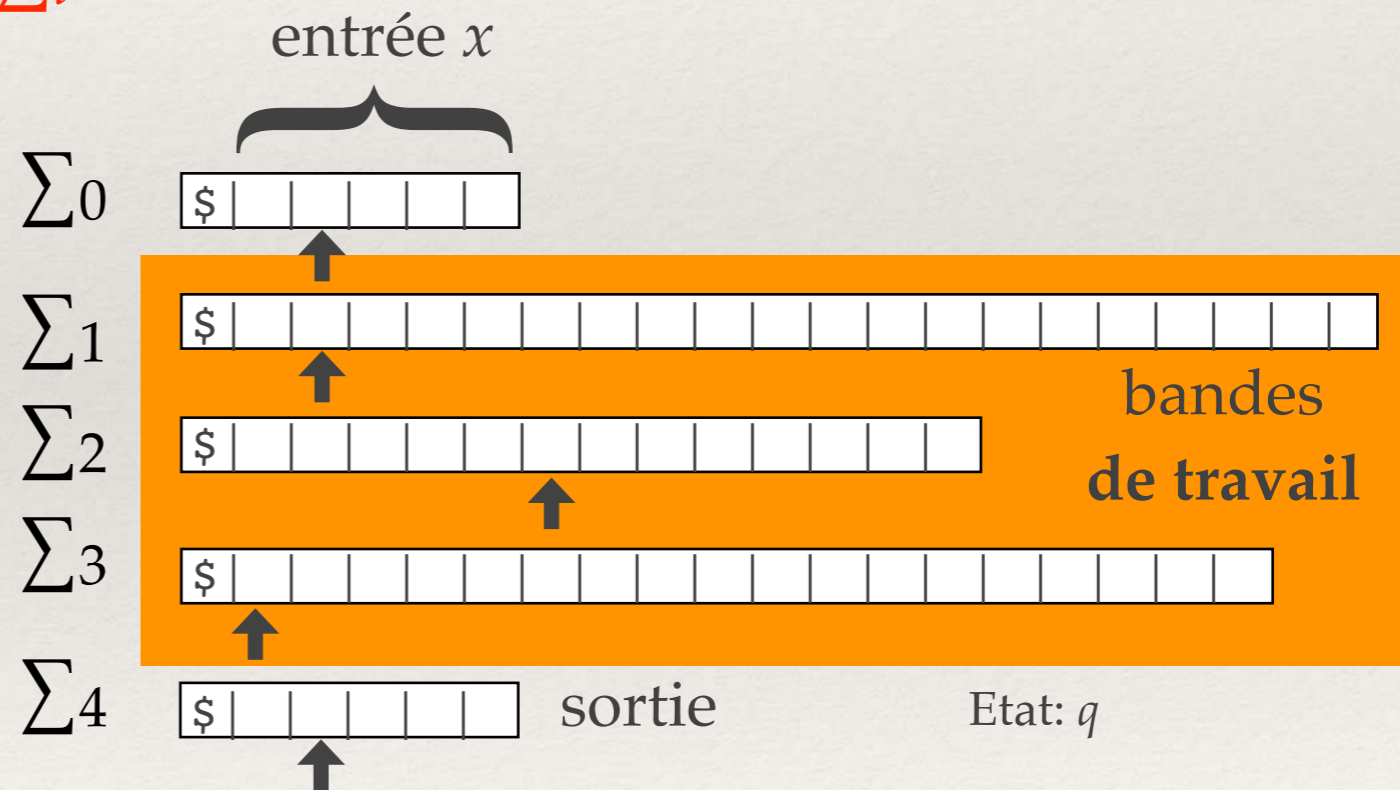
❖ $\delta : Q \times \prod_{i=0}^k \Sigma_i \rightarrow Q^+ \times \prod_{i=1}^{k+1} \Sigma_i \times \text{Dir}^{k+2}$

[soumis aux contraintes usuelles]



Machines I/O à k bandes de travail

- ❖ On peut encore raffiner: on n'a pas besoin que l'alphabet soit le même sur toutes les bandes
- ❖ $\delta : Q \times \prod_{i=0}^k \Sigma_i \rightarrow Q^+ \times \prod_{i=1}^{k+1} \Sigma_i \times \text{Dir}^{k+2}$
[soumis aux contraintes usuelles]
- ❖ Intérêt: on peut changer l'alphabet des **bandes de travail** en binaire, tout en laissant intact l'alphabet de l'entrée



Equivalence entre les modèles

- ❖ Tous ces modèles calculent les **mêmes** fonctions, et décident les **mêmes** langages
- ❖ On va le montrer, tout en montrant quels coûts ces traductions impliquent,
 - en **temps**,
 - et en **espace**

Complexité en temps, en espace

Temps

- ❖ Le **temps** d'exécution de M (1 bande, k bandes, I/O, etc.) sur l'entrée x est le nombre de mouvements effectués jusqu'à l'arrêt
[$=m$, si $\gamma_0 \vdash_M \gamma_1 \vdash_M \dots \vdash_M \gamma_m$ avec $\gamma_0 = (\varepsilon, q_0, \$x)$ et γ_m finale; ∞ sinon]
- ❖ M calcule **en temps** $f(n)$ ssi le temps d'exécution de M sur toute entrée de taille n
est $\leq f(n)$ [inférieur ou égal, pas =]

Classes de complexité en temps

- ❖ Un langage L est dans $\text{TIME}(f(n))$ ssi il est décidé par une machine I/O à k bandes M qui termine en temps $f(n)$ [pour un certain k]
- ❖ i.e.: pour toute entrée x [de taille n],
 M décide [accepte si dans L , rejette sinon]
en temps $f(n)$

Espace

- ❖ **L'espace** utilisé par une configuration est:
 - la taille de la bande [machine à 1 bande]
 - la somme des tailles des bandes [machines à k bandes]
 - la somme des tailles des bandes **de travail** [machines I/O]

Espace

- ❖ L'**espace** utilisé par une configuration est:
 - la taille de la bande [machine à 1 bande]
 - la somme des tailles des bandes [machines à k bandes]
 - la somme des tailles des bandes **de travail** [machines I/O]
- ❖ L'**espace** d'exécution de M sur l'entrée x est l'espace **sup** utilisé au cours de l'exécution correspondante

Espace

- ❖ L'**espace** utilisé par une configuration est:
 - la taille de la bande [machine à 1 bande]
 - la somme des tailles des bandes [machines à k bandes]
 - la somme des tailles des bandes **de travail** [machines I/O]
- ❖ L'**espace** d'exécution de M sur l'entrée x est l'espace **sup** utilisé au cours de l'exécution correspondante
- ❖ M calcule **en espace** $f(n)$ ssi l'espace d'exécution de M sur toute entrée de taille n
est $\leq f(n)$ [inférieur ou égal, pas =]

Classes de complexité en espace

- ❖ Un langage L est dans **SPACE($f(n)$)** ssi il est décidé par une machine I/O à k rubans M qui termine en espace $f(n)$ [pour un certain k]
- ❖ i.e.: pour toute entrée x [de taille n],
 M décide [accepte si dans L , rejette sinon]
en espace $f(n)$
- ❖ Note: M termine, mais on n'a pas dit en combien de temps

Equivalence des modèles de calcul

Une bande $\rightarrow k$ bandes

- ❖ Trivial ($k \geq 1$),
à condition que l'entrée **et** la sortie soient sur la bande 1

Une bande $\rightarrow k$ bandes

- ❖ Trivial ($k \geq 1$),
à condition que l'entrée **et** la sortie soient sur la bande 1
- ❖ Avec l'entrée sur la bande 1 et la sortie sur la bande k ,
on doit d'abord **copier** la bande 1 sur la bande k ,
puis simuler M en utilisant la bande k exclusivement

Une bande $\rightarrow k$ bandes

- ❖ Trivial ($k \geq 1$),
à condition que l'entrée **et** la sortie soient sur la bande 1
- ❖ Avec l'entrée sur la bande 1 et la sortie sur la bande k ,
on doit d'abord **copier** la bande 1 sur la bande k ,
puis simuler M en utilisant la bande k exclusivement
- ❖ Si M en temps $f(n)$, M' en temps $f(n) + 2n$
Si M en espace $g(n)$, M' en espace $g(n) + n$

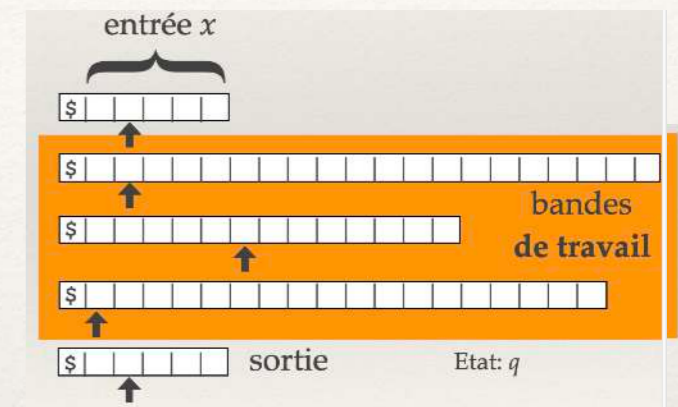
k bandes $\rightarrow k$ bandes I/O

- ❖ Soit M à k bandes,
entrée sur bande 1, sortie sur bande k



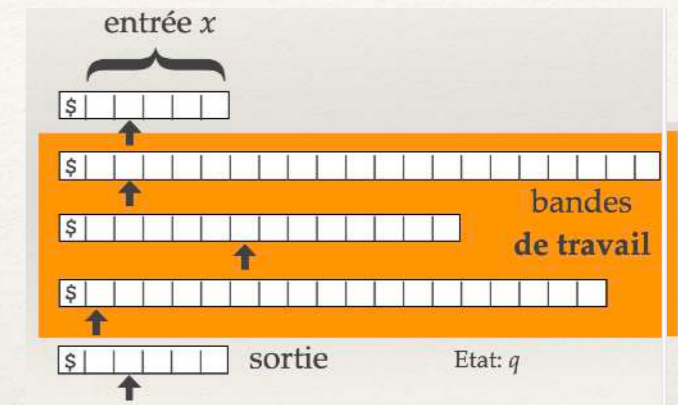
k bandes $\rightarrow k$ bandes I/O

- ❖ Soit M à k bandes, entrée sur bande 1, sortie sur bande k
- ❖ M' : — recopie la bande 1 sur la bande 2
— simule M sur les bandes $2 \dots k+1$ (au lieu de $1 \dots k$) sans lire ni écrire sur les bandes 1 et $k+2$
— si **accept**, recopie la bande $k+1$ sur la bande $k+2$



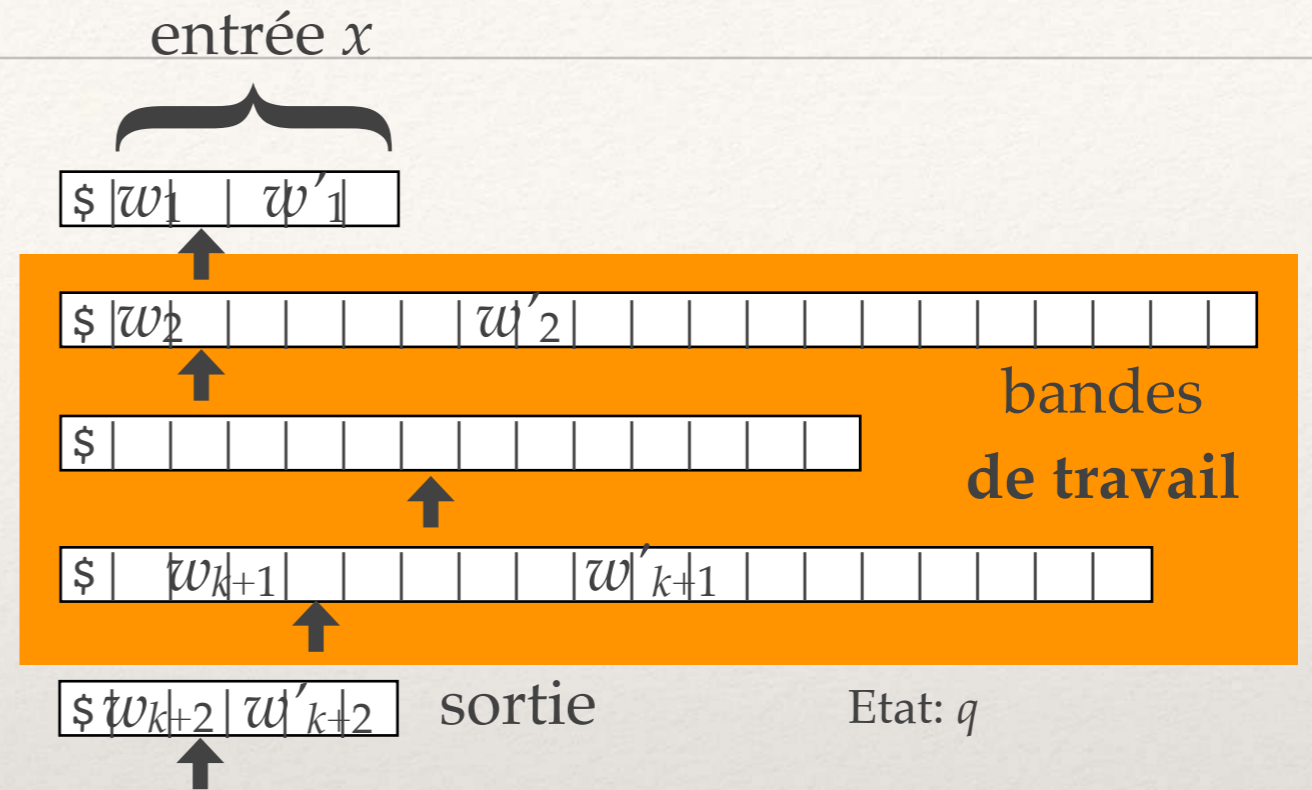
k bandes $\rightarrow k$ bandes I/O

- ❖ Soit M à k bandes, entrée sur bande 1, sortie sur bande k
- ❖ M' : — recopie la bande 1 sur la bande 2
— simule M sur les bandes $2 \dots k+1$ (au lieu de $1 \dots k$)
sans lire ni écrire sur les bandes 1 et $k+2$
— si **accept**, recopie la bande $k+1$ sur la bande $k+2$
- ❖ Si M en temps $f(n)$ et en espace $g(n)$,
 M' en temps $f(n)+2n+g(n)$
 M' en espace $g(n)$ [rappel: on ne compte que l'espace de bandes de travail]



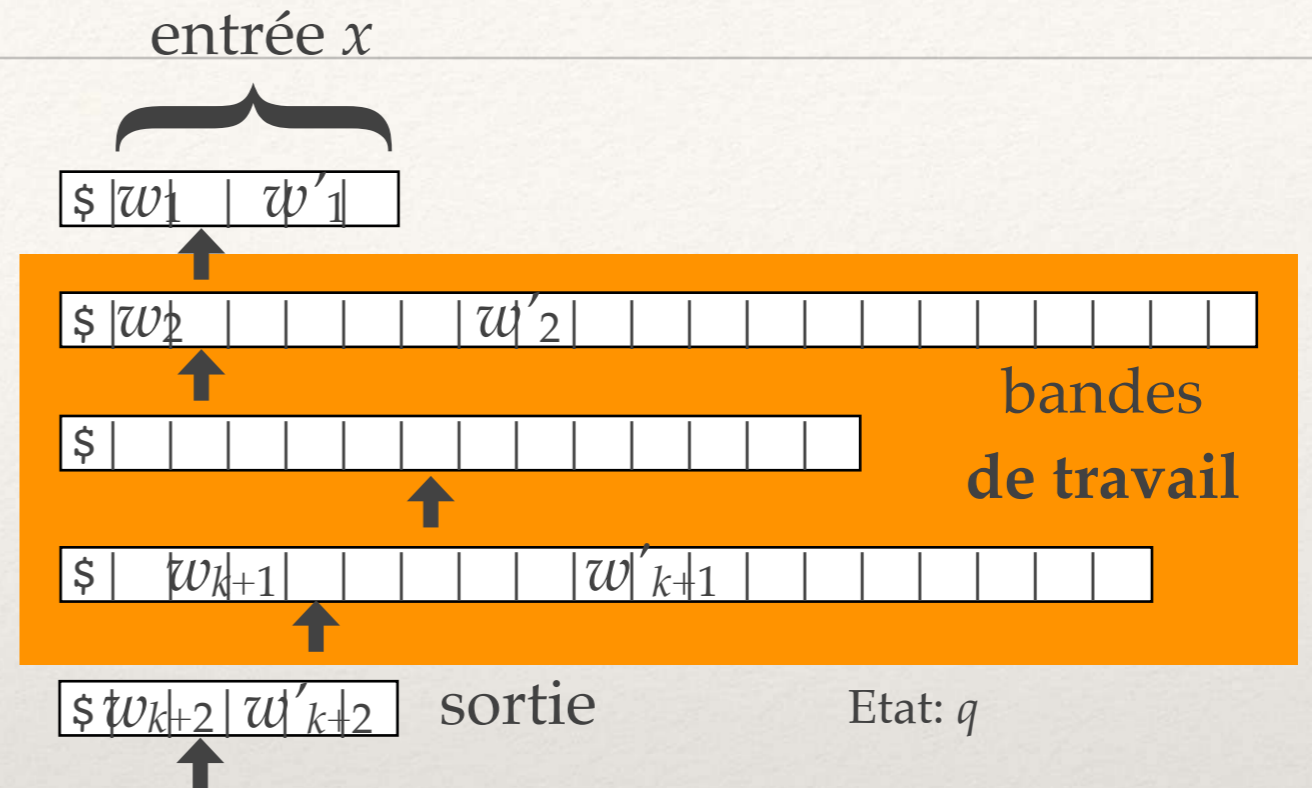
k bandes I/O \rightarrow une bande

- ❖ C'est bien sûr la partie intéressante.



k bandes I/O \rightarrow une bande

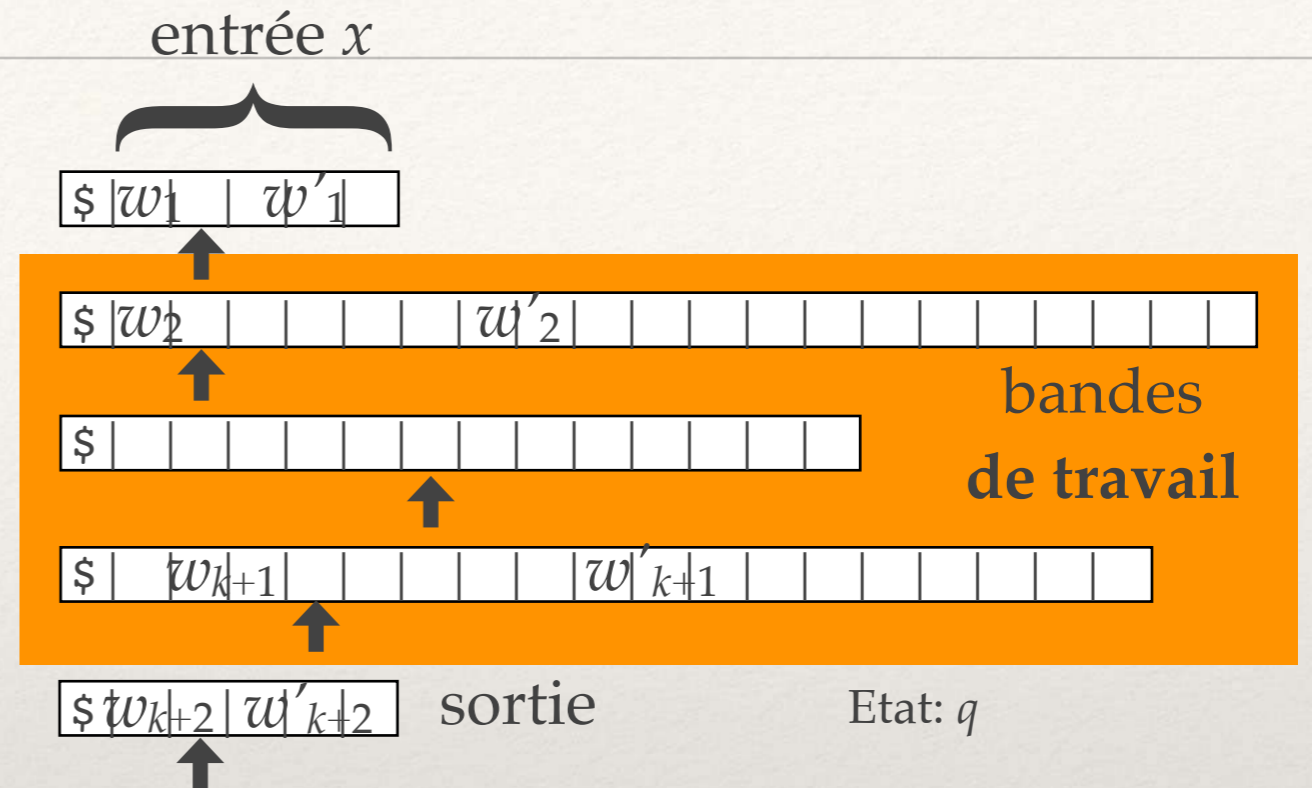
- ❖ C'est bien sûr la partie intéressante.
- ❖ On va recoder chaque configuration:



k bandes I/O \rightarrow une bande

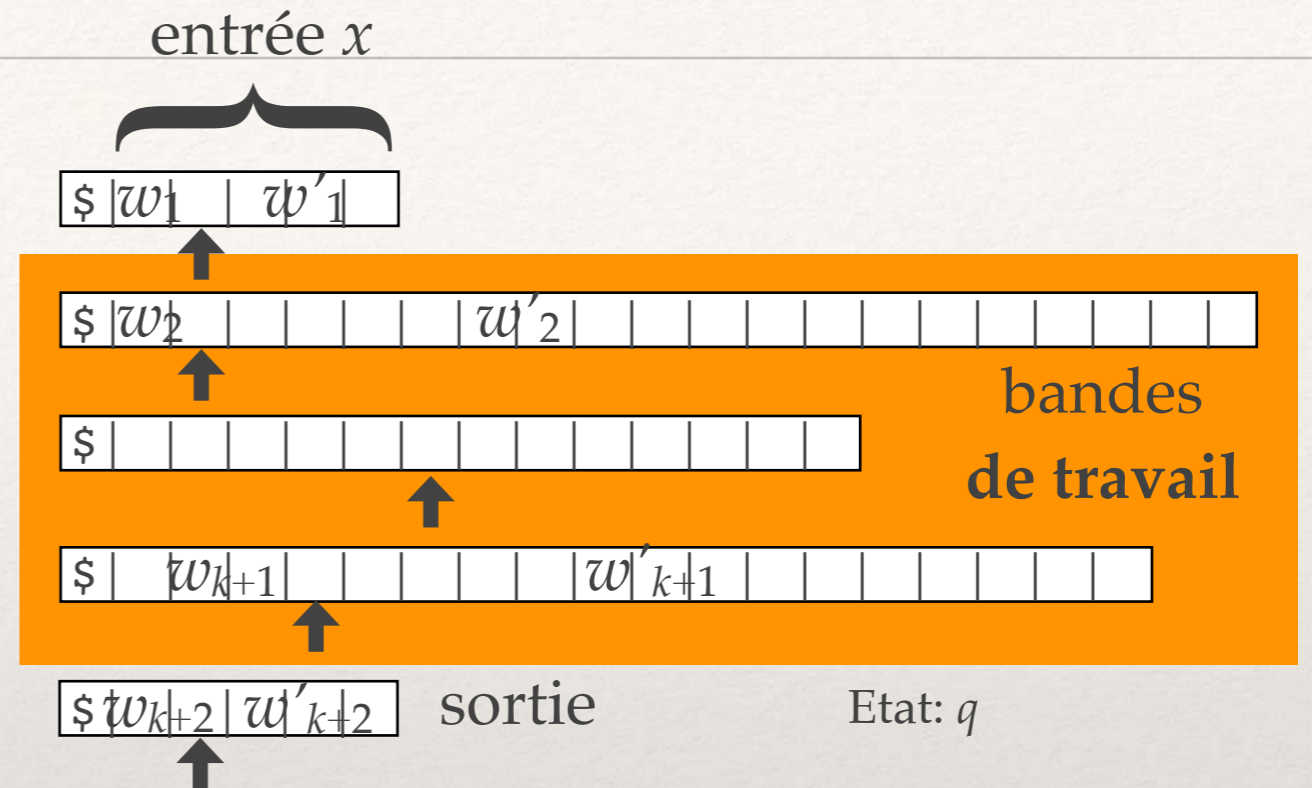
- ❖ C'est bien sûr la partie intéressante.
- ❖ On va recoder chaque configuration:
- ❖ sur **une seule bande**:

$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$



k bandes I/O \rightarrow une bande

- ❖ C'est bien sûr la partie intéressante.
- ❖ On va recoder chaque configuration:
- ❖ sur **une seule bande**:

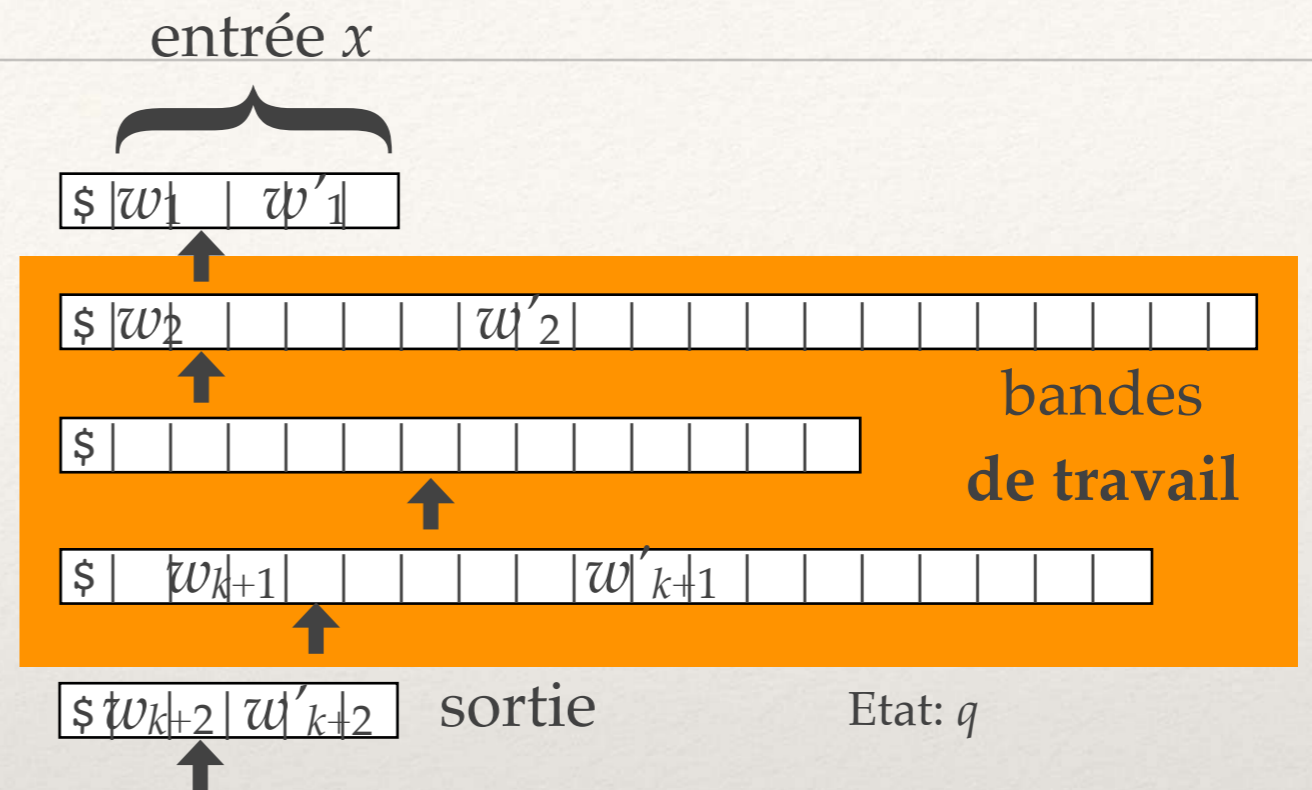


$$\underline{\$} w_1 \# w'_1 D w_2 \# w'_2 D \dots w_{k+2} \# w'_{k+2} D F$$

- ❖ l'état courant entre deux étapes simulées étant $[q, \varepsilon, 0]$
la tête entre deux étapes simulées étant complètement à gauche

k bandes I/O \rightarrow une bande

- ❖ C'est bien sûr la partie intéressante.
- ❖ On va recoder chaque configuration:
- ❖ sur **une seule bande**:



$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$

- ❖ l'état courant entre deux étapes simulées étant $[q, \varepsilon, 0]$
la tête entre deux étapes simulées étant complètement à gauche
- ❖ On étend l'alphabet avec de nouvelles lettres: $\underline{\$}, \#, D, F$
On découvrira au fur et à mesure les états de la nouvelle machine

k bandes I/O \rightarrow une bande

- ❖ On encode chaque configuration sur **une seule bande**:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

- ❖ Stratégie: pour simuler **une** étape de la machine I/O,

k bandes I/O \rightarrow une bande

- ❖ On encode chaque configuration sur **une seule bande**:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

 ↑ ↑ ↑

- ❖ Stratégie: pour simuler **une** étape de la machine I/O,
 - ❖ parcourir toute la bande une fois pour collecter (dans le contrôle) les lettres sous chacune des têtes

k bandes I/O \rightarrow une bande

- ❖ On encode chaque configuration sur **une seule bande**:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

 ↑ ↑ ↑

- ❖ Stratégie: pour simuler **une** étape de la machine I/O,
 - ❖ parcourir toute la bande une fois pour collecter (dans le contrôle) les lettres sous chacune des têtes
 - ❖ parcourir une deuxième fois pour mettre à jour chacune des bandes simulées

k bandes I/O \rightarrow une bande

- ❖ On encode chaque configuration sur **une seule bande**:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

 ↑ ↑ ↑

- ❖ Stratégie: pour simuler **une** étape de la machine I/O,
 - ❖ parcourir toute la bande une fois pour collecter (dans le contrôle) les lettres sous chacune des têtes
 - ❖ parcourir une deuxième fois pour mettre à jour chacune des bandes simulées
- ❖ Difficulté: chaque bande simulée peut **grossir** (d'au plus une case)

Elargir les bandes

❖ On fabrique une (sous-)machine qui, partant de:

$$\underline{w}_1 \# w'_1 D w_2 \# w'_2 D \dots w_{k+2} \# w'_{k+2} D F$$

produit:

$$\underline{w}_1 \# w'_1 \mathbf{B} D w_2 \# w'_2 \mathbf{B} D \dots w_{k+2} \# w'_{k+2} \mathbf{B} D F$$

Elargir les bandes

- ❖ On fabrique une (sous-)machine qui, partant de:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

produit:

$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$

- ❖ (Lemme 6.4.1 du poly, sauf qu'on a $k+2$ bandes au lieu de k)

Elargir les bandes

- ❖ On fabrique une (sous-)machine qui, partant de:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

produit:

$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$

- ❖ (Lemme 6.4.1 du poly, sauf qu'on a $k+2$ bandes au lieu de k)
- ❖ Parcourir l'entrée de gauche à droite jusqu'à rencontrer F , en:

Elargir les bandes

- ❖ On fabrique une (sous-)machine qui, partant de:
$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$
produit:
$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$
- ❖ (Lemme 6.4.1 du poly, sauf qu'on a $k+2$ bandes au lieu de k)
- ❖ Parcourir l'entrée de gauche à droite jusqu'à rencontrer F , en:
 - ❖ insérant B chaque fois qu'on voit un D

Elargir les bandes

- ❖ On fabrique une (sous-)machine qui, partant de:

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

produit:

$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$

- ❖ (Lemme 6.4.1 du poly, sauf qu'on a $k+2$ bandes au lieu de k)
- ❖ Parcourir l'entrée de gauche à droite jusqu'à rencontrer F , en:
 - ❖ insérant B chaque fois qu'on voit un D
 - ❖ **décalant** les sous-chaînes nécessite de **mémoriser** jusqu'à $k+2$ lettres: dans le contrôle (possible car $k+2$ est une **constante**)

Elargir les bandes

- ❖ On fabrique une (sous-)machine qui, partant de:
$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$
produit:
$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$
- ❖ (Lemme 6.4.1 du poly, sauf qu'on a $k+2$ bandes au lieu de k)
- ❖ Parcourir l'entrée de gauche à droite jusqu'à rencontrer F , en:
 - ❖ insérant B chaque fois qu'on voit un D
 - ❖ **décalant** les sous-chaînes nécessite de **mémoriser** jusqu'à $k+2$ lettres: dans le contrôle (possible car $k+2$ est une **constante**)
- ❖ enfin faire revenir la tête complètement à gauche

Elargir les bandes, en animation

$\underline{S}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$



Etat q_ε

Elargir les bandes, en animation

$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$



Etat q_ϵ

On voyage à droite jusqu'à rencontrer D

Elargir les bandes, en animation

$$\delta(q_\varepsilon, a) \stackrel{\text{def}}{=} (q_\varepsilon, a, \rightarrow) \text{ si } a \neq D, F$$

\$ $w_1 \# w'_1 D w_2 \# w'_2 D \dots w_{k+2} \# w'_{k+2} D F$



Etat q_ε

On voyage à droite jusqu'à rencontrer D

Elargir les bandes, en animation

$$\delta(q_\varepsilon, a) \stackrel{\text{def}}{=} (q_\varepsilon, a, \rightarrow) \text{ si } a \neq D, F$$

\$ $\tau w_1 \# \tau w'_1$ **B** $\tau w_2 \# \tau w'_2$ D ... $\tau w_{k+2} \# \tau w'_{k+2}$ D F



Etat q_D

On écrit un B, on mémorise D dans l'état, on va à droite

Elargir les bandes, en animation

$$\delta(q_\varepsilon, a) \stackrel{\text{def}}{=} (q_\varepsilon, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_\varepsilon, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $\tau w_1 \# \tau w'_1$ **B** $\tau w_2 \# \tau w'_2$ D ... $\tau w_{k+2} \# \tau w'_{k+2}$ D F



Etat q_D

On écrit un B, on mémorise D dans l'état, on va à droite

Elargir les bandes, en animation

$$\delta(q_\varepsilon, a) \stackrel{\text{def}}{=} (q_\varepsilon, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_\varepsilon, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1$ **B** $w_2 \# w'_2$ D ... $w_{k+2} \# w'_{k+2}$ D F



Etat q_D

On écrit un B, on mémorise D dans l'état, on va à droite

Passons à la lettre suivante; sur l'exemple, supposons que $w_2 = abc$, $w'_2 = def$

Elargir les bandes, en animation

$$\delta(q_\varepsilon, a) \stackrel{\text{def}}{=} (q_\varepsilon, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_\varepsilon, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1$ **B**abc#defD ... $w_{k+2} \# w'_{k+2}$ DF



Etat q_D

Elargir les bandes, en animation

$$\delta(q_\varepsilon, a) \stackrel{\text{def}}{=} (q_\varepsilon, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_\varepsilon, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1$ **B**abc#defD ... $w_{k+2} \# w'_{k+2}$ DF



Etat q_D

On écrit la lettre qui est dans l'état (D),
et on mémorise la lettre écrasée par l'écriture (a) dans l'état;
puis, à droite

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1$ **B**abc#defD ... $w_{k+2} \# w'_{k+2}$ DF



Etat q_D

On écrit la lettre qui est dans l'état (D),
et on mémorise la lettre écrasée par l'écriture (a) dans l'état;
puis, à droite

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $\tau w_1 \# \tau' w'_1$ **B** **D** $bc \# def D \dots \tau w_{k+2} \# \tau' w'_{k+2} DF$



Etat q_a

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $\tau w_1 \# \tau' w'_1 B D a c \# \text{def} D \dots \tau w_{k+2} \# \tau' w'_{k+2} D F$



Etat q_b

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1 B D a c \# d e f D \dots w_{k+2} \# w'_{k+2} D F$



Etat q_b

et ainsi de suite jusqu'à arriver sur le D suivant

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e D \dots w_{k+2} \# w'_{k+2} D F$



Etat q_f

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_D, B, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e D \dots w_{k+2} \# w'_{k+2} D F$



Etat q_f

on doit maintenant écrire la dernière lettre mémorisée (**f**)
puis un B

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e D \dots w_{k+2} \# w'_{k+2} D F$



Etat q_f

on doit maintenant écrire la dernière lettre mémorisée (**f**)
puis un B

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e D \dots w_{k+2} \# w'_{k+2} D F$



Etat q_f

on doit maintenant écrire la dernière lettre mémorisée (**f**)
puis un B

Imaginons que $w_3 = g h i$, $w'_3 = j k l$

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e D g h i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_f

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e D g h i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_f

comme promis, écrivons la lettre mémorisée (**f**)

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D,B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f g h i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{BD}

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f g h i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{BD}

maintenant on doit écrire B, et mémoriser g

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f$ **B** $h i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{Dg}

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f$ **B** $h i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{Dg}

maintenant on doit écrire D, et mémoriser h

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f B D i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{gh}

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f B D i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{gh}

et ainsi de suite, jusqu'à atteindre le F de fin:

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f B D i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{gh}

et ainsi de suite, jusqu'à atteindre le F de fin:

$$\delta(q_{bw}, F) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f B D i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{gh}

et ainsi de suite, jusqu'à atteindre le F de fin:

$$\delta(q_{bw}, F) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

on n'a plus qu'à écrire tout le mot mémorisé:

$$\delta(q_{bw}, _) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f B D i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{gh}

et ainsi de suite, jusqu'à atteindre le F de fin:

$$\delta(q_{bw}, F) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

on n'a plus qu'à écrire tout le mot mémorisé:

$$\delta(q_{c_{bw}}, _) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

à la fin, il n'y a plus qu'à ramener la tête à gauche:

$$\delta(q_{c_{\varepsilon}}, a) \stackrel{\text{def}}{=} (q_{fini}, F, \leftarrow)$$

$$\delta(q_{fini}, a) \stackrel{\text{def}}{=} (q_{fini}, a, \leftarrow) \text{ si } a \neq \underline{\$}$$

Elargir les bandes, en animation

$$\delta(q_{bw}, a) \stackrel{\text{def}}{=} (q_{wa}, b, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{\varepsilon}, a) \stackrel{\text{def}}{=} (q_{\varepsilon}, a, \rightarrow) \text{ si } a \neq D, F$$

$$\delta(q_{bw}, D) \stackrel{\text{def}}{=} (q_{wBD}, b, \rightarrow) \text{ si } |w| \leq k$$

$$\delta(q_{\varepsilon}, D) \stackrel{\text{def}}{=} (q_{D, B}, \rightarrow)$$

\$ $w_1 \# w'_1 B D a b c \# d e f B D i \# j k l \dots w_{k+2} \# w'_{k+2} D F$



Etat q_{gh}

et ainsi de suite, jusqu'à atteindre le F de fin:

$$\delta(q_{bw}, F) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

on n'a plus qu'à écrire tout le mot mémorisé:

$$\delta(q_{c_{bw}}, _) \stackrel{\text{def}}{=} (q_{cw}, b, \rightarrow)$$

à la fin, il n'y a plus qu'à ramener la tête à gauche:

$$\delta(q_{c_{\varepsilon}}, a) \stackrel{\text{def}}{=} (q_{fini}, F, \leftarrow)$$

$$\delta(q_{fini}, a) \stackrel{\text{def}}{=} (q_{fini}, a, \leftarrow) \text{ si } a \neq \underline{\$}$$

Détails (et preuve formelle) dans le poly.

Elargir les bandes: complexité

On fabrique une (sous-)machine qui, partant de:

produit: $\underline{w}_1 \# w'_1 D w_2 \# w'_2 D \dots w_{k+2} \# w'_{k+2} D F$
 $\underline{w}_1 \# w'_1 B D w_2 \# w'_2 B D \dots w_{k+2} \# w'_{k+2} B D F$

- ❖ Nombre d'étapes de calcul:
2 x (longueur de la bande de fin d'étape + 1)
= 2 x (longueur de la bande de début d'étape + k+3)

k bandes I/O \rightarrow une bande

❖ Pour simuler **une** étape de la machine I/O,

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

k bandes I/O \rightarrow une bande

❖ Pour simuler **une** étape de la machine I/O,

❖ élargir la bande

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$

k bandes I/O \rightarrow une bande

❖ Pour simuler **une** étape de la machine I/O,

❖ élargir la bande

$$\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$$

❖ de gauche à droite,

$$\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$$

collecter les lettres a_i juste à droite des #

[il y en a toujours une!]

$$\underline{\$}w_1\#\overbrace{a_1}^{w'_1}Dw_2\#\overbrace{a_2}^{w'_2}D \dots w_{k+2}\#\overbrace{a_{k+2}}^{w'_{k+2}}DF$$

k bandes I/O \rightarrow une bande

❖ Pour simuler **une** étape de la machine I/O,

- ❖ élargir la bande $\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$
- ❖ de gauche à droite, $\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$
 collecter les lettres a_i juste à droite des #
 [il y en a toujours une!]
 $\underline{\$}w_1\#\overbrace{a_1}^{w'_1}Dw_2\#\overbrace{a_2}^{w'_2}D \dots w_{k+2}\#\overbrace{a_{k+2}}^{w'_{k+2}}DF$
- ❖ calculer $\delta(q, a_1, \dots, a_{k+1}) = (q', (b_2, \dots, b_{k+2}), (dir_1, dir_2, \dots, dir_{k+2}))$
 [dans le contrôle; pour l'exemple, prenons $dir_1 = \rightarrow, dir_2 = \rightarrow, \dots, dir_{k+2} = \downarrow$]

k bandes I/O \rightarrow une bande

❖ Pour simuler **une** étape de la machine I/O,

❖ élargir la bande $\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$

❖ de gauche à droite, $\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$
 collecter les lettres a_i juste à droite des #
 [il y en a toujours une!]

$$\underline{\$}w_1\#\overbrace{a_1}^{w'_1}Dw_2\#\overbrace{a_2}^{w'_2}D \dots w_{k+2}\#\overbrace{a_{k+2}}^{w'_{k+2}}DF$$

❖ calculer $\delta(q, a_1, \dots, a_{k+1}) = (q', (b_2, \dots, b_{k+2}), (dir_1, dir_2, \dots, dir_{k+2}))$
 [dans le contrôle; pour l'exemple, prenons $dir_1=\rightarrow, dir_2=\rightarrow, \dots, dir_{k+2}=\downarrow$]

❖ de droite à gauche, $\underline{\$}w_1\mathbf{a}_1\#\text{---}Dw_2\mathbf{b}_2\#\text{---}D \dots w_{k+2}\#\mathbf{b}_{k+2}\text{---}DF$

faire les remplacements adéquats, en bougeant les # de ≤ 1 case

k bandes I/O \rightarrow une bande

❖ Pour simuler **une** étape de la machine I/O,

❖ élargir la bande $\underline{\$}w_1\#w'_1Dw_2\#w'_2D \dots w_{k+2}\#w'_{k+2}DF$

❖ de gauche à droite, $\underline{\$}w_1\#w'_1\mathbf{B}Dw_2\#w'_2\mathbf{B}D \dots w_{k+2}\#w'_{k+2}\mathbf{B}DF$
 collecter les lettres a_i juste à droite des #
 [il y en a toujours une!]

$$\underline{\$}w_1\#\overbrace{a_1}^{w'_1}Dw_2\#\overbrace{a_2}^{w'_2}D \dots w_{k+2}\#\overbrace{a_{k+2}}^{w'_{k+2}}DF$$

❖ calculer $\delta(q, a_1, \dots, a_{k+1}) = (q', (b_2, \dots, b_{k+2}), (dir_1, dir_2, \dots, dir_{k+2}))$
 [dans le contrôle; pour l'exemple, prenons $dir_1 = \rightarrow, dir_2 = \rightarrow, \dots, dir_{k+2} = \downarrow$]

❖ de droite à gauche, $\underline{\$}w_1\mathbf{a}_1\#\text{---}Dw_2\mathbf{b}_2\#\text{---}D \dots w_{k+2}\#\mathbf{b}_{k+2}\text{---}DF$

faire les remplacements adéquats, en bougeant les # de ≤ 1 case

❖ Détails dans le poly (Théorème 6.4.2)

k bandes I/O \rightarrow une bande: complexité

- ❖ Si la machine M à k bandes I/O s'exécute en **temps** $f(n)$

k bandes I/O \rightarrow une bande: complexité

- ❖ Si la machine M à k bandes I/O s'exécute en **temps** $f(n)$
 - ❖ la machine que nous avons construite s'exécute en temps
$$4(n+(k+2)+1) \text{ [étape 1]} + 4(n+2(k+2)+1) \text{ [étape 2]} \\ + \dots + 4(n+f(n)(k+2)+1) \text{ [étape } f(n)\text{]} \\ = 4(n.f(n)) + 2(k+2).f(n).(f(n)+1) + 4f(n)$$

k bandes I/O \rightarrow une bande: complexité

- ❖ Si la machine M à k bandes I/O s'exécute en **temps** $f(n)$
 - ❖ la machine que nous avons construite s'exécute en temps
$$4(n+(k+2)+1) \text{ [étape 1]} + 4(n+2(k+2)+1) \text{ [étape 2]} \\ + \dots + 4(n+f(n)(k+2)+1) \text{ [étape } f(n)\text{]} \\ = 4(n.f(n)) + 2(k+2).f(n).(f(n)+1) + 4f(n)$$
 - ❖ Notamment, si $f(n)$ est surlinéaire ($f(n) \geq n$ pour tout n), la simulation ne prend qu'un **temps** $O(f(n)^2)$

k bandes I/O \rightarrow une bande: complexité

- ❖ Si la machine M à k bandes I/O s'exécute en **temps** $f(n)$
 - ❖ la machine que nous avons construite s'exécute en temps
$$4(n+(k+2)+1) \text{ [étape 1]} + 4(n+2(k+2)+1) \text{ [étape 2]} \\ + \dots + 4(n+f(n)(k+2)+1) \text{ [étape } f(n)\text{]} \\ = 4(n.f(n)) + 2(k+2).f(n).(f(n)+1) + 4f(n)$$
 - ❖ Notamment, si $f(n)$ est surlinéaire ($f(n) \geq n$ pour tout n), la simulation ne prend qu'un **temps** $O(f(n)^2)$
- ❖ Si la machine M à k bandes I/O s'exécute en **espace** $g(n)$... un codage plus malin (qui n'étend la bande que si nécessaire) donnerait une simulation en **espace** $g(n)+cste$

Composition

Composition

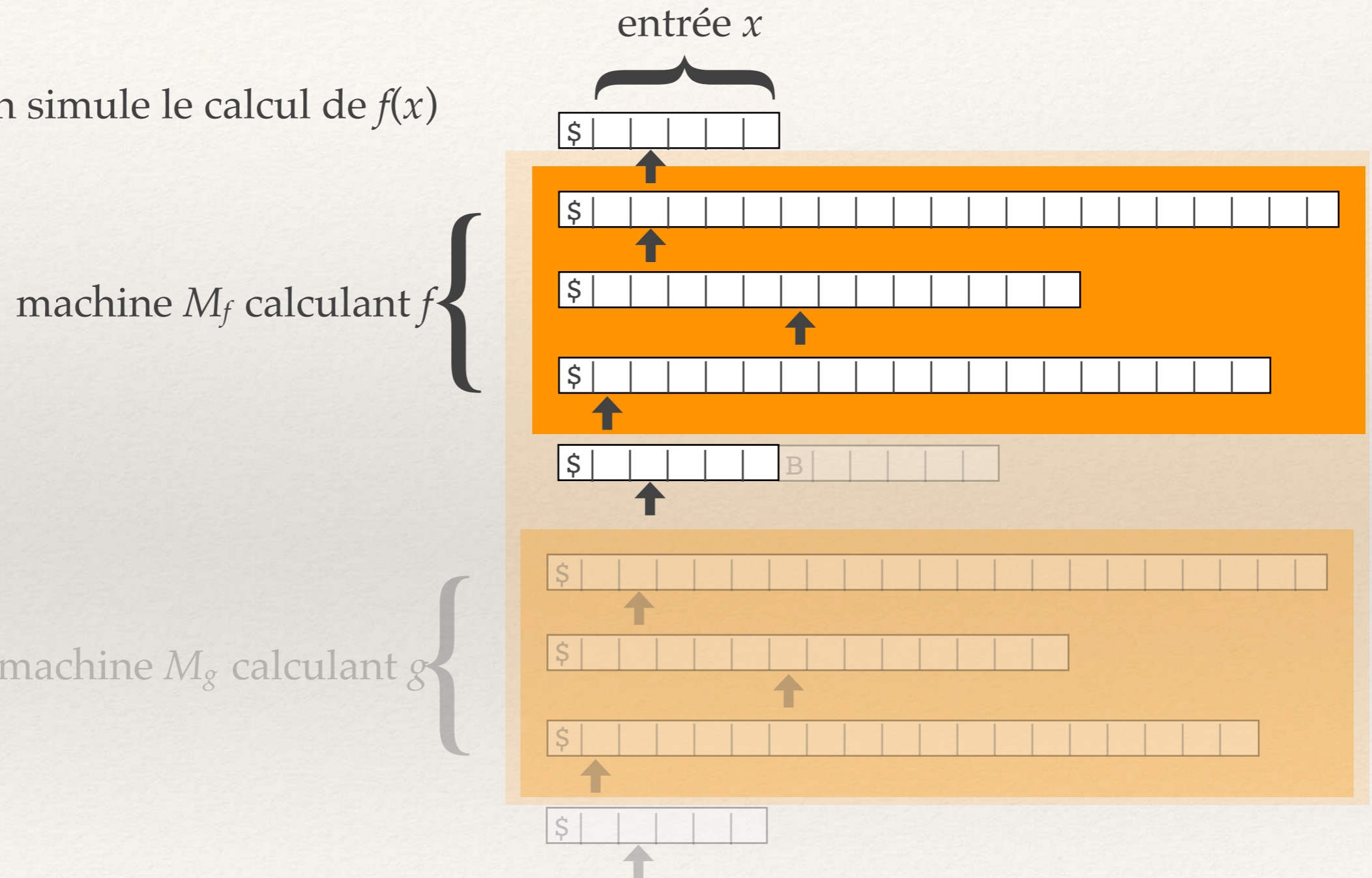
- ❖ Je vous avais dit que la composée de deux fonctions (semi-)calculables est (semi-)calculable... mais que la preuve du poly est erronée; voici une réparation
- ❖ On peut supposer que f et g sont calculées par des machines I/O à k bandes, et...

Composition

- ❖ Je vous avais dit que la composée de deux fonctions (semi-)calculables est (semi-)calculable... mais que la preuve du poly est erronée; voici une réparation
- ❖ On peut supposer que f et g sont calculées par des machines I/O à k bandes, et...
- ❖ ces machines ne vont jamais à droite du premier B sur leur bande d'entrée

Composition

1ère phase: on simule le calcul de $f(x)$



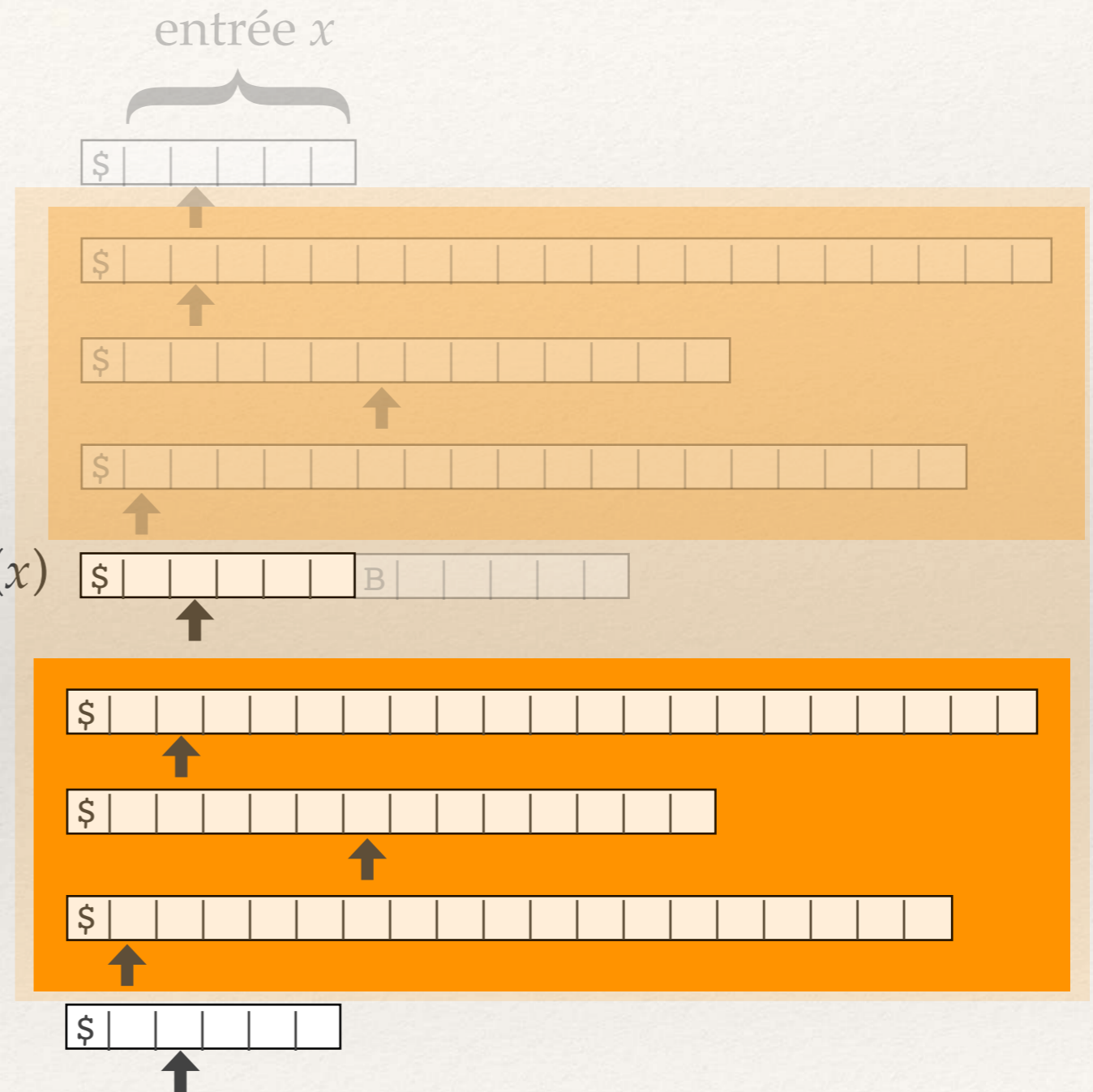
Composition

1ère phase: on simule le calcul de $f(x)$

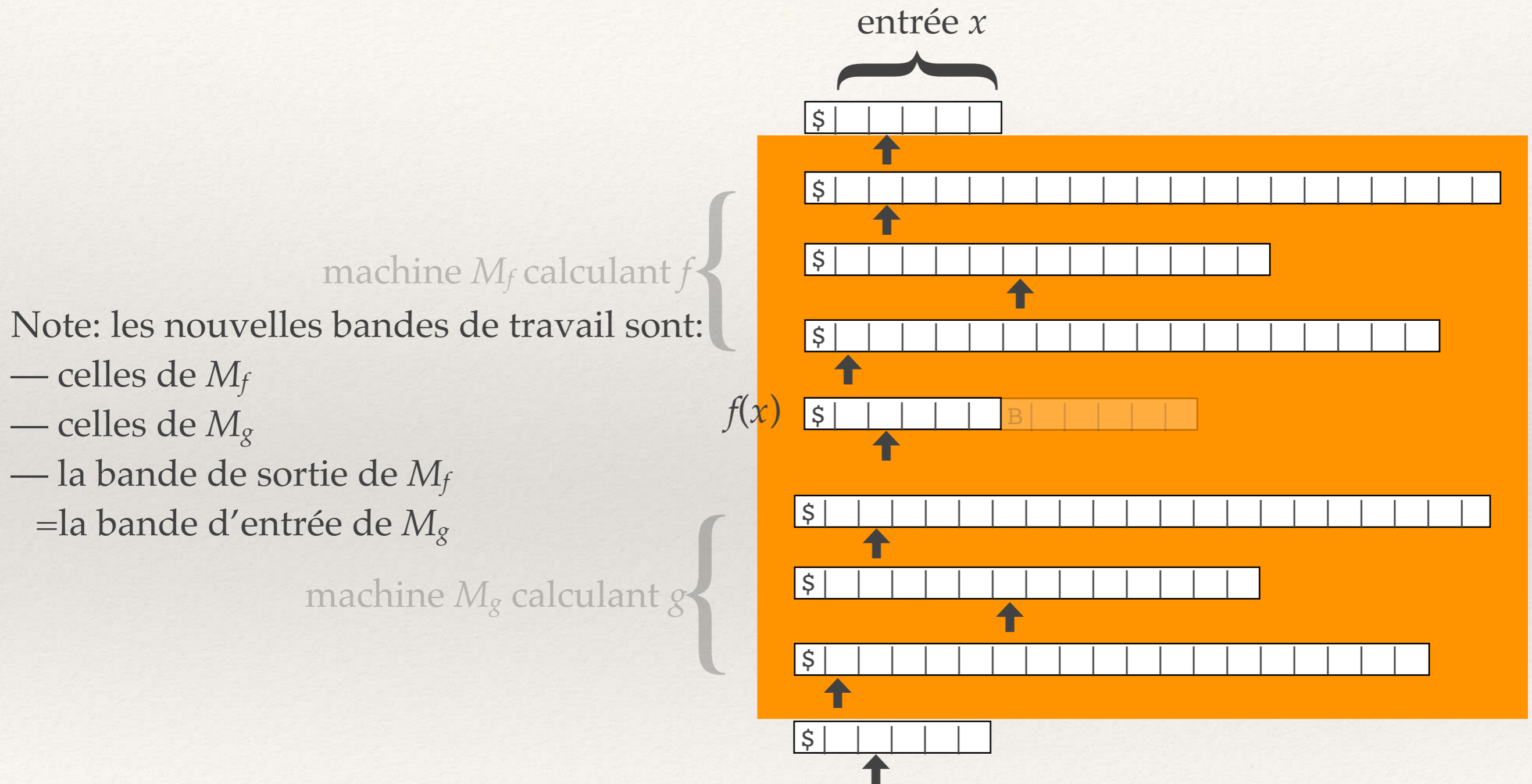
machine M_f calculant f

2ème phase: on calcule g sur $f(x)$

machine M_g calculant g



Composition



Pouvoir expressif: un aperçu

Le langage IMP

Commandes

$c ::=$	$x := e$	affectation
	skip	ne rien faire
	$c_1; c_2$	séquence
	if e then c_1 else c_2	conditionnelle
	while e do c	boucle while

Expressions

$e ::=$	x	variables
	n	constante entière ($n \in \mathbb{Z}$)
	$e + e$	addition
	$-e$	opposé

- ❖ (Un langage que nous reverrons en Prog 1, 2ème partie)
- ❖ Nous allons traduire chaque programme IMP
en une machine de Turing à k bandes de travail
(sketch)

Le langage IMP

Commandes

$c ::=$	$x := e$	affectation
	skip	ne rien faire
	$c_1; c_2$	séquence
	if e then c_1 else c_2	conditionnelle
	while e do c	boucle while

Expressions

$e ::=$	x	variables
	n	constante entière ($n \in \mathbb{Z}$)
	$e + e$	addition
	$-e$	opposé

Sémantique: si la valeur de e est
non nulle
alors faire c_1 sinon c_2

- ❖ (Un langage que nous reverrons en Prog 1, 2ème partie)
- ❖ Nous allons traduire chaque programme IMP
en une machine de Turing à k bandes de travail
(sketch)

Simplifications (1/2)

Commandes		Expressions
$c ::= x := e$	affectation	$e ::= x$ variables
skip	ne rien faire	n constante entière ($n \in \mathbb{Z}$)
$c_1; c_2$	séquence	$e + e$ addition
if e then c_1 else c_2	conditionnelle	$-e$ opposé
while e do c	boucle while	

- ❖ On demande à ce que les affectations soient de la forme

$x := y$

ou $x := \text{constante}$

ou $x := y + z$ (avec x, y, z deux à deux distinctes)

ou $x := -y$ (avec x, y distinctes)

- ❖ quitte à utiliser des variables supplémentaires

Simplifications (1/2)

Commandes		Expressions
$c ::= x := e$	affectation	$e ::= x$ variables
skip	ne rien faire	n constante entière ($n \in \mathbb{Z}$)
$c_1; c_2$	séquence	$e + e$ addition
if e then c_1 else c_2	conditionnelle	$-e$ opposé
while e do c	boucle while	

- ❖ On demande à ce que les affectations soient de la forme

$$x := y$$

ou $x := \text{constante}$

ou $x := y + z$ (avec x, y, z deux à deux distinctes)

ou $x := -y$ (avec x, y distinctes)

- ❖ quitte à utiliser des variables supplémentaires

- ❖ Exemple: $x := y + y + (-x) \rightarrow y_1 := y; y_2 := y; x_1 := y_1 + y_2; x_2 := -x; x := x_1 + x_2$

Simplifications (2/2)

Commandes

$c ::= x := e$

| skip

| $c_1; c_2$

| if e then c_1 else c_2

| while e do c

affectation

ne rien faire

séquence

conditionnelle

boucle while

Expressions

$e ::= x$ variables

| n constante entière ($n \in \mathbb{Z}$)

| $e + e$ addition

| $-e$ opposé

- ❖ Dans **if e then c_1 else c_2** et dans **while e do c** , on demande que e soit une **variable**
- ❖ quitte à utiliser des variables supplémentaires

Simplifications (2/2)

Commandes

$c ::= x := e$

| skip

| $c_1; c_2$

| if e then c_1 else c_2

| while e do c

affectation

ne rien faire

séquence

conditionnelle

boucle while

Expressions

$e ::= x$ variables

| n constante entière ($n \in \mathbb{Z}$)

| $e + e$ addition

| $-e$ opposé

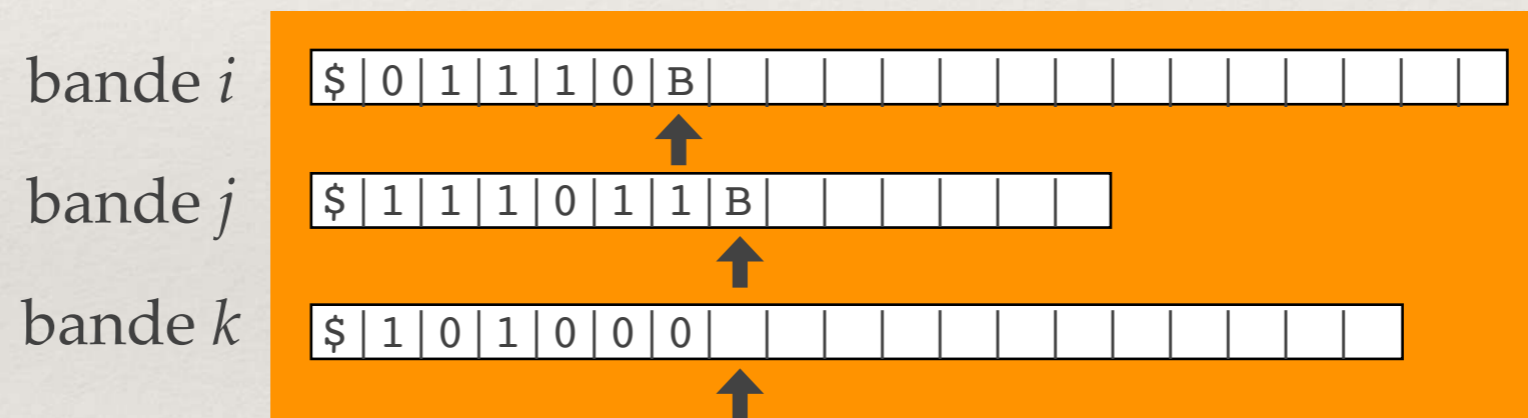
- ❖ Dans `if e then c_1 else c_2` et dans `while e do c` , on demande que e soit une **variable**
- ❖ quitte à utiliser des variables supplémentaires
- ❖ Exemple: `if $x+y+(-3)$ then c_1 else c_2`
→ `$z_1 := x+y; z_2 := -3; z := z_1+z_2; \text{if } z \text{ then } c_1 \text{ else } c_2$`

Traduction

- ❖ Etant donnée une commande c_0 , où apparaissent $k+2$ variables:
 - x_0 (entrée)
 - x_1, \dots, x_k
 - x_{k+1} (sortie)on crée une machine I/O à k bandes
- ❖ On traduit chaque sous-commande $c...$

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

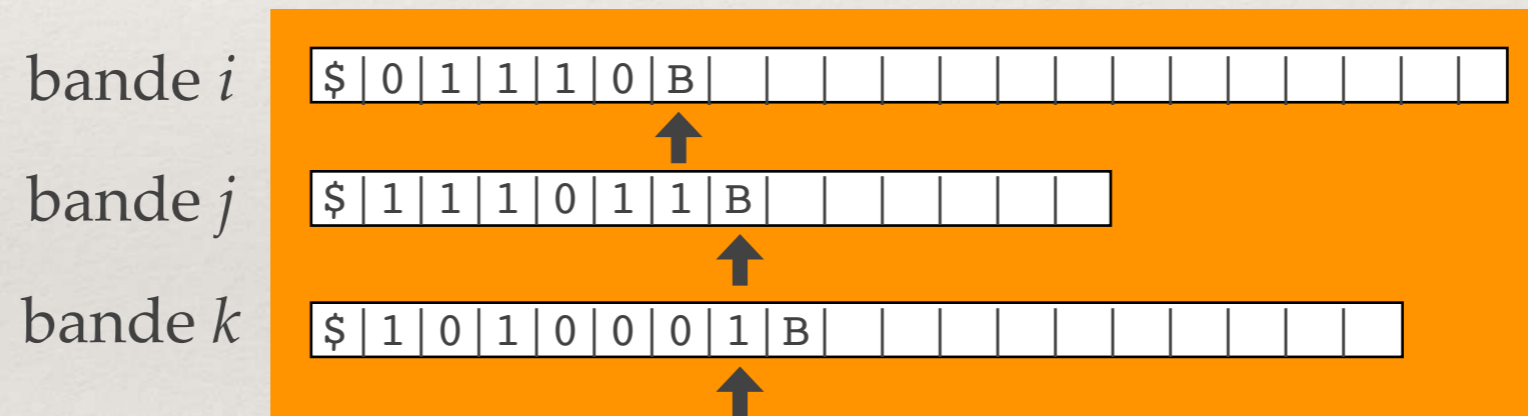


Etat « retenue »

Note: un B est vu comme un 0, sauf que la tête n'en bouge plus

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

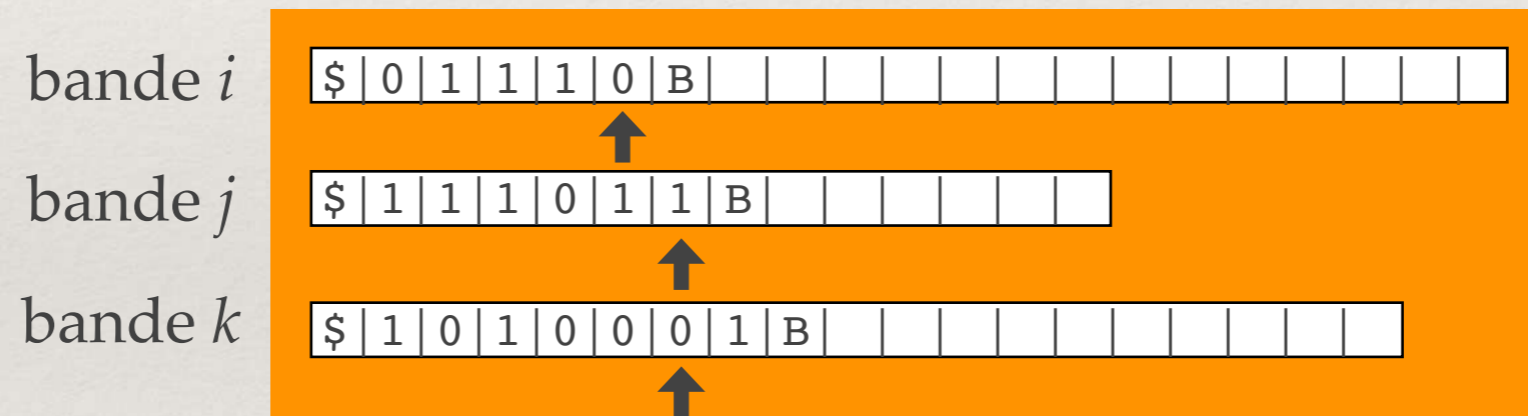


Etat « nettoyage »

Dans l'état « nettoyage »,
on ramène les trois têtes
en début de bande

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

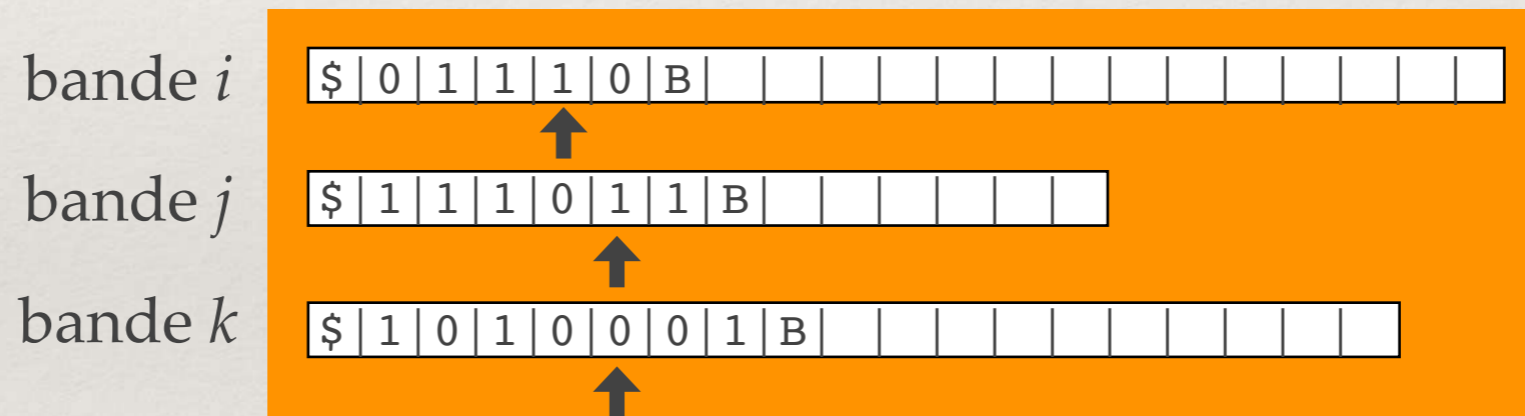


Etat « nettoyage »

Dans l'état « nettoyage »,
on ramène les trois têtes
en début de bande

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

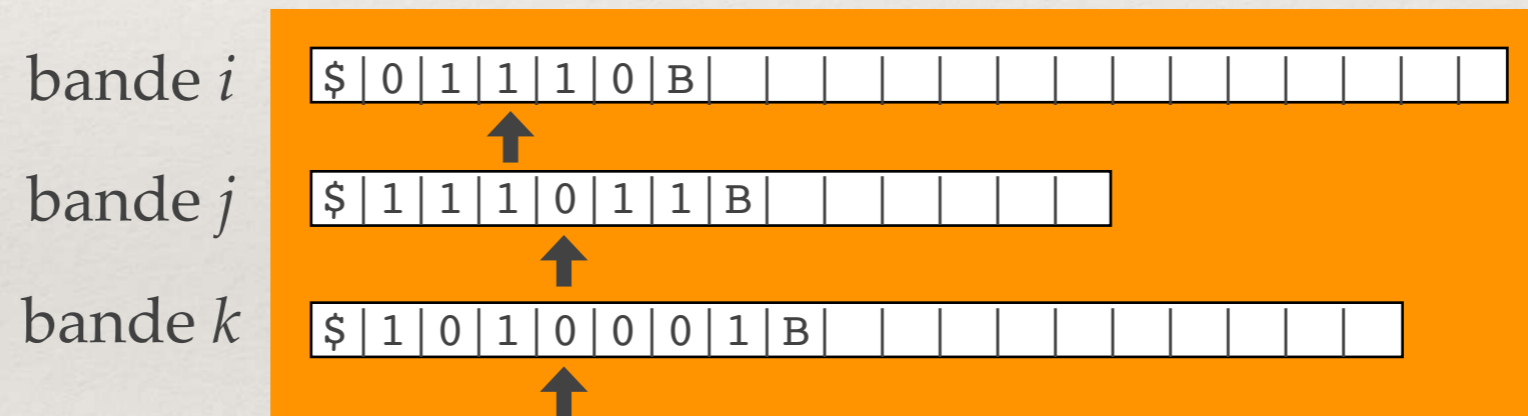


Etat « nettoyage »

Dans l'état « nettoyage »,
on ramène les trois têtes
en début de bande

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

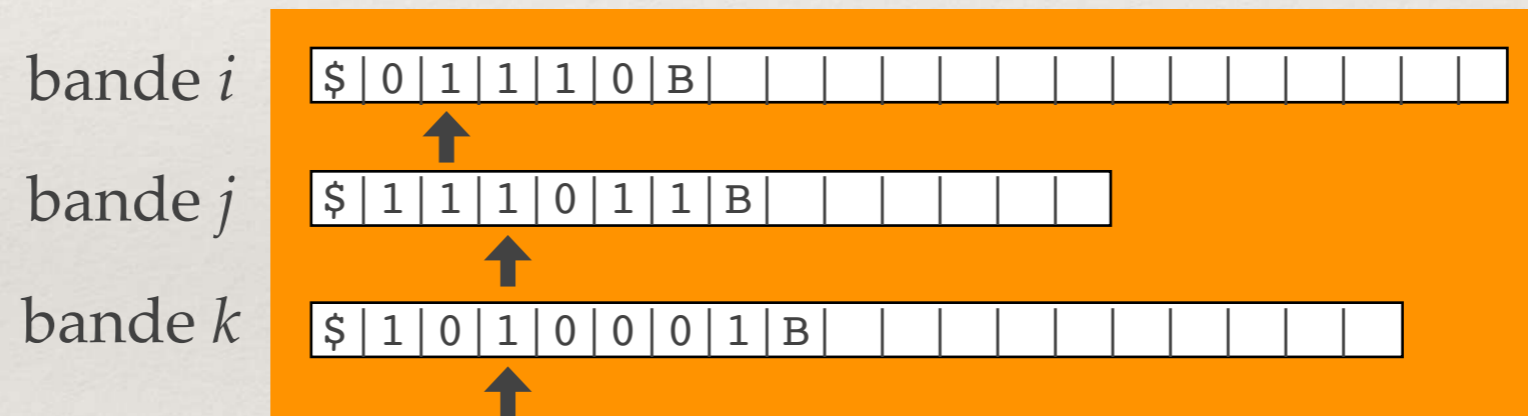


Etat « nettoyage »

Dans l'état « nettoyage »,
on ramène les trois têtes
en début de bande

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

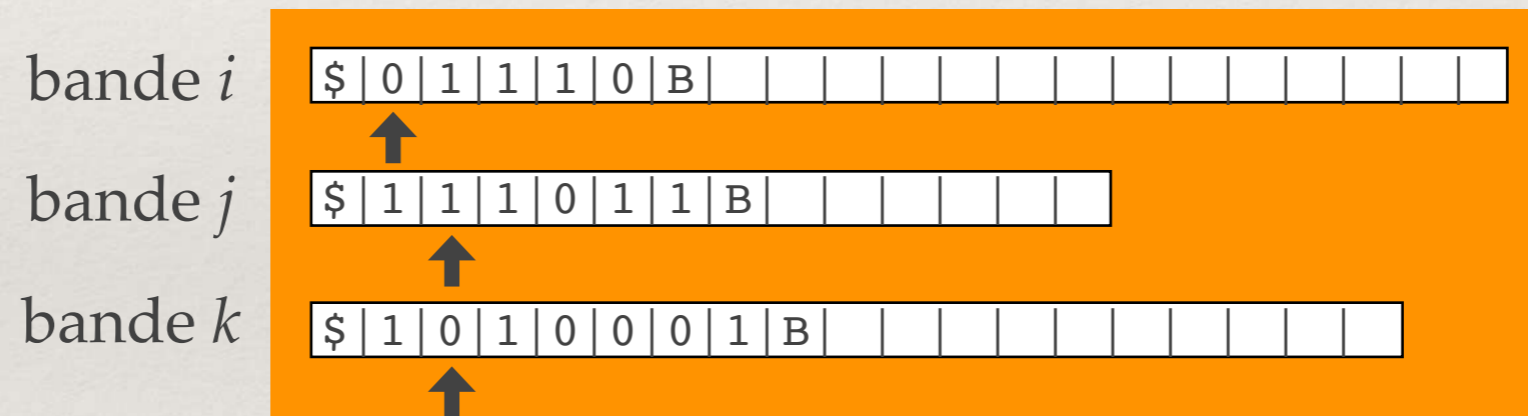


Etat « nettoyage »

Dans l'état « nettoyage »,
on ramène les trois têtes
en début de bande

Traduction de $x := y + z$

- ❖ $x_i := x_j + x_k$, avec x_i, x_j, x_k deux à deux distinctes
- ❖ On implémente l'addition sur trois bandes de travail
- ❖ par l'algorithme usuel d'addition avec retenue

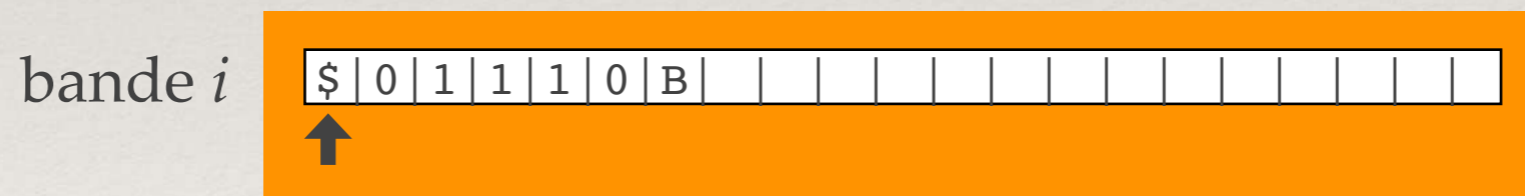


Etat « nettoyage »

Dans l'état « nettoyage »,
on ramène les trois têtes
en début de bande

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:

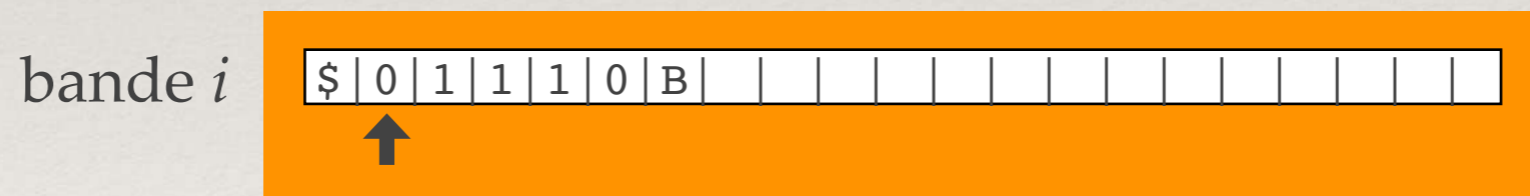


Etat « début »

Dans l'état « début »,
on teste si la bande i contient
le nombre zéro, i.e. un mot
dans $\$0^*B$

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:

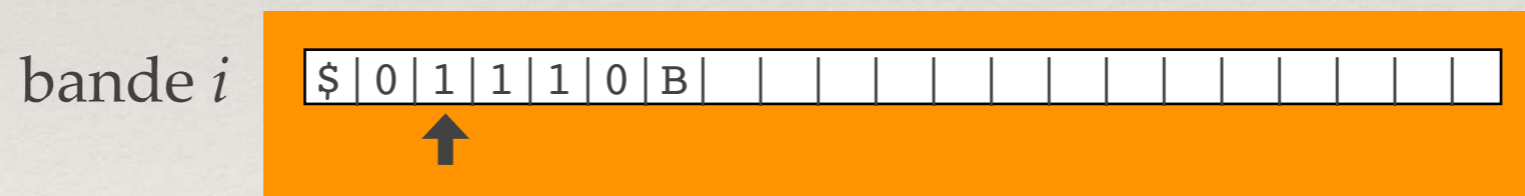


Etat « début »

Dans l'état « début »,
on teste si la bande i contient
le nombre zéro, i.e. un mot
dans $\$0^*B$

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:

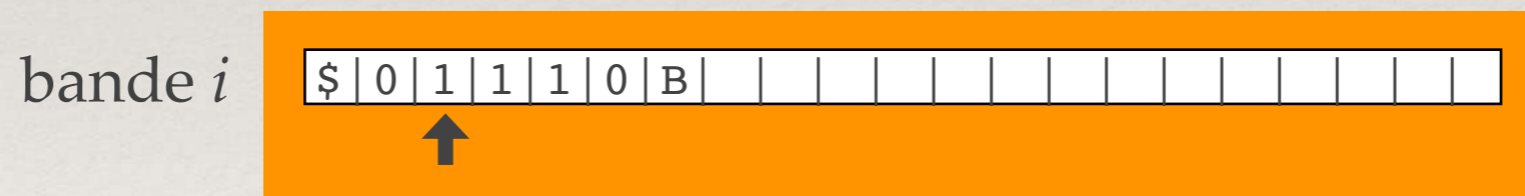


Etat « début »

Dans l'état « début »,
on teste si la bande i contient
le nombre zéro, i.e. un mot
dans $\$0^*B$

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:

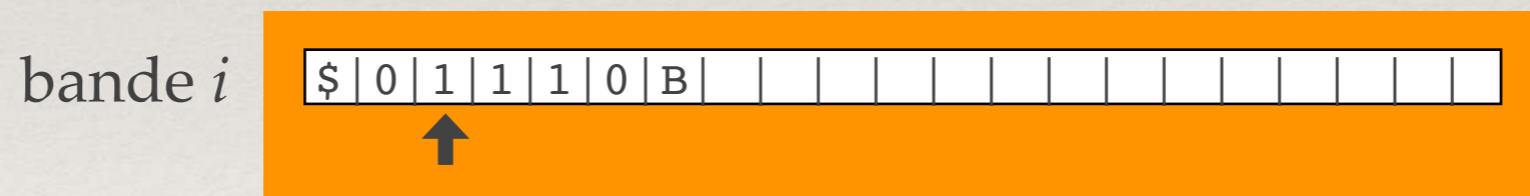


Etat « nettoyage 1 »

La bande i ne contient pas zéro:
on démarre donc l'exécution de c_1

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:



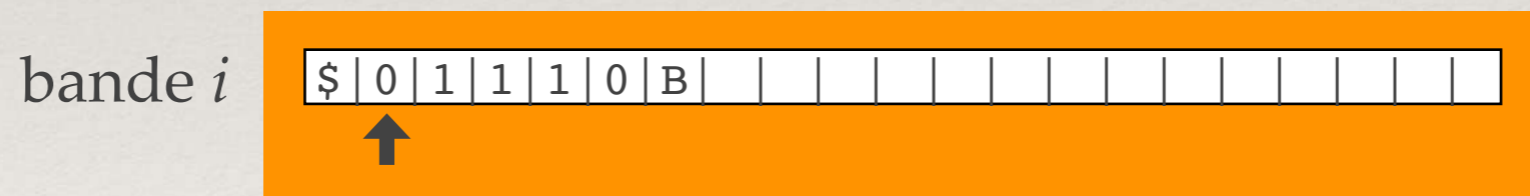
Etat « nettoyage 1 »

La bande i ne contient pas zéro:
on démarre donc l'exécution de c_1

... mais avant, on nettoie: on ramène la tête au début de la bande i

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:



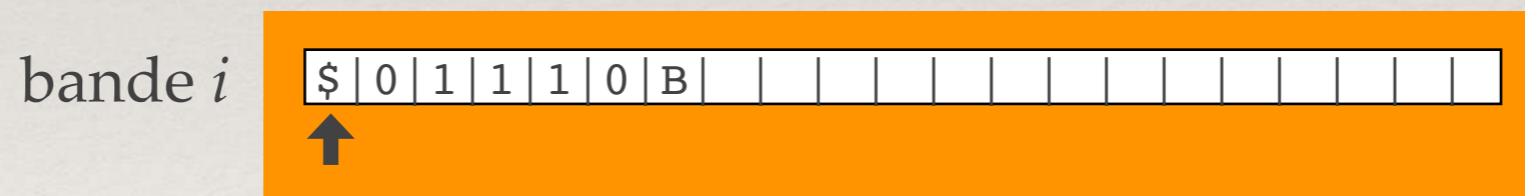
Etat « nettoyage 1 »

La bande i ne contient pas zéro:
on démarre donc l'exécution de c_1

... mais avant, on nettoie: on ramène la tête au début de la bande i

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:



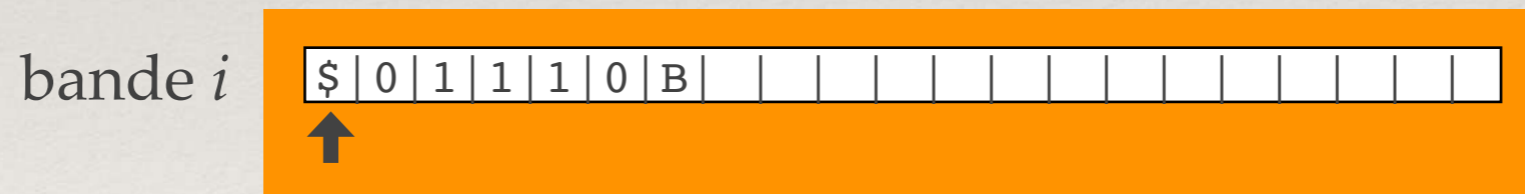
Etat « nettoyage 1 »

La bande i ne contient pas zéro:
on démarre donc l'exécution de c_1

... mais avant, on nettoie: on ramène la tête au début de la bande i

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:

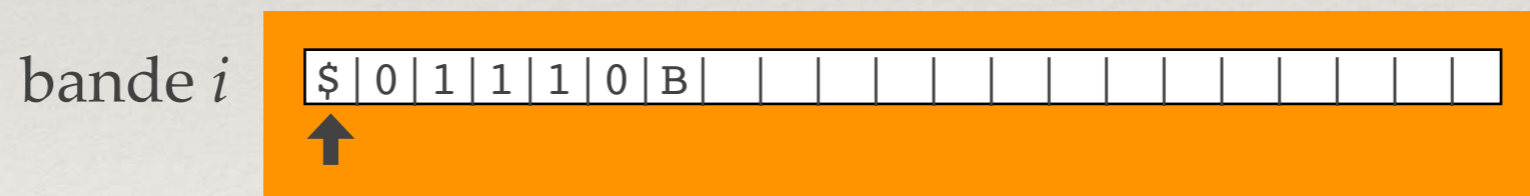


Etat « début 1 »

Nettoyage fini: on démarre c_1

Traduction de $\text{if } x_i \text{ then } c_1 \text{ else } c_2$

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:



Etat « début 1 »

Nettoyage fini: on démarre c_1

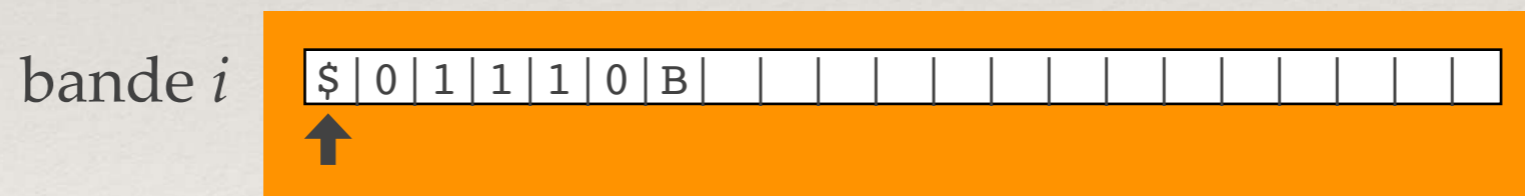
Si à la place on avait atteint un B en restant dans l'état « début », on serait ensuite allé en « début 2 » après nettoyage

Traduction de `if x_i then c_1 else c_2`

- ❖ Par hypothèse de récurrence, on sait traduire
 - c_1 en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
 - c_2 en une fonction de transition, de sorte que le calcul commence en un état « début 2 » et finisse en un état « fin 2 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:
- ❖ ... finalement, on déclare des transitions
 - de « fin 1 » vers « fin »
 - et de « fin 2 » vers « fin »,qui ne modifient pas les bandes et laissent les têtes immobiles

Traduction de $\text{while } x_i \text{ do } c$

- ❖ Par hypothèse de récurrence, on sait traduire
— c en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:

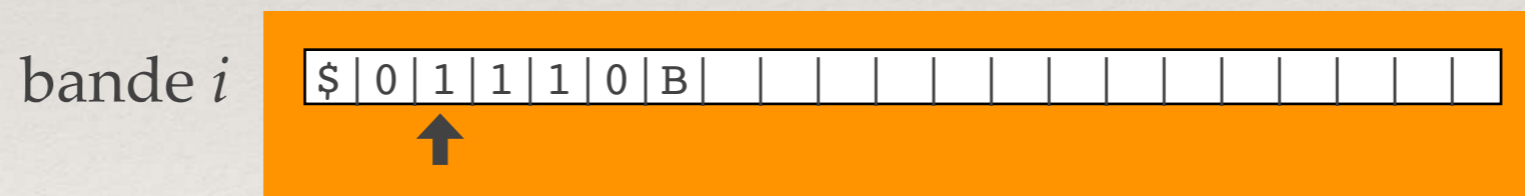


Etat « début »

Dans l'état « début »,
on teste si la bande i contient
le nombre zéro, i.e. un mot
dans $\$0^*B$, comme pour le `if`

Traduction de $\text{while } x_i \text{ do } c$

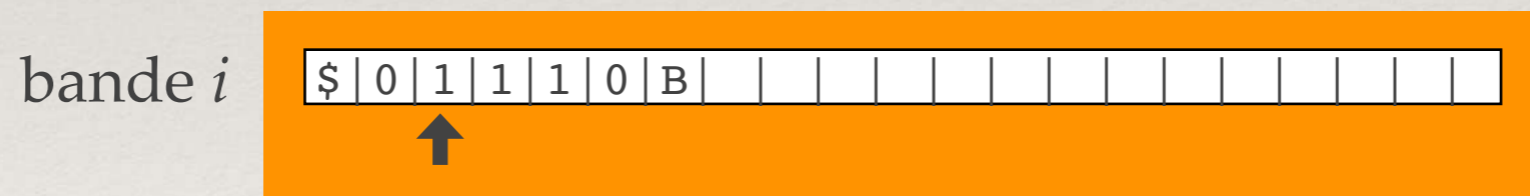
- ❖ Par hypothèse de récurrence, on sait traduire
 - c en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:



- Si la bande i contient au moins un 1, on va en « début 1 » après nettoyage
- Si elle ne contient pas de 1, on va en « fin » après nettoyage

Traduction de `while x_i do c`

- ❖ Par hypothèse de récurrence, on sait traduire
 - c en une fonction de transition, de sorte que le calcul commence en un état « début 1 » et finisse en un état « fin 1 »
- ❖ On crée deux états frais « début » et « fin »
- ❖ On enrichit la fonction de transition:



- Si la bande i contient au moins un 1, on va en « début 1 » après nettoyage
 - Si elle ne contient pas de 1, on va en « fin » après nettoyage
- Boucle:** ne pas oublier les transitions de « fin 1 » vers « début »!

Autres constructions

- ❖ Autres constructions: exercice!
- ❖ Note: ceci est un **sketch** de construction
- ❖ Notamment, je n'ai effectué **aucune preuve**

Intersection, union de langages

Union de langages r.e.

- ❖ On va montrer que l'union L d'un nombre fini de langages r.e. L_1, \dots, L_m est r.e.

Union de langages r.e.

- ❖ On va montrer que l'union L d'un nombre fini de langages r.e. L_1, \dots, L_m est r.e.
- ❖ Par hypothèse, pour tout i on a: $L_i = L(M_i)$ pour une certaine machine de Turing M_i , **à une bande**

Union de langages r.e.

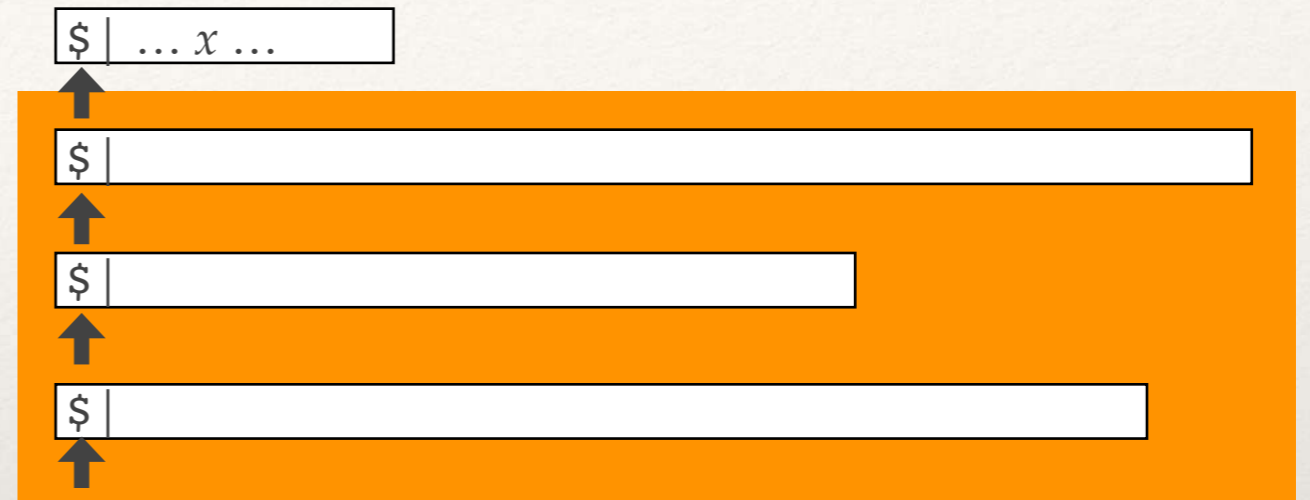
- ❖ On va montrer que l'union L d'un nombre fini de langages r.e. L_1, \dots, L_m est r.e.
- ❖ Par hypothèse, pour tout i on a: $L_i = L(M_i)$ pour une certaine machine de Turing M_i , **à une bande**
- ❖ Note: la difficulté est qu'on ne peut pas simuler M_1, \dots, M_m **en séquence** (que se passe-t-il si M_1 ne termine pas?)

Union de langages r.e.

- ❖ On va montrer que l'union L d'un nombre fini de langages r.e. L_1, \dots, L_m est r.e.
- ❖ Par hypothèse, pour tout i on a: $L_i = L(M_i)$ pour une certaine machine de Turing M_i , **à une bande**
- ❖ Note: la difficulté est qu'on ne peut pas simuler M_1, \dots, M_m **en séquence** (que se passe-t-il si M_1 ne termine pas?)
- ❖ On va donc les simuler **en parallèle**, via une machine **à m bandes I/O** acceptant L (= dont le langage est L)

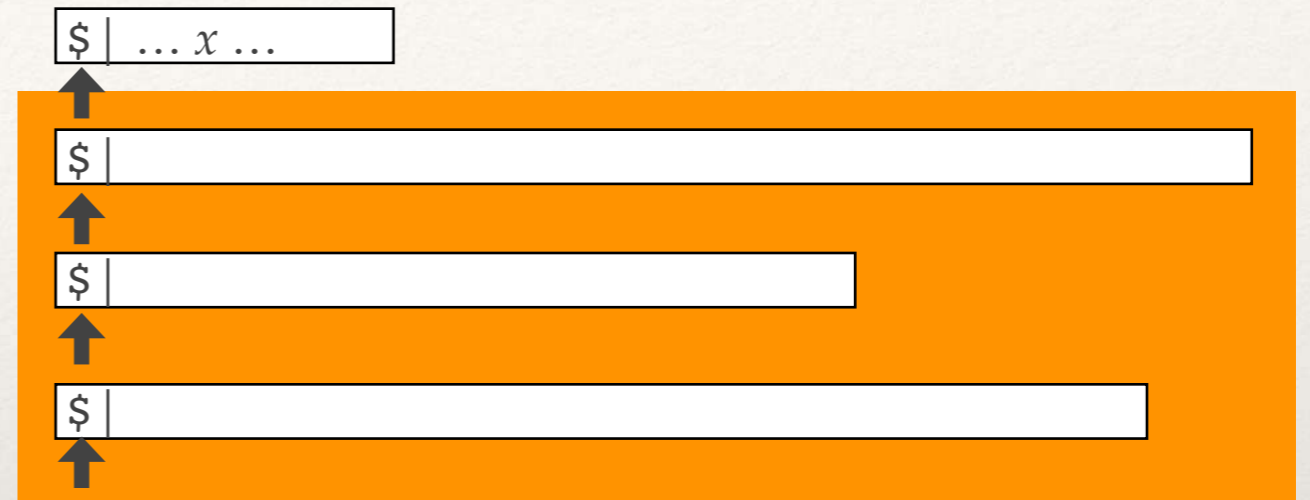
Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i



Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i

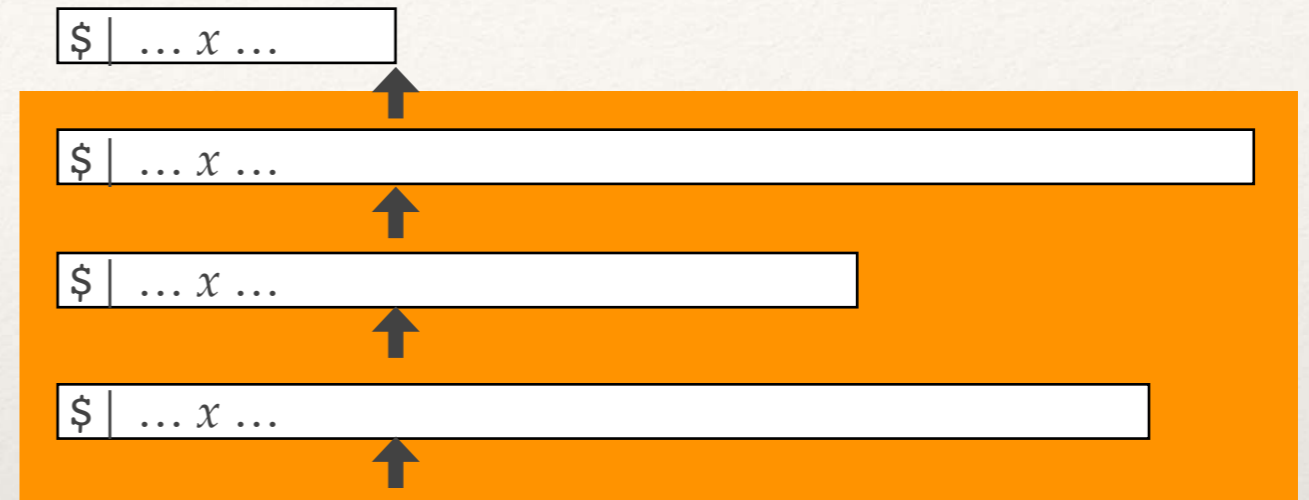


$$\delta(q_0, (a, a_1, \dots, a_m)) = (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$$$

On commence par recopier x sur toutes les bandes de travail en parallèle

Simulation parallèle

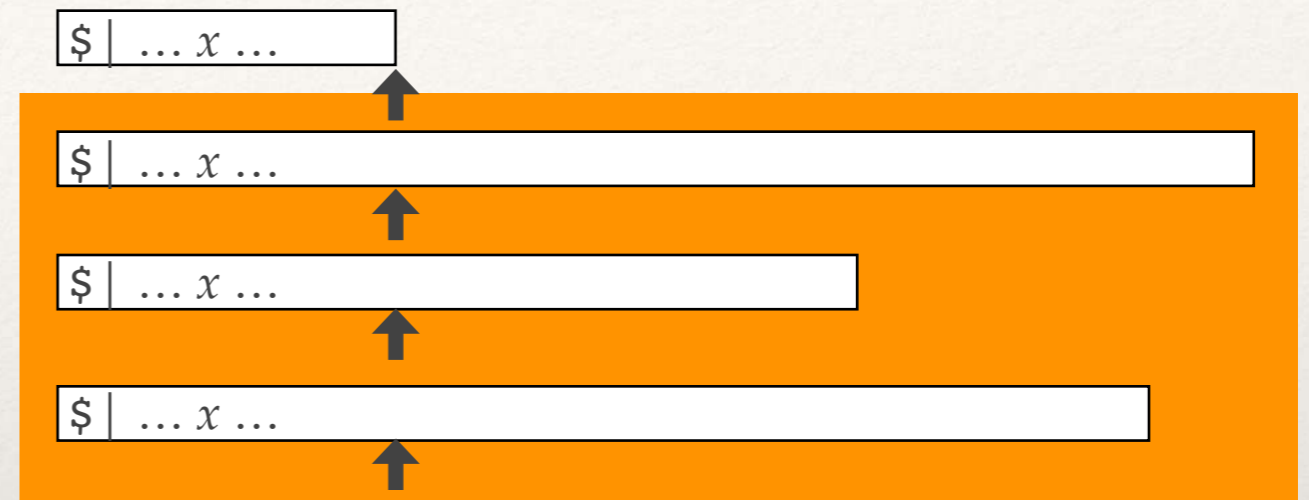
- ❖ La bande i va servir à simuler la machine M_i



$$\delta(q_0, (a, a_1, \dots, a_m)) = (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$$$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i

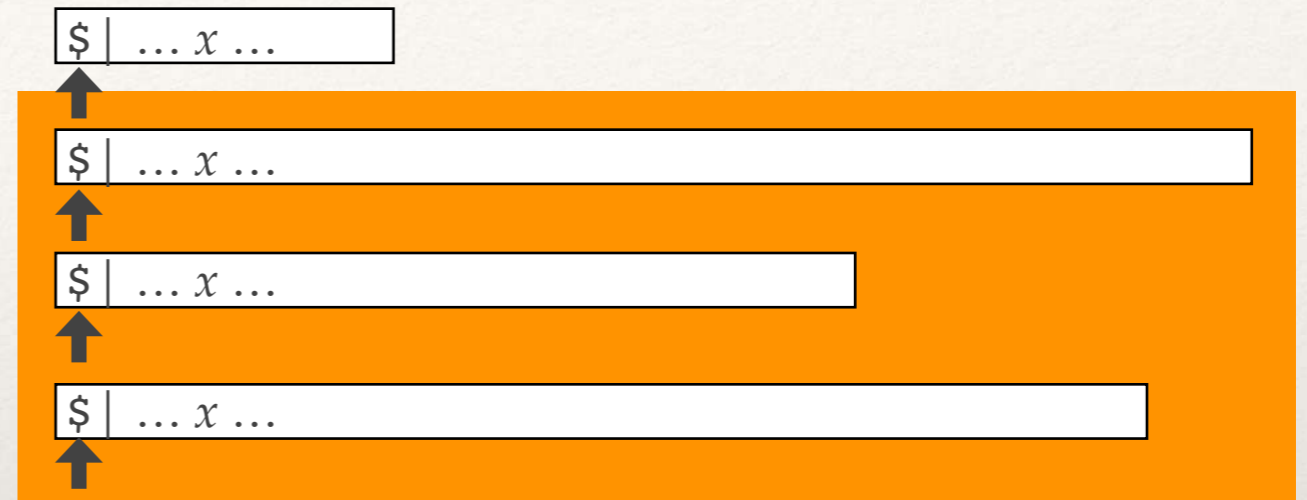


$$\begin{aligned}\delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$\end{aligned}$$

On ramène toutes les têtes au début de leurs bandes

Simulation parallèle

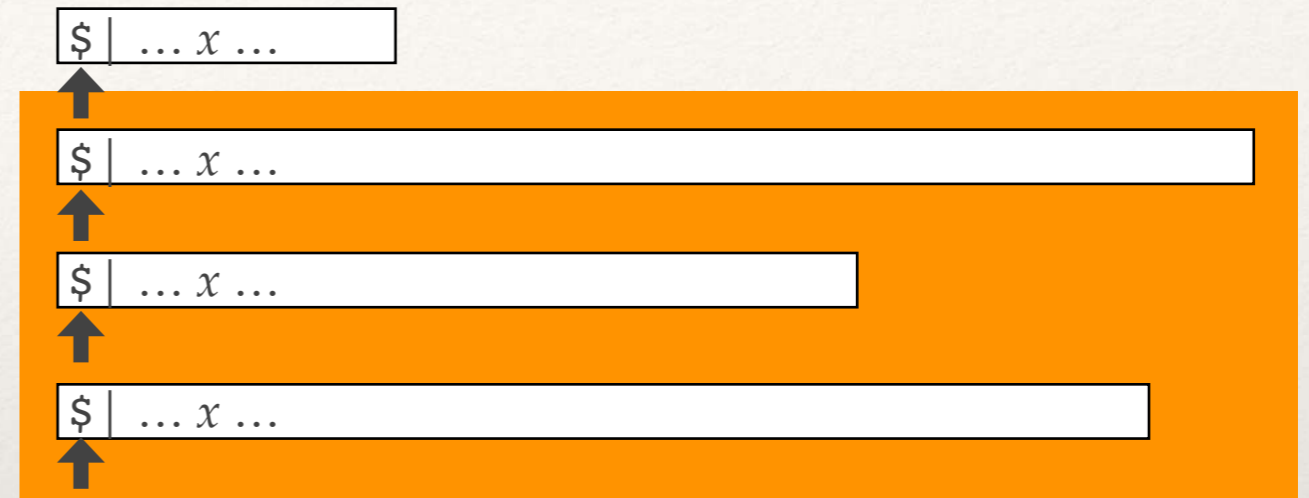
- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i ,
 $init[i]$ son état initial



$$\begin{aligned}\delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$\end{aligned}$$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $init[i]$ son état initial



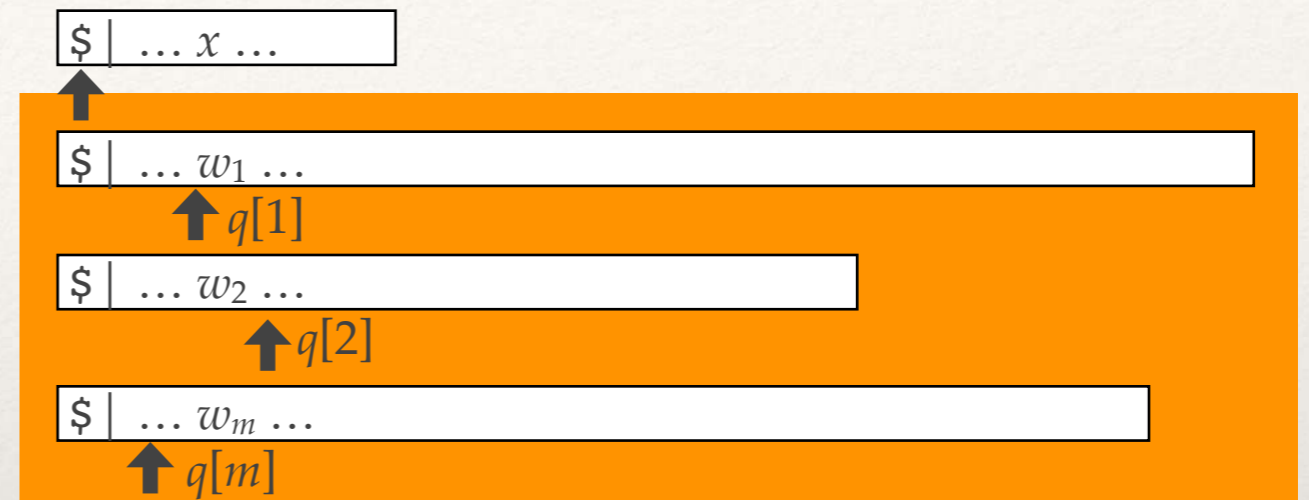
$$\begin{aligned} \delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$ \\ \delta(q_1, (\$, a_1, \dots, a_m)) &= ((init[1], \dots, init[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \end{aligned}$$

On simule une étape de chaque M_i en parallèle:

Les états de notre machine sont des m -uplets $(q[1], \dots, q[m])$ d'états de chaque M_i

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $\text{init}[i]$ son état initial
- ❖ et **accept** $[i]$, **reject** $[i]$ leurs états finaux



$$\delta(q_0, (a, a_1, \dots, a_m)) = (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$$$

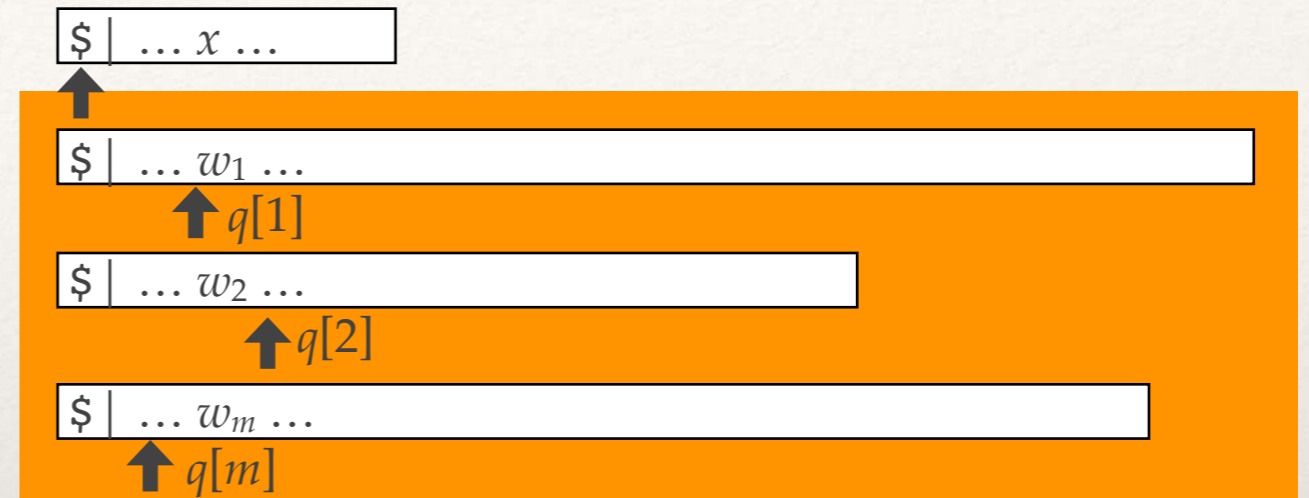
$$\delta(q_0, (B, a_1, \dots, a_m)) = (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow))$$

$$\delta(q_1, (a, a_1, \dots, a_m)) = (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$$$

$$\delta(q_1, (\$, a_1, \dots, a_m)) = ((\text{init}[1], \dots, \text{init}[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow))$$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $\text{init}[i]$ son état initial
- ❖ et $\text{accept}[i]$, $\text{reject}[i]$ leurs états finaux



$$\delta(q_0, (a, a_1, \dots, a_m)) = (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$$$

$$\delta(q_0, (B, a_1, \dots, a_m)) = (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow))$$

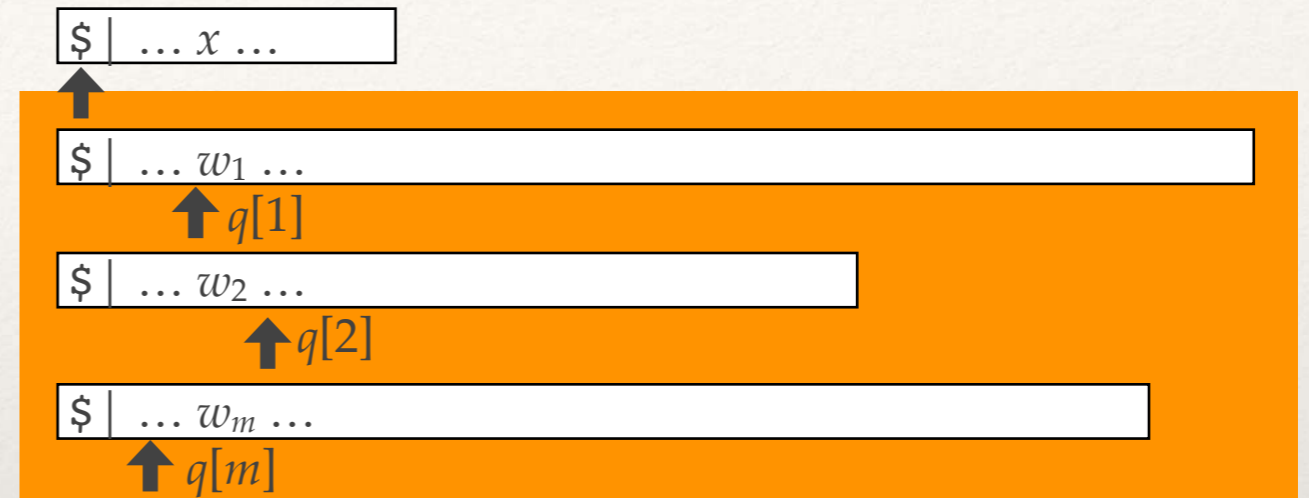
$$\delta(q_1, (a, a_1, \dots, a_m)) = (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$$$

$$\delta(q_1, (\$, a_1, \dots, a_m)) = ((\text{init}[1], \dots, \text{init}[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow))$$

... qu'on rajoute à l'ensemble d'états de M_i on étend aussi δ_i de sorte que M_i **boucle** sur $\text{accept}[i]$ / $\text{reject}[i]$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $\text{init}[i]$ son état initial et $\text{accept}[i]$, $\text{reject}[i]$ leurs états finaux



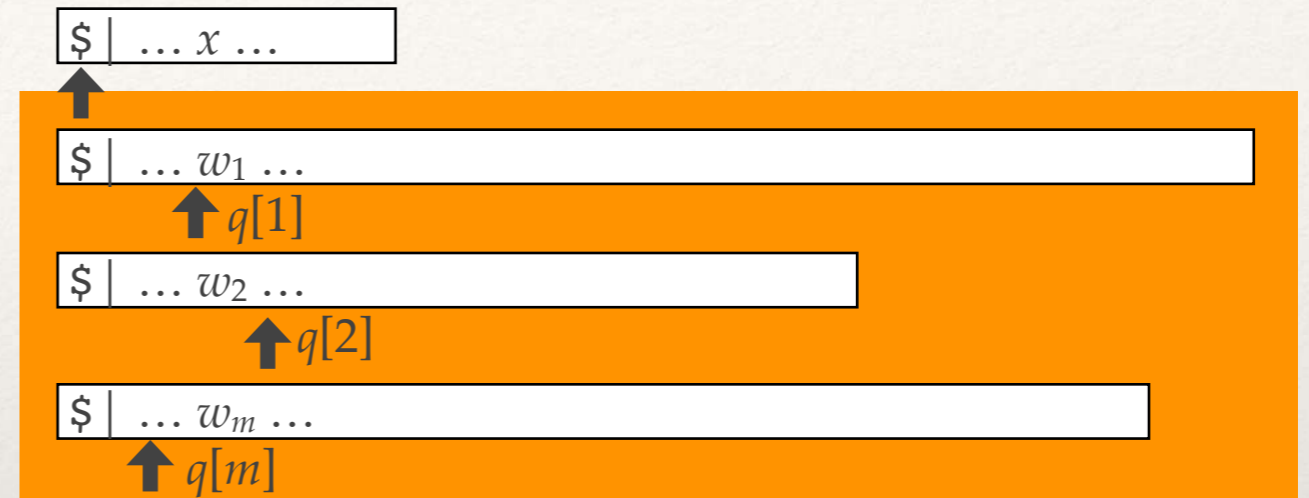
$$\begin{aligned} \delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$ \\ \delta(q_1, (\$, a_1, \dots, a_m)) &= ((\text{init}[1], \dots, \text{init}[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \\ \delta((q[1], \dots, q[m]), (a, a_1, \dots, a_m)) &= ((q'[1], \dots, q'[m]), (b_1, \dots, b_m), (\downarrow, d_1, \dots, d_m)) \\ &\text{ si } \delta_1(q[1], a_1) = (q'[1], b_1, d_1), \dots, \delta_m(q[m], a_m) = (q'[m], b_m, d_m) \end{aligned}$$

On simule une étape de chaque M_i en parallèle

... qu'on rajoute à l'ensemble d'états de M_i on étend aussi δ_i de sorte que M_i **boucle** sur $\text{accept}[i]$ / $\text{reject}[i]$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $\text{init}[i]$ son état initial et $\text{accept}[i]$, $\text{reject}[i]$ leurs états finaux



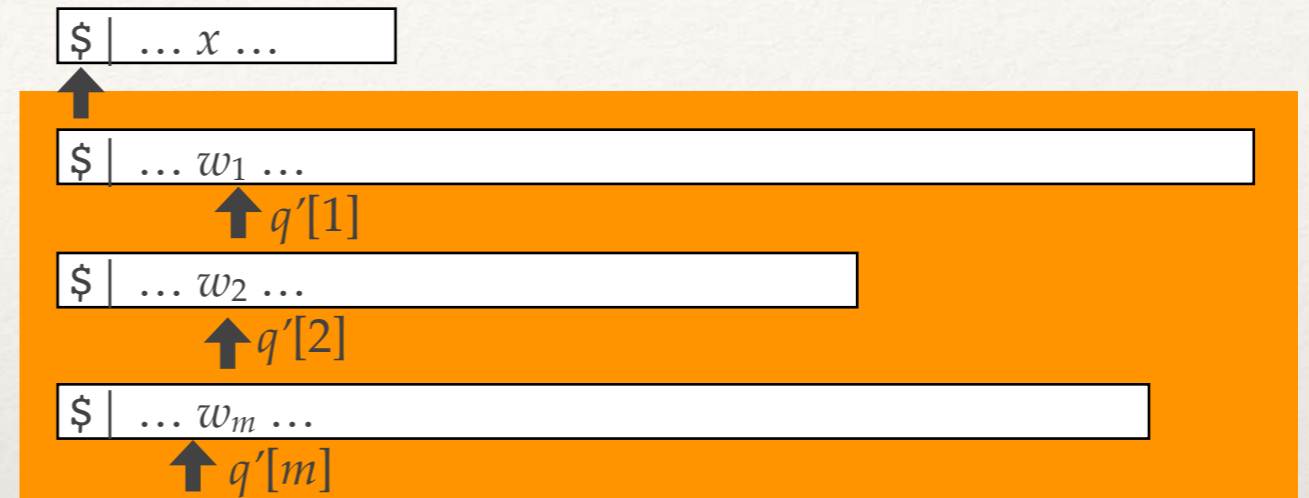
$$\begin{aligned} \delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \quad \text{si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \quad \text{si } a \neq \$ \\ \delta(q_1, (\$, a_1, \dots, a_m)) &= ((\text{init}[1], \dots, \text{init}[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \\ \delta((q[1], \dots, q[m]), (a, a_1, \dots, a_m)) &= ((q'[1], \dots, q'[m]), (b_1, \dots, b_m), (\downarrow, d_1, \dots, d_m)) \\ &\quad \text{si } \delta_1(q[1], a_1) = (q'[1], b_1, d_1), \dots, \delta_m(q[m], a_m) = (q'[m], b_m, d_m) \\ &\quad \text{et (tous les } q'[i] \neq \text{accept}[i]) \\ &\quad \text{et (au moins un } q'[i] \neq \text{reject}[i]) \end{aligned}$$

On simule une étape de chaque M_i en parallèle

... qu'on rajoute à l'ensemble d'états de M_i on étend aussi δ_i de sorte que M_i boucle sur $\text{accept}[i]$ / $\text{reject}[i]$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $\text{init}[i]$ son état initial et $\text{accept}[i]$, $\text{reject}[i]$ leurs états finaux



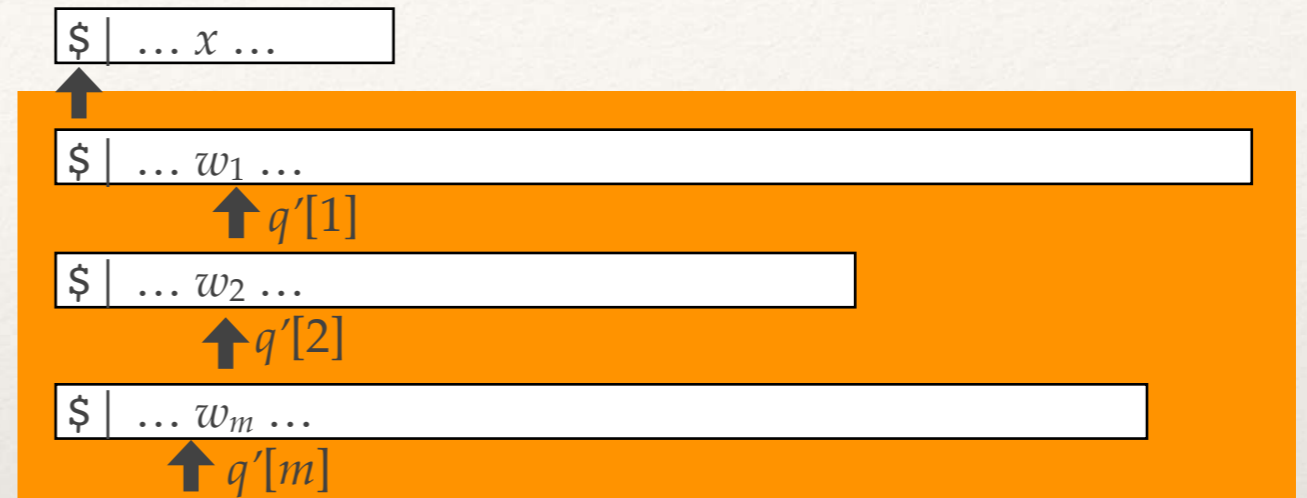
$$\begin{aligned} \delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$ \\ \delta(q_1, (\$, a_1, \dots, a_m)) &= ((\text{init}[1], \dots, \text{init}[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \\ \delta((q[1], \dots, q[m]), (a, a_1, \dots, a_m)) &= ((q'[1], \dots, q'[m]), (b_1, \dots, b_m), (\downarrow, d_1, \dots, d_m)) \\ &\text{ si } \delta_1(q[1], a_1) = (q'[1], b_1, d_1), \dots, \delta_m(q[m], a_m) = (q'[m], b_m, d_m) \\ &\text{ et (tous les } q'[i] \neq \text{accept}[i]) \\ &\text{ et (au moins un } q'[i] \neq \text{reject}[i]) \end{aligned}$$

... qu'on rajoute à l'ensemble d'états de M_i
on étend aussi δ_i de sorte que M_i boucle sur $\text{accept}[i]$ / $\text{reject}[i]$

Simulation parallèle

- ❖ La bande i va servir à simuler la machine M_i
- ❖ Soit δ_i la fonction de transition de M_i , $\text{init}[i]$ son état initial et $\text{accept}[i]$, $\text{reject}[i]$ leurs états finaux

On accepte si toutes les machines acceptent, on rejette si au moins une machine rejette



$$\begin{aligned} \delta(q_0, (a, a_1, \dots, a_m)) &= (q_0, (a, \dots, a), (\rightarrow, \rightarrow, \dots, \rightarrow)) \text{ si } a \neq B, \$ \\ \delta(q_0, (B, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \\ \delta(q_1, (a, a_1, \dots, a_m)) &= (q_1, (a_1, \dots, a_m), (\leftarrow, \leftarrow, \dots, \leftarrow)) \text{ si } a \neq \$ \\ \delta(q_1, (\$, a_1, \dots, a_m)) &= ((\text{init}[1], \dots, \text{init}[m]), (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \\ \delta((q[1], \dots, q[m]), (a, a_1, \dots, a_m)) &= ((q'[1], \dots, q'[m]), (b_1, \dots, b_m), (\downarrow, d_1, \dots, d_m)) \\ &\text{ si } \delta_1(q[1], a_1) = (q'[1], b_1, d_1), \dots, \delta_m(q[m], a_m) = (q'[m], b_m, d_m) \\ &\text{ et (tous les } q'[i] \neq \text{accept}[i]) \\ &\text{ et (au moins un } q'[i] \neq \text{reject}[i]) \\ \delta((q[1], \dots, q[m]), (a, a_1, \dots, a_m)) &= (\text{accept}, (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \\ &\text{ si au moins un } q'[i] = \text{accept}[i] \\ \delta((q[1], \dots, q[m]), (a, a_1, \dots, a_m)) &= (\text{reject}, (a_1, \dots, a_m), (\downarrow, \downarrow, \dots, \downarrow)) \\ &\text{ si tous les } q'[i] = \text{reject}[i] \end{aligned}$$

... qu'on rajoute à l'ensemble d'états de M_i
on étend aussi δ_i de sorte que M_i boucle sur $\text{accept}[i]$ / $\text{reject}[i]$

Opérations booléennes

- ❖ Donc: l'**union** d'un nombre fini de langages **r.e.** est **r.e.**

Opérations booléennes

- ❖ Donc: l'**union** d'un nombre fini de langages **r.e.** est r.e.
- ❖ La même construction montre: l'**union** d'un nombre fini de langages **récur­sifs** est récur­sif

Opérations booléennes

- ❖ Donc: l'**union** d'un nombre fini de langages **r.e.** est r.e.
- ❖ La même construction montre: l'**union** d'un nombre fini de langages **récur­sifs** est récur­sif
- ❖ Symétriquement,
l'**intersection** d'un nombre fini de langages **r.e.** est r.e.

Opérations booléennes

- ❖ Donc: l'**union** d'un nombre fini de langages **r.e.** est r.e.
- ❖ La même construction montre: l'**union** d'un nombre fini de langages **rékursifs** est rékursif
- ❖ Symétriquement,
l'**intersection** d'un nombre fini de langages **r.e.** est r.e.
- ❖ L'**intersection** d'un nombre fini de langages **rékursifs** est réursive

Opérations booléennes

- ❖ Donc: l'**union** d'un nombre fini de langages **r.e.** est r.e.
- ❖ La même construction montre: l'**union** d'un nombre fini de langages **récur­sifs** est récur­sif
- ❖ Symétriquement,
l'**intersection** d'un nombre fini de langages **r.e.** est r.e.
- ❖ L'**intersection** d'un nombre fini de langages **récur­sifs** est récur­sive
- ❖ De plus, le **complémentaire** d'un langage **récur­sif** est récur­sif (on n'a pas l'analogue pour les langages r.e.!).

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.
- ❖ *Preuve* (seulement si).
Si L est récursif, donc décidé par une mT M ,

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.
- ❖ *Preuve* (seulement si).
Si L est récursif, donc décidé par une mT M ,
- ❖ alors $L(M) = L$ (langage *accepté* par M)

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.
- ❖ *Preuve* (seulement si).
Si L est récursif, donc décidé par une mT M ,
- ❖ alors $L(M) = L$ (langage *accepté* par M)
- ❖ et $L(M^c) = \complement L$, où M^c est la machine obtenue à partir de M en échangeant **accept** et **reject**

Complémentaires de langages r.e.

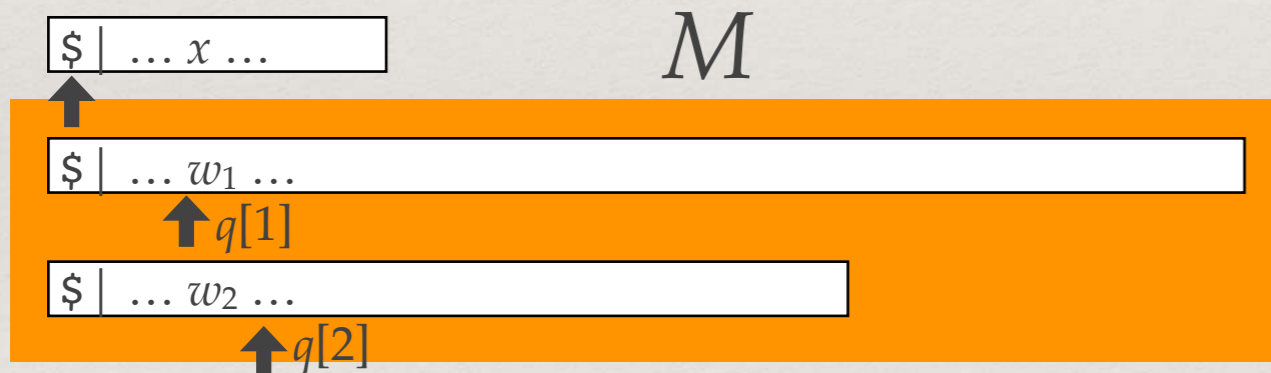
- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.
- ❖ *Preuve (si).* Supposons $L = L(M_1)$, complémentaire $L(M_2)$

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.
- ❖ *Preuve (si).* Supposons $L = L(M_1)$, complémentaire $L(M_2)$
- ❖ On définit une machine I/O M à 2 bandes de travail qui, sur toute entrée x , effectue une simulation parallèle de M_1 et M_2 , et **accepte** si M_1 accepte, **rejette** si M_2 accepte

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.
- ❖ *Preuve (si).* Supposons $L = L(M_1)$, complémentaire $L(M_2)$

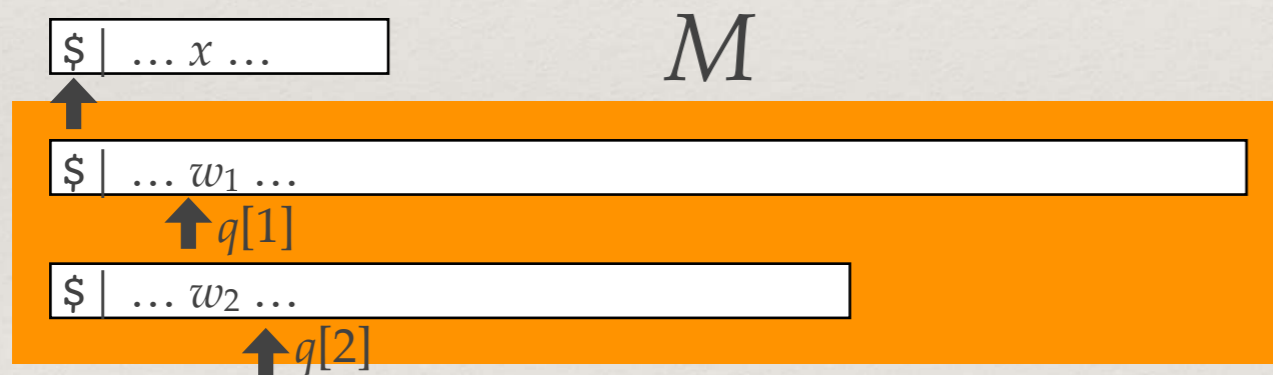


$$\begin{aligned}
\delta(q_0, (a, a_1, a_2)) &= (q_0, (a, a), (\rightarrow, \rightarrow, \rightarrow)) \quad \text{si } a \neq B, \$ \\
\delta(q_0, (B, a_1, a_2)) &= (q_1, (a_1, a_2), (\leftarrow, \leftarrow, \leftarrow)) \\
\delta(q_1, (a, a_1, a_2)) &= (q_1, (a_1, a_2), (\leftarrow, \leftarrow, \leftarrow)) \quad \text{si } a \neq \$ \\
\delta(q_1, (\$, a_1, a_2)) &= ((\text{init}[1], \text{init}[2]), (a_1, a_2), (\downarrow, \downarrow, \downarrow)) \\
\delta((q[1], q[2]), (a, a_1, a_2)) &= ((q'[1], q'[2]), (b_1, b_2), (\downarrow, d_1, d_2)) \\
&\quad \text{si } \delta_1(q[1], a_1) = (q'[1], b_1, d_1), \delta_2(q[2], a_2) = (q'[2], b_2, d_2) \\
&\quad \text{et } q'[1] \neq \text{accept}[1], q'[2] \neq \text{accept}[2] \\
\delta((q[1], q[2]), (a, a_1, a_2)) &= (\text{accept}, (a_1, a_2), (\downarrow, \downarrow, \downarrow)) \\
&\quad \text{si } q'[1] = \text{accept}[1] \\
\delta((q[1], q[2]), (a, a_1, a_2)) &= (\text{reject}, (a_1, a_2), (\downarrow, \downarrow, \downarrow)) \\
&\quad \text{si } q'[2] = \text{accept}[2] \quad (\text{et } q'[1] \neq \text{accept}[1])
\end{aligned}$$

Complémentaires de langages r.e.

- ❖ Un langage est **co-r.e.** ssi son complémentaire est r.e.
- ❖ **Théorème.** Un langage est récursif ssi il est r.e. et co-r.e.

- ❖ *Preuve (si).* Supposons $L = L(M_1)$, complémentaire $L(M_2)$



$$\begin{aligned} \delta(q_0, (a, a_1, a_2)) &= (q_0, (a, a), (\rightarrow, \rightarrow, \rightarrow)) \quad \text{si } a \neq B, \$ \\ \delta(q_0, (B, a_1, a_2)) &= (q_1, (a_1, a_2), (\leftarrow, \leftarrow, \leftarrow)) \\ \delta(q_1, (a, a_1, a_2)) &= (q_1, (a_1, a_2), (\leftarrow, \leftarrow, \leftarrow)) \quad \text{si } a \neq \$ \\ \delta(q_1, (\$, a_1, a_2)) &= ((\text{init}[1], \text{init}[2]), (a_1, a_2), (\downarrow, \downarrow, \downarrow)) \\ \delta((q[1], q[2]), (a, a_1, a_2)) &= ((q'[1], q'[2]), (b_1, b_2), (\downarrow, d_1, d_2)) \\ &\quad \text{si } \delta_1(q[1], a_1) = (q'[1], b_1, d_1), \delta_2(q[2], a_2) = (q'[2], b_2, d_2) \\ &\quad \text{et } q'[1] \neq \text{accept}[1], q'[2] \neq \text{accept}[2] \\ \delta((q[1], q[2]), (a, a_1, a_2)) &= (\text{accept}, (a_1, a_2), (\downarrow, \downarrow, \downarrow)) \\ &\quad \text{si } q'[1] = \text{accept}[1] \\ \delta((q[1], q[2]), (a, a_1, a_2)) &= (\text{reject}, (a_1, a_2), (\downarrow, \downarrow, \downarrow)) \\ &\quad \text{si } q'[2] = \text{accept}[2] \quad (\text{et } q'[1] \neq \text{accept}[1]) \end{aligned}$$

- ❖ Alors M décide L :
si $x \in L$, M accepte;
si $x \notin L$, M rejette.

Notamment, M termine sur toute entrée x . \square

La prochaine fois

La prochaine fois

- ❖ Machines de Turing universelles
- ❖ Problème de l'arrêt
- ❖ Il existe des langages r.e. non rékursifs