

Jean Goubault-Larrecq

Calculabilité

3. Machines
universelles,
problèmes
indécidables

Aujourd'hui

- ❖ Machines de Turing universelles
- ❖ Indécidabilité, problème de l'arrêt
- ❖ Basé sur le poly d'Hubert Comon (suite):
<http://www.lsv.fr/~comon/Calculabilite1/cours20.2.pdf>

Indécidabilité et problème de l'arrêt, informellement

En utilisant Lisp

- ❖ Imaginons un langage, tel que Lisp,
 - permettant d'exprimer exactement les fonctions semi-calculables, comme les machines de Turing
 - et disposant de `eval`, `quote`, `apply` (et autre chose)
- ❖ On notera `'M` pour `(quote M)`, comme en Lisp
- ❖ $(\text{eval } 'M) \rightarrow^* M$
- ❖ `'M` est une *donnée*; `M` est un *programme*
[Plus tard: `'M` sera un mot, et `M` une machine de Turing]

eval, apply, make-apply, list

- ❖ $(\text{eval } 'M) \rightarrow^* M$
- ❖ $(\text{apply } 'M 'N) \rightarrow^* (M N)$
[petit mensonge: en fait $(\text{apply } 'M '(N_1 \dots N_k)) \rightarrow^* (M N_1 \dots N_k)$]
- ❖ On va supposer une primitive `make-apply` telle que
 $(\text{make-apply } 'M 'N) \rightarrow^* '(M N)$
- ❖ En Lisp, `make-apply=list`
- ❖ `apply` est du sucre syntaxique:
 $(\text{apply } M N) = (\text{eval } (\text{make-apply } M N))$

kwote

- ❖ On ne peut pas calculer $'M$ à partir de M
on peut calculer des programmes à partir de données, pas l'inverse
- ❖ ... mais Lisp dispose d'une fonction `kwote`
(données \rightarrow données)
telle que $(\text{kwote } a) \rightarrow^* \text{le quote de la valeur de } a$
- ❖ Par exemple,
 $(\text{kwote } (\text{cons } 1 (\text{cons } 2 \text{ nil}))) \rightarrow^* '(1 2)$
- ❖ Aussi, $(\text{kwote } 'M) \rightarrow^* ''M$

Problème de l'arrêt

- ❖ Supposons que l'on ait un programme Lisp `haltp` tel que `(haltp 'M)`
 - retourne `t` (vrai) si `M` termine,
 - et `nil` (faux) sinon
- ❖ Par exemple, définissons:

```
(defun forever-loop ()  
  (forever-loop))
```
- ❖ Nécessairement,
`(haltp 'forever-loop)` retourne `nil`

Problème de l'arrêt

Supposons que l'on ait un programme Lisp `haltp` tel que `(haltp 'M)`
— retourne `t` (vrai) si `M` termine,
— et `nil` (faux) sinon

❖ On forme:

```
(defun turing (prog)
  (cond ((haltp (make-apply prog (kwote prog)))
         (forever-loop))
        (t nil)))
```

❖ Que retourne `(turing 'turing)`?

Problème de l'arrêt

Supposons que l'on ait un programme Lisp
haltp tel que (haltp 'M)
— retourne t (vrai) si M termine,
— et nil (faux) sinon

```
(defun turing (prog)
  (cond ((haltp (make-apply prog (kwote prog)))
         (forever-loop))
        (t nil)))
```

❖ (turing 'turing) →*

```
(cond ((haltp (make-apply 'turing (kwote 'turing)))
       (forever-loop))
      (t nil)))
```

(1) si (turing 'turing) termine,
(haltp '(turing 'turing)) →* t

❖ →* (cond ((haltp '(turing 'turing))
(forever-loop))
(t nil)))

(2) alors on exécute (forever-loop),
qui ne termine pas

(3) donc (turing 'turing) ne termine pas:
contradiction

Problème de l'arrêt

Supposons que l'on ait un programme Lisp
haltp tel que (haltp 'M)
— retourne t (vrai) si M termine,
— et nil (faux) sinon

```
(defun turing (prog)
  (cond ((haltp (apply prog (kwote prog)))
         (forever-loop))
        (t nil)))
```

❖ (turing 'turing) →*

```
(cond ((haltp (make-apply 'turing (kwote 'turing)))
       (forever-loop))
      (t nil)))
```

❖ →* (cond ((haltp '(turing 'turing))
 (forever-loop))
 (t nil)))

(1) si (turing 'turing) ne termine pas,
(haltp '(turing 'turing)) →* t

(2) alors on termine sur nil

(3) donc (turing 'turing) termine:
contradiction de nouveau.

On obtient une contradiction dans tous les cas:
haltp n'existe pas
**On ne peut pas décider de l'arrêt d'un
programme donné en entrée.**

Indécidabilité et problème de l'arrêt, formellement

Codages de machines de Turing

- ❖ On va voir qu'on peut déjà implémenter tout ça dans le modèle des machines de Turing:
- ❖ ' M : un mot qui représente la machine de Turing M
dans le poly: noté $\langle M \rangle$
- ❖ `apply`: machine de Turing **universelle**
prend $\langle M \rangle$; x en entrée, et simule $M(x)$
- ❖ `kwote`: prend $\langle M \rangle$ en entrée, et retourne $\langle \langle M \rangle \rangle$
- ❖ Note: je supposerai dans la suite que
l'alphabet Σ de M contient au moins $0, 1, ;$ en plus de **B** et **\$**

Quote: $\langle M \rangle$

Pour le moment, on fixe Q, Σ

- ❖ Soit $M \stackrel{\text{def}}{=} (Q, q_0, \Sigma, \delta, \underline{B}, \underline{\$})$
- ❖ On énumère $Q \times \Sigma \stackrel{\text{def}}{=} \{(q_1, a_1), \dots, (q_m, a_m)\}$
et on pose $(q'_i, b_i, d_i) \stackrel{\text{def}}{=} \delta(q_i, a_i), 1 \leq i \leq m$
- ❖ On code δ par le mot
 $\langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
sur l'alphabet $Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$
- ❖ On pose $\langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{B} \underline{\$} [\text{autres lettres de } \Sigma] \# \langle \delta \rangle$

Pas exactement le codage
du théorème 6.5.1 du poly

Apply: la machine $U_{Q\Sigma}$

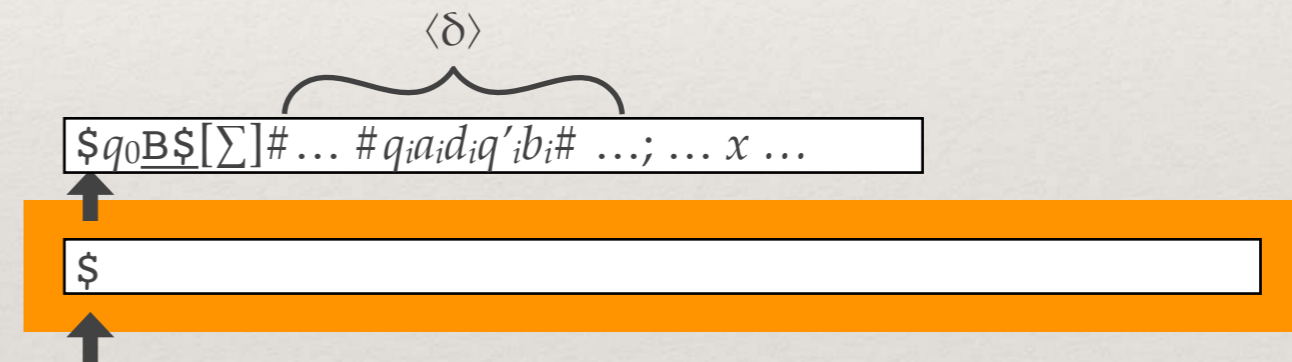
- ❖ $U_{Q\Sigma}$: machine I/O à 1 bande de travail, pas de bande de sortie, sur un **alphabet étendu** $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \mathbb{B}, \$\}$

$$\diamond \langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$$

$$\diamond \langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathbb{B}} \underline{\$} [\Sigma] \# \langle \delta \rangle$$

$$\diamond \text{alphabet } Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$$

- ❖ En entrée: $\langle M \rangle; x$



$U_{Q\Sigma}$ commence par:

— lire et mémoriser $q_0, \underline{\mathbb{B}}, \underline{\$}$ dans son état interne

Apply: la machine $U_{Q\Sigma}$

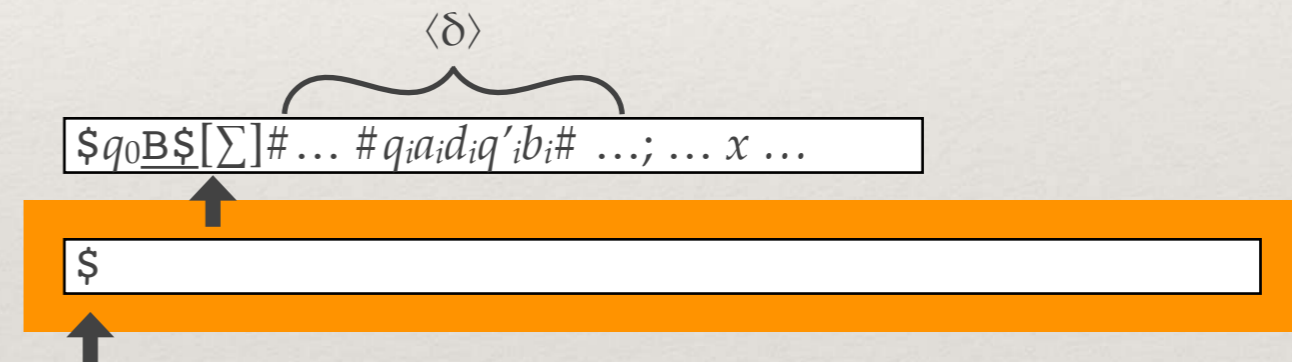
- ❖ $U_{Q\Sigma}$: machine I/O à 1 bande de travail, pas de bande de sortie, sur un alphabet étendu $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \mathbb{B}, \$\}$

$$\diamond \langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$$

$$\diamond \langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathbb{B}} \underline{\$} [\Sigma] \# \langle \delta \rangle$$

$$\diamond \text{alphabet } Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$$

- ❖ En entrée: $\langle M \rangle; x$



$U_{Q\Sigma}$ commence par:

- lire et mémoriser $q_0, \underline{\mathbb{B}}, \underline{\$}$ dans son état interne
- écrire $q_0, \underline{\$}$ sur la bande de travail

Apply: la machine $U_{Q\Sigma}$

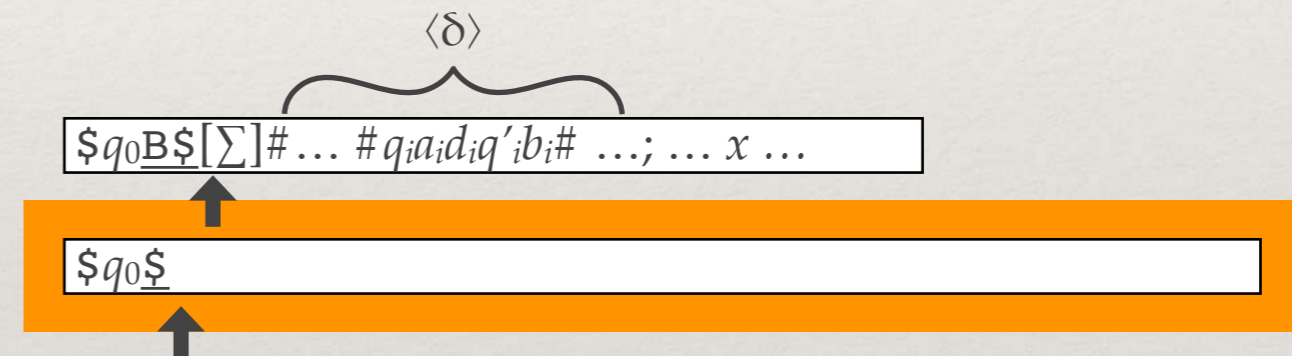
- ❖ $U_{Q\Sigma}$: machine I/O à 1 bande de travail, pas de bande de sortie, sur un **alphabet étendu** $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \mathbb{B}, \$\}$

$$\diamond \langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$$

$$\diamond \langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathbb{B}} \underline{\$} [\Sigma] \# \langle \delta \rangle$$

$$\diamond \text{alphabet } Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$$

- ❖ En entrée: $\langle M \rangle; x$



$U_{Q\Sigma}$ commence par:

- lire et mémoriser $q_0, \underline{\mathbb{B}}, \underline{\$}$ dans son état interne
- écrire $q_0, \underline{\$}$ sur la bande de travail
- avancer jusqu'au ;

Apply: la machine $U_{Q\Sigma}$

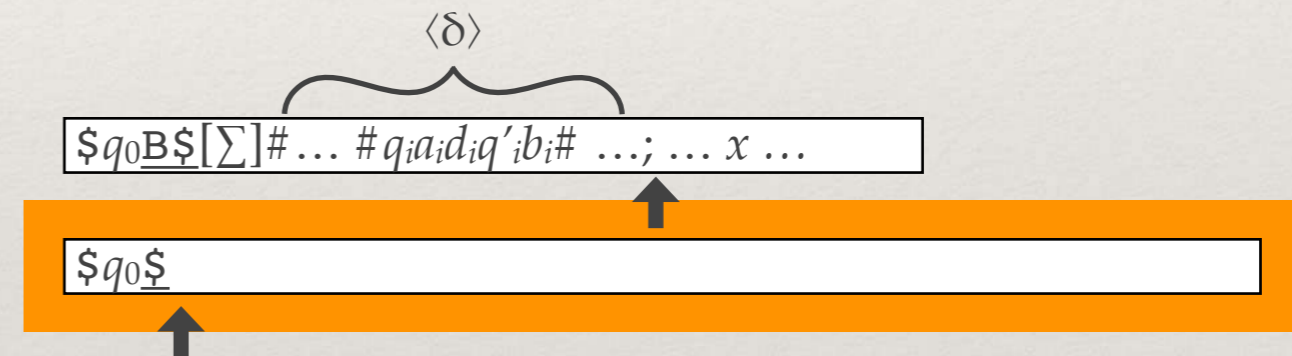
- ❖ $U_{Q\Sigma}$: machine I/O à 1 bande de travail, pas de bande de sortie, sur un **alphabet étendu** $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \mathbf{B}, \$\}$

$$\diamond \langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$$

$$\diamond \langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathbf{B}} \underline{\$} [\Sigma] \# \langle \delta \rangle$$

$$\diamond \text{alphabet } Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$$

- ❖ En entrée: $\langle M \rangle; x$



$U_{Q\Sigma}$ commence par:

- lire et mémoriser $q_0, \underline{\mathbf{B}}, \underline{\$}$ dans son état interne
- écrire $q_0, \underline{\$}$ sur la bande de travail
- avancer jusqu'au ;
- recopier x sur la bande de travail

Apply: la machine $U_{Q\Sigma}$

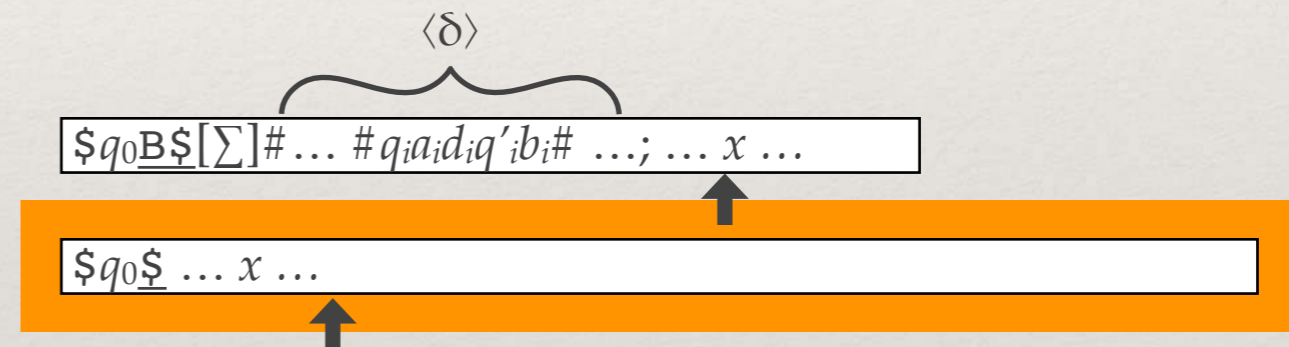
- ❖ $U_{Q\Sigma}$: machine I/O à 1 bande de travail, pas de bande de sortie, sur un **alphabet étendu** $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \mathbf{B}, \$\}$

$$\diamond \langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$$

$$\diamond \langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathbf{B}} \underline{\$} [\Sigma] \# \langle \delta \rangle$$

$$\diamond \text{alphabet } Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$$

- ❖ En entrée: $\langle M \rangle; x$



$U_{Q\Sigma}$ commence par:

- lire et mémoriser $q_0, \underline{\mathbf{B}}, \underline{\$}$ dans son état interne
- écrire $q_0, \underline{\$}$ sur la bande de travail
- avancer jusqu'au ;
- recopier x sur la bande de travail
- nettoyer: ramener les têtes en début de bande

Apply: la machine $U_{Q\Sigma}$

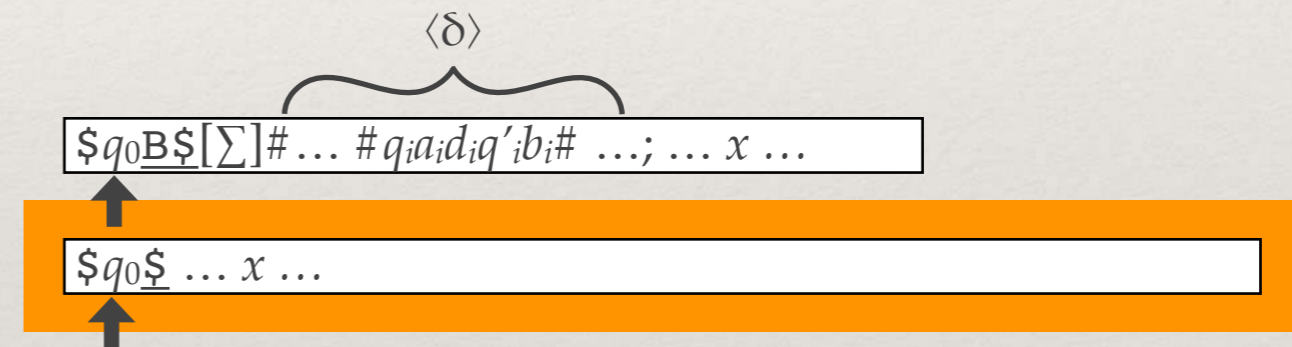
- ❖ $U_{Q\Sigma}$: machine I/O à 1 bande de travail, pas de bande de sortie, sur un **alphabet étendu** $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \mathbb{B}, \$\}$

$$\diamond \langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$$

$$\diamond \langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathbb{B}} \underline{\$} [\Sigma] \# \langle \delta \rangle$$

$$\diamond \text{alphabet } Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\}$$

- ❖ En entrée: $\langle M \rangle; x$

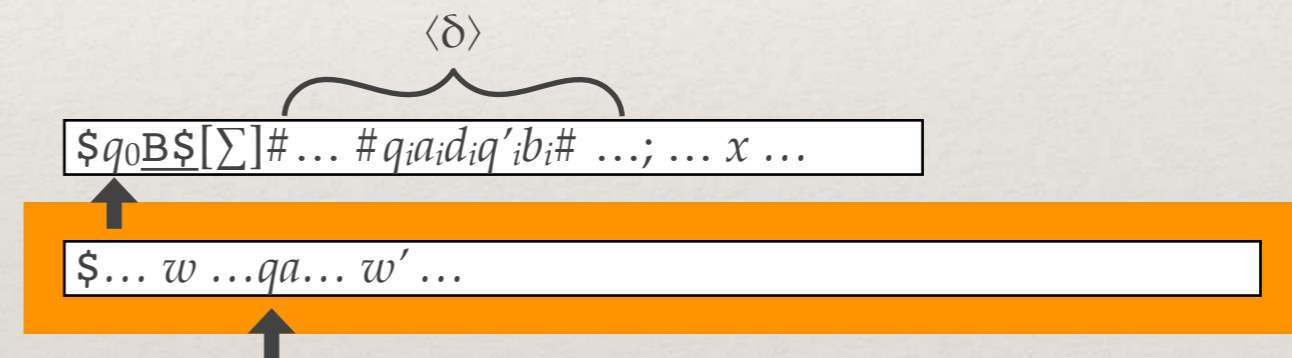


$U_{Q\Sigma}$ commence par:

- lire et mémoriser $q_0, \underline{\mathbb{B}}, \underline{\$}$ dans son état interne
- écrire $q_0, \underline{\$}$ sur la bande de travail
- avancer jusqu'au ;
- recopier x sur la bande de travail
- nettoyer: ramener les têtes en début de bande (après les \$)

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wq\bar{w}'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



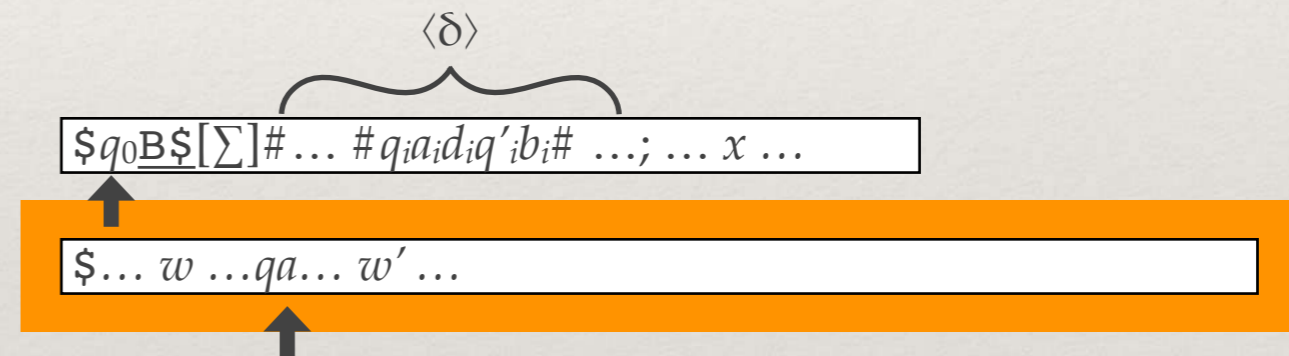
Pour simuler une étape de M , $U_{Q\Sigma}$:

— lit q et a sur la bande de travail, et les mémorise dans son état interne

si pas de a (à droite de la bande),
ou plus généralement si $a=B$,
on fait comme si $a=\underline{B}$ (exercice)

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\$$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

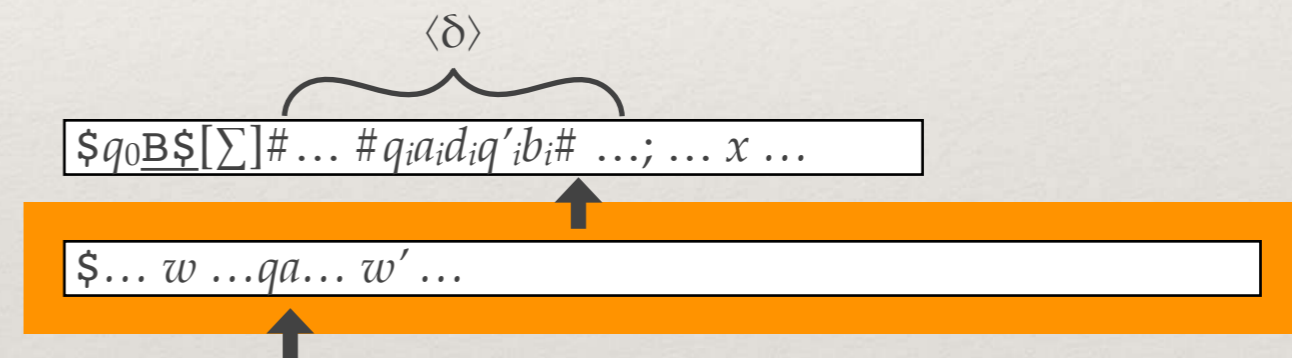


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\# q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

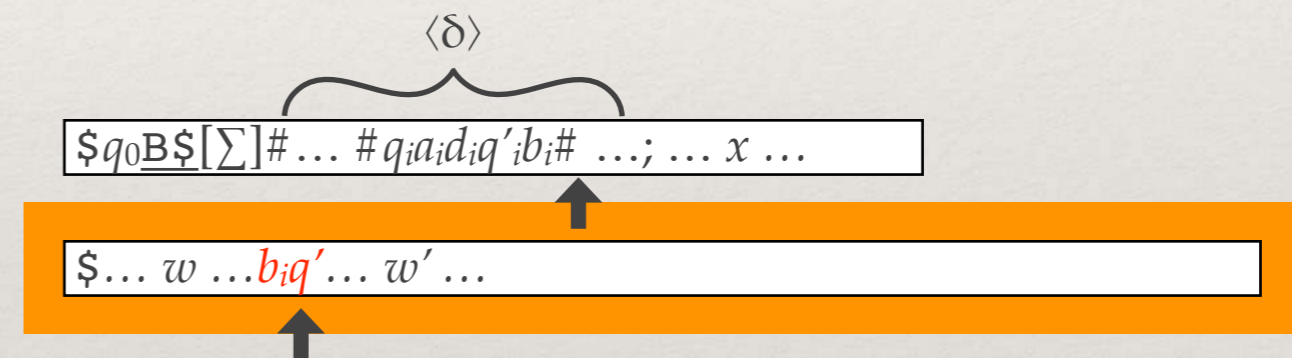


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\# q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \rightarrow$, ...

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wq\bar{w}'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

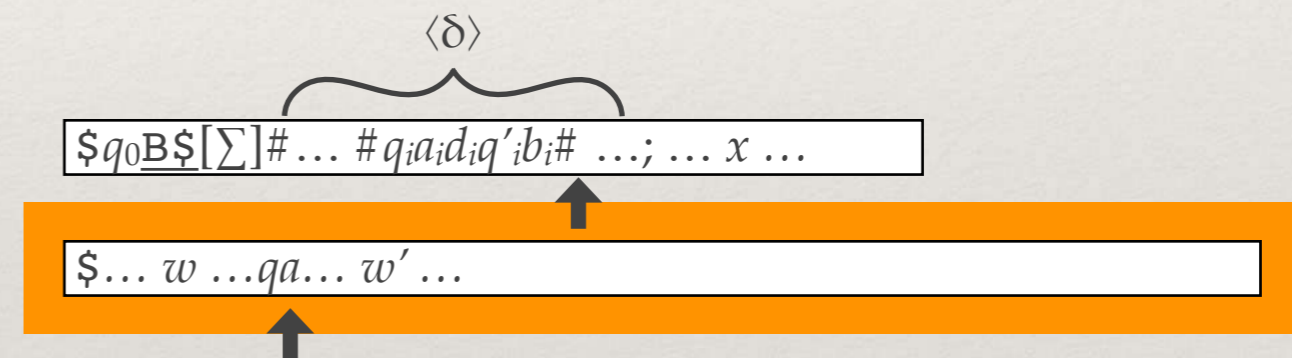


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\#q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \rightarrow$, on écrit b_i , q' et on laisse la tête sous q'

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

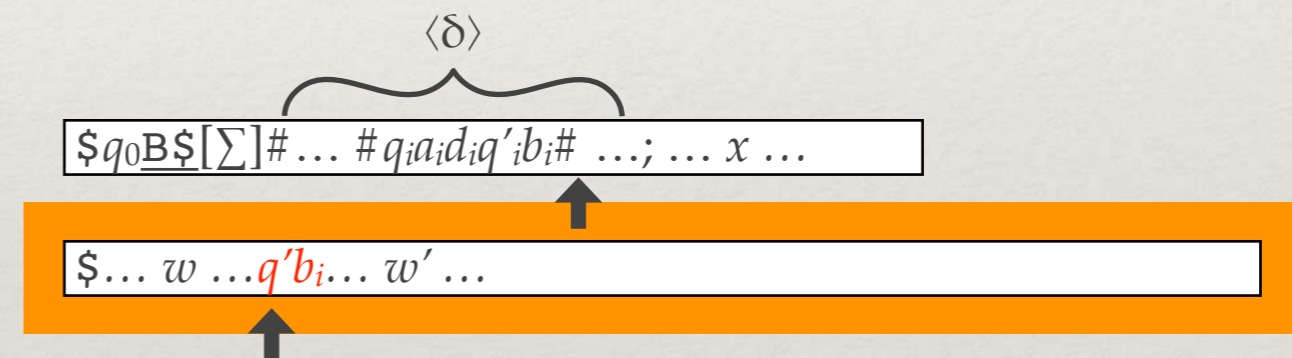


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\# q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \downarrow$, ...

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

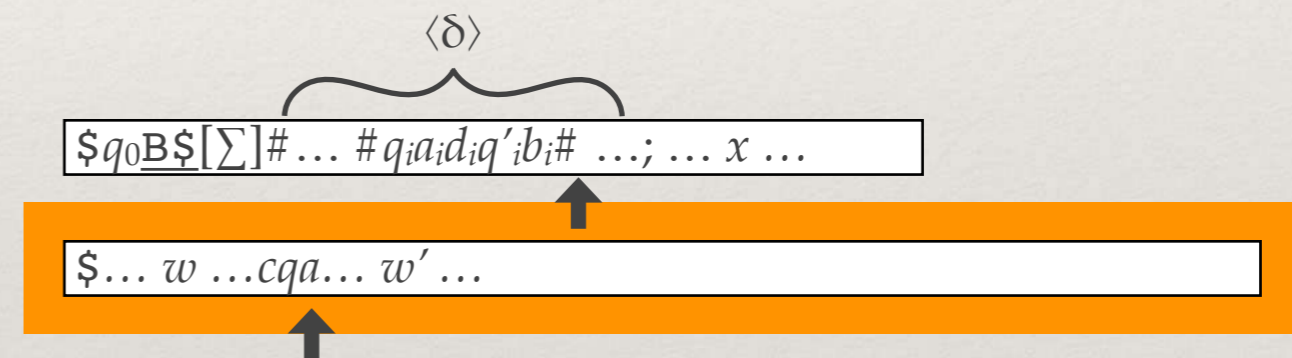


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\#q_i a_i d_i q' b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \downarrow$, on écrit q' , b_i , et on ramène la tête sous q'

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

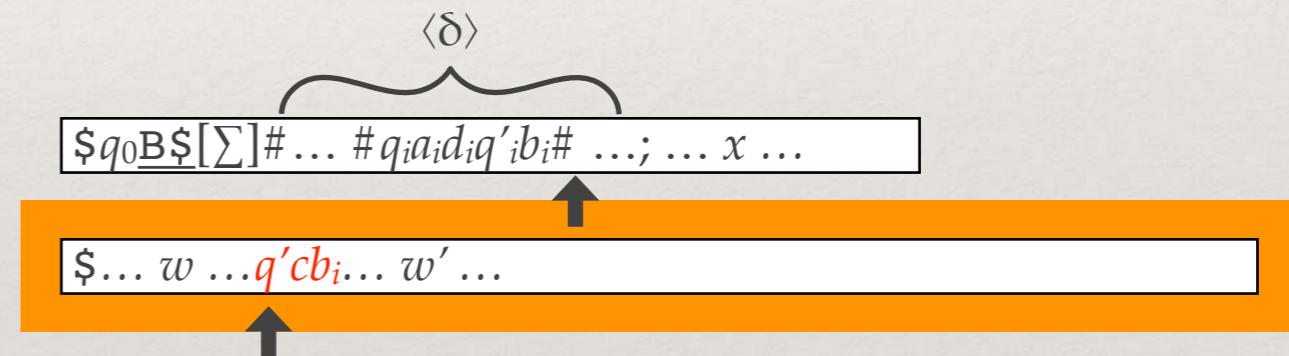


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\#q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \leftarrow$, nécessairement il y a une lettre c avant q et...

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

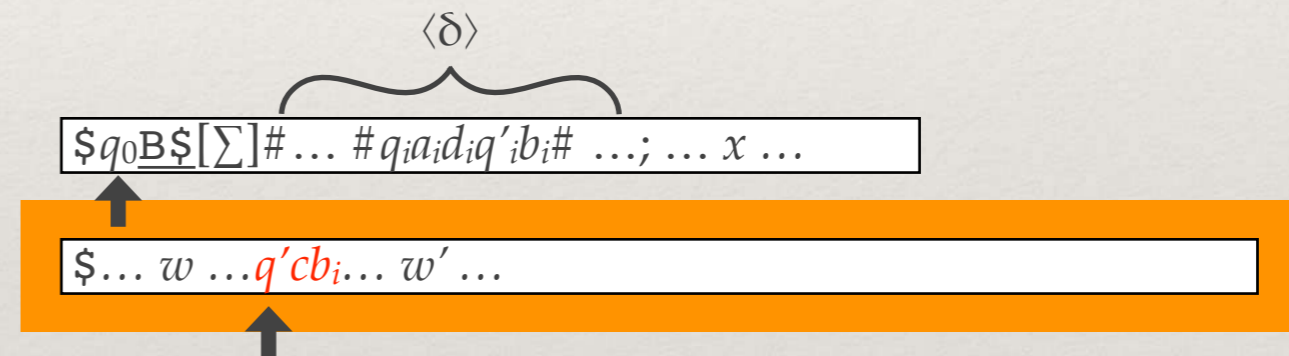


Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\#q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \downarrow$, nécessairement il y a une lettre c avant q , on écrit q' , c , b_i , et on ramène la tête sous q'
- enfin, on nettoie, en ramenant la tête de la bande d'entrée au début

Apply: la machine $U_{Q\Sigma}$

- ❖ A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$, si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$ [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



Pour simuler une étape de M , $U_{Q\Sigma}$:

- lit q et a sur la bande de travail, et les mémorise dans son état interne
- parcourt son entrée jusqu'à trouver $\# q_i a_i d_i q' i b_i$ avec $q_i = q$, $a_i = a$ et mémorise d_i , q'_i , b_i , dans son état interne
- si $d_i = \downarrow$, nécessairement il y a une lettre c avant q , on écrit q' , c , b_i , et on ramène la tête sous q'
- enfin, on nettoie, en ramenant la tête de la bande d'entrée au début

Apply: la machine $U_{Q\Sigma}$

- ❖ Dernier point: on fait tout ceci uniquement si $q' \neq \text{accept, reject}$
- ❖ Sinon, $U_{Q\Sigma}$ accepte, ou rejette, selon le cas
- ❖ **Proposition.** Pour tout mT M (sur l'alphabet Σ et d'états dans Q), pour toute entrée $x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*$,
 - $U_{Q\Sigma}$ accepte $\langle M \rangle; x$ ssi M accepte x
 - $U_{Q\Sigma}$ rejette $\langle M \rangle; x$ ssi M rejette x .
- ❖ **Attention** (rappel):
l'alphabet de $U_{Q\Sigma}$ est $\Sigma' \stackrel{\text{def}}{=} Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#, \underline{B}, \underline{\$}\}$, pas Σ
- ❖ Pour corriger ce problème...

Réduire à deux lettres (plus ;)

- ❖ Une astuce consiste à tout recoder en binaire, avec ; comme séparateur:

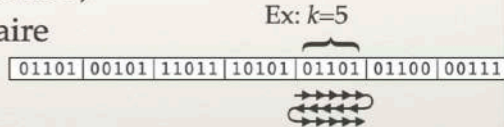
$$(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} \text{bin}_{\Sigma'}(0); \text{bin}_{\Sigma'}(1); \text{bin}_{\Sigma'}(;); \text{bin}_{\Sigma'}(\text{accept}); \text{bin}_{\Sigma'}(\text{reject}); \dots;; \text{bin}_{\Sigma'}(\langle M \rangle; x)$$

Point subtil: la fonction $\text{bin}_{\Sigma'}$ n'est pas donnée de l'extérieur, mais est codée par le préfixe $\text{bin}_{\Sigma'}(0); \text{bin}_{\Sigma'}(1); \text{bin}_{\Sigma'}(;); \text{bin}_{\Sigma'}(\text{accept}); \text{bin}_{\Sigma'}(\text{reject}); \dots;;$

- ❖ La machine U **simule** chaque étape de calcul de U_{Σ} comme avant...
- ❖ à ceci près que la longueur k des codes binaires de chaque lettre **dépend** de l'entrée $\langle M \rangle; x$
- ❖ On va donc tout reprendre depuis le début...

Réduire le nombre de lettres

- ❖ On peut descendre à **deux lettres** dans Γ , en écrivant toutes les lettres en binaire sur $k \approx \Theta(\log |\Sigma|)$ bits

Ex: $k=5$

- ❖ Détails omis: pour simuler un mouvement à droite, on doit maintenant faire k **mouvements** à droite, en mémorisant les bits lus dans le contrôle
- ❖ ... mais aussi k mouvements à gauche pour écrire la nouvelle lettre (sur k bits) et de nouveau k mouvements à droite
- ❖ Finalement, le langage reconnu n'est plus L , mais $\{\text{bin}(w) \mid w \in L\}$ [léger bug du poly]

Codage binaire (avec séparateur ;)

- ❖ Pour chaque Σ' , on choisit un code binaire telle que toutes les lettres a aient un code $\text{bin}_{\Sigma'}(a)$ de la **même longueur** (typiquement $O(\log |\Sigma'|)$)
- ❖ Pour un mot $w = a_1 a_2 \dots a_n \in \Sigma'^*$, (et notamment $\langle M \rangle; x$)
$$\text{bin}_{\Sigma'}(w) \stackrel{\text{def}}{=} \text{bin}_{\Sigma'}(a_1); \text{bin}_{\Sigma'}(a_2); \dots; \text{bin}_{\Sigma'}(a_n);$$
- ❖ Dans la suite, j'écrirai w (en gras) pour $\text{bin}_{\Sigma'}(w)$
[sinon, on aura des expressions immenses]
- ❖ Par exemple, si $\text{bin}_{\Sigma'}(a) = 010$, $\text{bin}_{\Sigma'}(b) = 001$, $\text{bin}_{\Sigma'}(c) = 110$,
$$\mathbf{abbcbc} = 010;001;001;110;001;110;$$
- ❖ Avec cette convention, on a donc:
$$\begin{aligned} (\langle M \rangle; x)^{\text{bin}} &\stackrel{\text{def}}{=} \text{bin}_{\Sigma'}(0); \text{bin}_{\Sigma'}(1); \text{bin}_{\Sigma'}(;); \text{bin}_{\Sigma'}(a); \text{bin}_{\Sigma'}(r); \dots;; \text{bin}_{\Sigma'}(\langle M \rangle; x) \\ &= \mathbf{01;ar\dots;\langle M \rangle;x} \end{aligned}$$

[je note **a** pour **accept**, **r** pour **reject**]

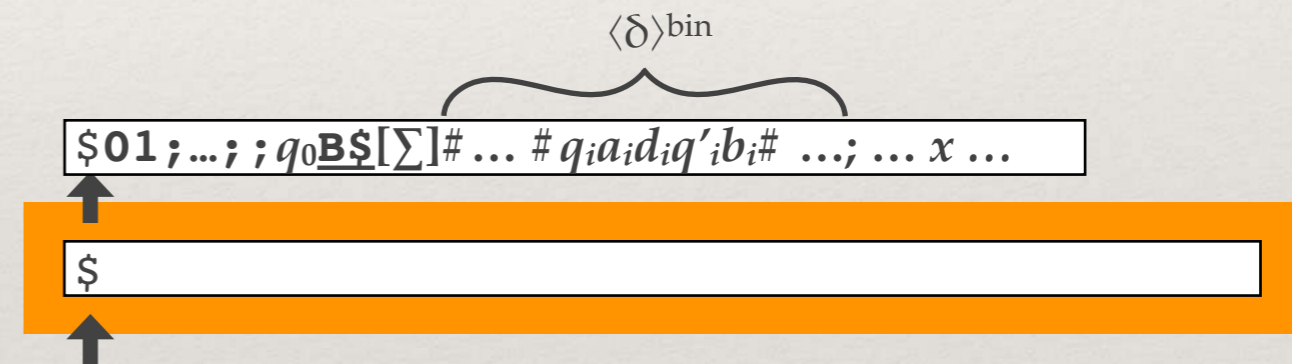
Quote: $\langle M \rangle^{\text{bin}}$

- ❖ Soit $M \stackrel{\text{def}}{=} (Q, q_0, \Sigma, \delta, \underline{B}, \underline{\$})$
- ❖ On énumère $Q \times \Sigma \stackrel{\text{def}}{=} \{(q_1, a_1), \dots, (q_m, a_m)\}$
et on pose $(q'_i, b_i, d_i) \stackrel{\text{def}}{=} \delta(q_i, a_i)$, $1 \leq i \leq m$
- ❖ On code δ par le mot
 $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
sur l'alphabet $Q^+ \cup \Sigma \cup \{\leftarrow, \downarrow, \rightarrow, \#\} \cup \{0, 1, ;\} \subseteq \Sigma$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} \mathbf{01;ar...;;q_0\underline{B}\underline{\$}[autres lettres de \Sigma]\# \langle \delta \rangle^{\text{bin}};x}$

Apply: la machine U

- ❖ U : machine I/O à 1 bande de travail, pas de bande de sortie, sur un **alphabet restreint** $\{0,1,;\}$ (et non plus étendu)
- ❖ En entrée: $(\langle M \rangle; x)^{\text{bin}}$

- ❖ $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 01; ar \dots ; ; q_0 \underline{B} \underline{\$} [\Sigma] \# \langle \delta \rangle; x$
- ❖ alphabet $\{0,1,;\}$



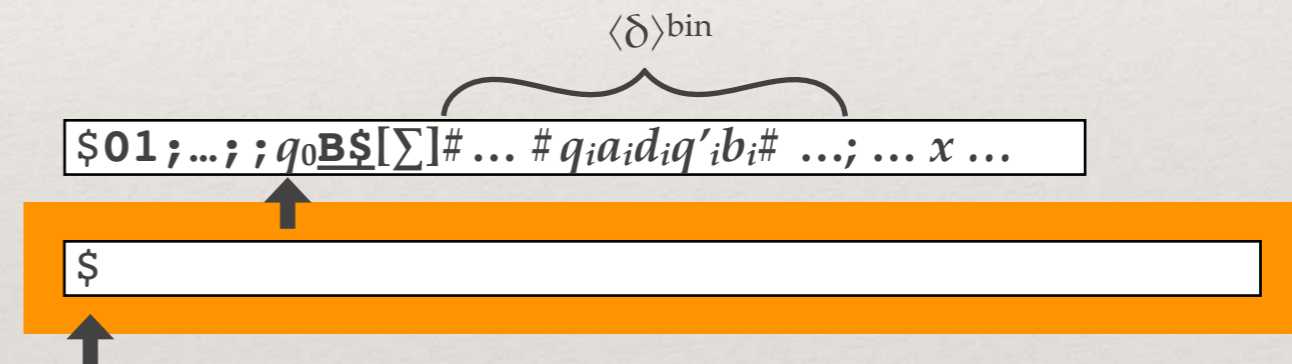
U commence par:

— lire et mémoriser q_0 , \underline{B} , $\underline{\$}$ dans son état interne

Apply: la machine U

- ❖ U : machine I/O à 1 bande de travail, pas de bande de sortie, sur un alphabet restreint $\{0,1,;\}$ (et non plus étendu)
- ❖ En entrée: $(\langle M \rangle; x)^{\text{bin}}$

- ❖ $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 01; a r \dots ; ; q_0 \underline{\mathbf{B}} \underline{\mathbf{\$}} [\Sigma] \# \langle \delta \rangle; x$
- ❖ alphabet $\{0,1,;\}$



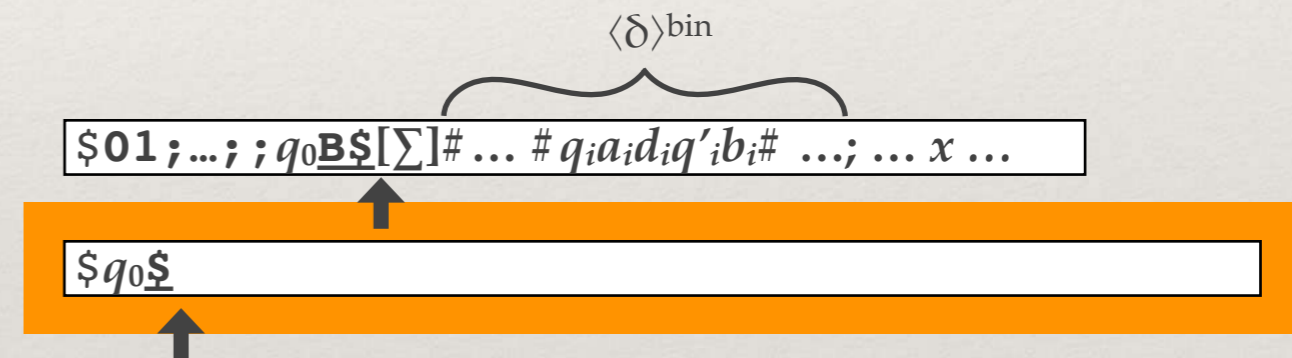
U commence par:

- lire et mémoriser $q_0, \underline{\mathbf{B}}, \underline{\mathbf{\$}}$ dans son état interne (après le $;;$)
- écrire $q_0 \underline{\mathbf{\$}}$ sur la bande de travail
(= tout jusqu'au premier $';$, puis tout entre le 2ème et le 3ème)

Apply: la machine U

- ❖ U : machine I/O à 1 bande de travail, pas de bande de sortie, sur un alphabet restreint $\{0,1,;\}$ (et non plus étendu)
- ❖ En entrée: $(\langle M \rangle; x)^{\text{bin}}$

- ❖ $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 01; \mathbf{ar} \dots ; ; q_0 \mathbf{B} \mathbf{\$} [\Sigma] \# \langle \delta \rangle; x$
- ❖ alphabet $\{0,1,;\}$



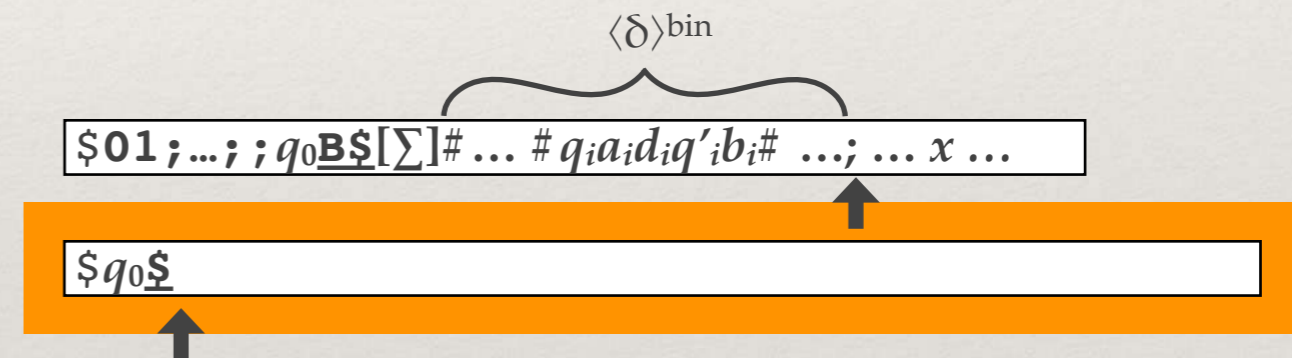
U commence par:

- lire et mémoriser q_0 , \mathbf{B} , $\mathbf{\$}$ dans son état interne (après le $;;$)
- écrire $q_0 \mathbf{\$}$ sur la bande de travail
(= tout jusqu'au premier ' $;$ ', puis tout entre le 2ème et le 3ème)
- avancer jusqu'au $;$ (en gras, = la suite de lettres $\text{bin}_{\Sigma} (;)$)

Apply: la machine U

- ❖ U : machine I/O à 1 bande de travail, pas de bande de sortie, sur un alphabet restreint $\{0,1,;\}$ (et non plus étendu)
- ❖ En entrée: $(\langle M \rangle; x)^{\text{bin}}$

- ❖ $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 01; \mathbf{ar} \dots ; ; q_0 \mathbf{B} \mathbf{\underline{\$}} [\Sigma] \# \langle \delta \rangle; x$
- ❖ alphabet $\{0,1,;\}$



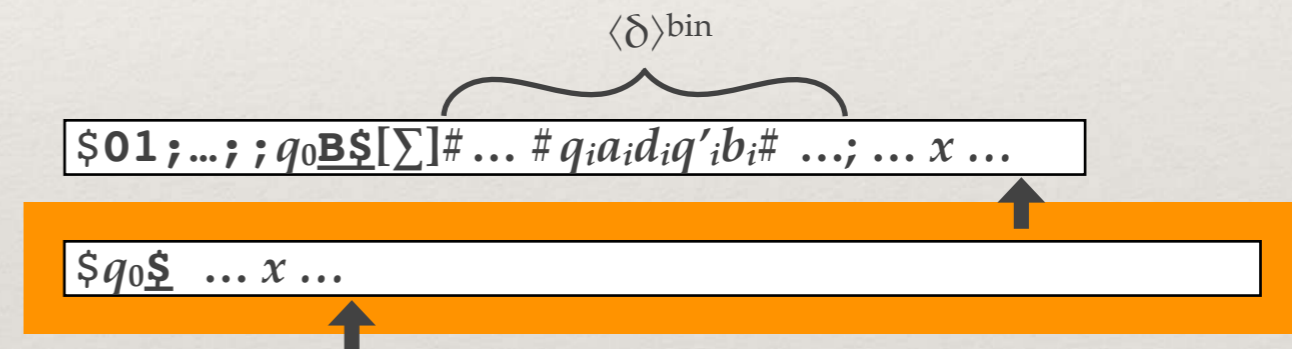
U commence par:

- lire et mémoriser q_0 , \mathbf{B} , $\mathbf{\underline{\$}}$ dans son état interne (après le $;;$)
- écrire $q_0 \mathbf{\underline{\$}}$ sur la bande de travail
(= tout jusqu'au premier ' $;$ ', puis tout entre le 2ème et le 3ème)
- avancer jusqu'au $;$ (en gras, = la suite de lettres $\text{bin}_\Sigma(;;)$)
- recopier x sur la bande de travail

Apply: la machine U

- ❖ U : machine I/O à 1 bande de travail, pas de bande de sortie, sur un alphabet restreint $\{0,1,;\}$ (et non plus étendu)
- ❖ En entrée: $(\langle M \rangle; x)^{\text{bin}}$

- ❖ $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 0 \mathbf{1}; \mathbf{ar} \dots ; ; q_0 \mathbf{B} \mathbf{\$} [\Sigma] \# \langle \delta \rangle; x$
- ❖ alphabet $\{0,1,;\}$



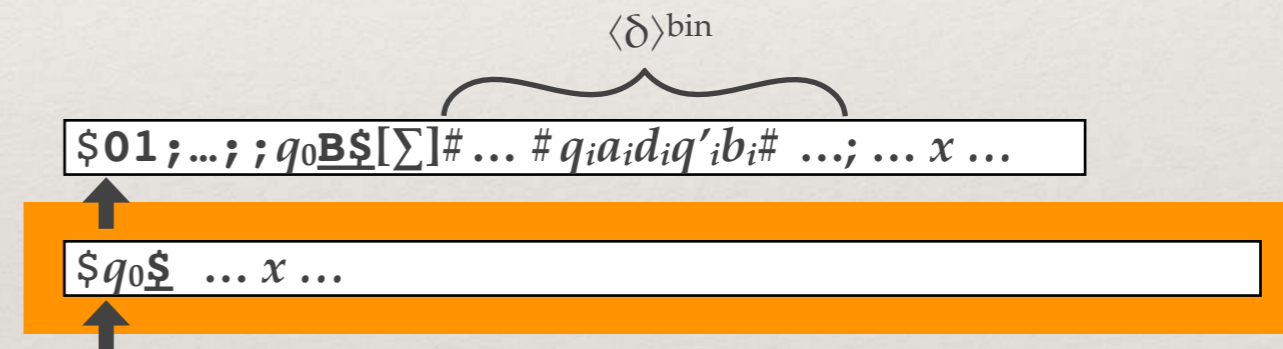
U commence par:

- lire et mémoriser q_0 , \mathbf{B} , $\mathbf{\$}$ dans son état interne (après le $;;$)
- écrire $q_0 \mathbf{\$}$ sur la bande de travail
(= tout jusqu'au premier $';$, puis tout entre le 2ème et le 3ème)
- avancer jusqu'au $;$ (en gras, = la suite de lettres $\text{bin}_{\Sigma} (;)$)
- recopier x sur la bande de travail
- nettoyer: ramener les têtes en début de bande

Apply: la machine U

- ❖ U : machine I/O à 1 bande de travail, pas de bande de sortie, sur un alphabet restreint $\{0,1,;\}$ (et non plus étendu)
- ❖ En entrée: $(\langle M \rangle; x)^{\text{bin}}$

- ❖ $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
- ❖ $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 0 \mathbf{1}; \mathbf{ar} \dots ; ; q_0 \mathbf{B} \mathbf{\$} [\Sigma] \# \langle \delta \rangle; x$
- ❖ alphabet $\{0,1,;\}$

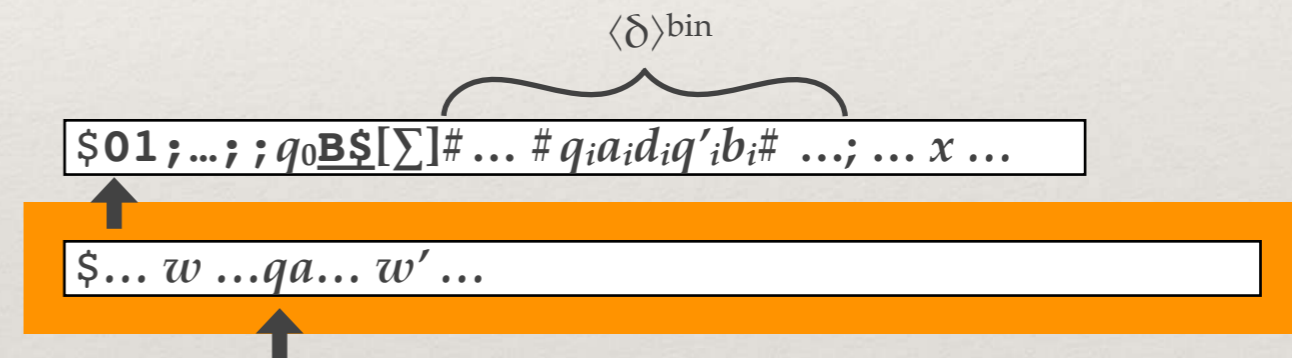


U commence par:

- lire et mémoriser q_0 , \mathbf{B} , $\mathbf{\$}$ dans son état interne (après le $;;$)
- écrire $q_0 \mathbf{\$}$ sur la bande de travail
(= tout jusqu'au premier ';', puis tout entre le 2ème et le 3ème)
- avancer jusqu'au ; (en gras, = la suite de lettres $\text{bin}_{\Sigma'}(;)$)
- recopier x sur la bande de travail
- nettoyer: ramener les têtes en début de bande (après les \$)

Apply: la machine U

- ❖ A toute étape de la simulation de M (en entrée) par U , si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de U contient $\$wqw'$ [en gras=codé sur 0,1,;] [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

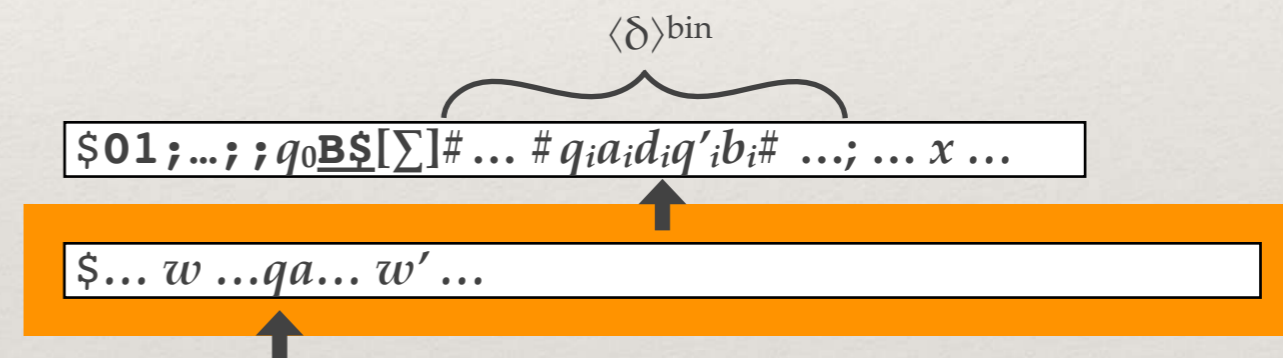


Pour simuler une étape de M , U :

— lit q et a sur la bande de travail, et les mémorise dans son état interne

Apply: la machine U

- ❖ A toute étape de la simulation de M (en entrée) par U , si la configuration de M est (w, q, w') [ww' commençant par $\$$] alors la bande de travail de U contient $\$wqw'$ [en gras=codé sur 0,1,;] [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



Pour simuler une étape de M , U :

- parcourt son entrée jusqu'à trouver $\# q_i a_i q' i b_i d_i$ avec $q_i a_i = q a$ et mémorise $d_i, q' i, b_i$ dans son état interne (d_i sur 2 bits+';')

il suffit de comparer en parallèle

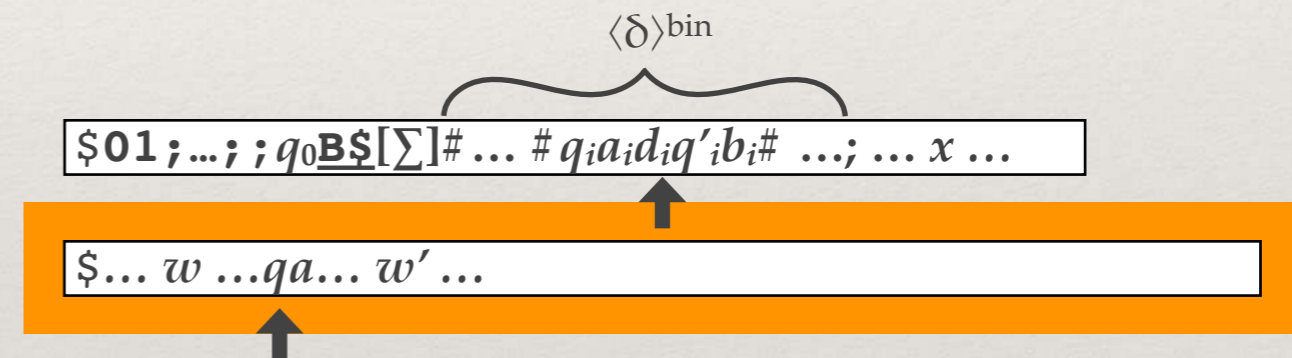
$q a = \text{bin}_{\Sigma'}(q); \text{bin}_{\Sigma'}(a)$; avec $q_i a_i = \text{bin}_{\Sigma'}(q_i); \text{bin}_{\Sigma'}(a_i)$;

- si on atteint le 2ème ';' en même temps, on a trouvé
- sinon, avancer la tête d'entrée jusqu'au prochain # (=bin $_{\Sigma'}(\#)$;) et ramener la tête de travail 2 ';' en arrière

si pas de a (i.e., un seul ';', pas 2)
ou plus généralement si $a = \mathbf{B}$,
on fait comme si $a = \mathbf{B}$

Apply: la machine U

- ❖ A toute étape de la simulation de M (en entrée) par U , si la configuration de M est (w, q, w') [ww' commençant par $\$$] alors la bande de travail de U contient $\$wqw'$ [en gras=codé sur 0,1,;] [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

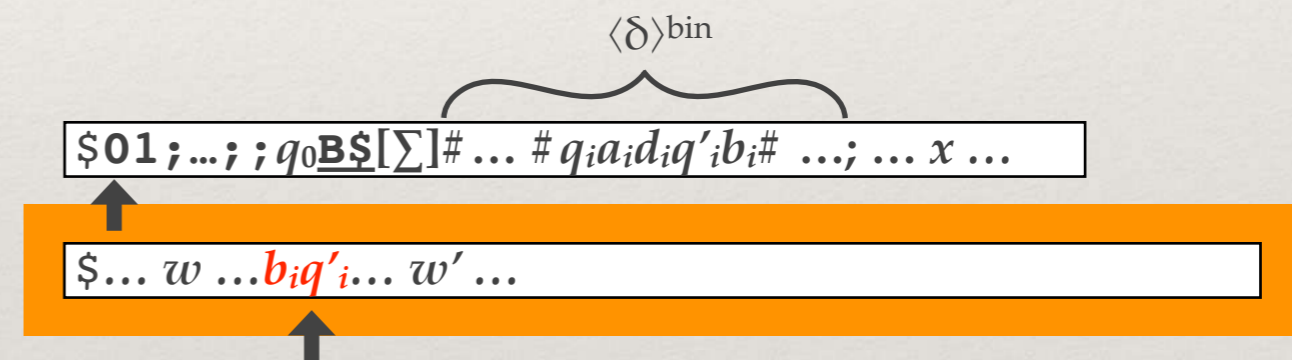


Pour simuler une étape de M , U :

- parcourt son entrée jusqu'à trouver $\# q_i a_i q' i b_i d_i$ avec $q_i a_i = q a$ et mémorise $d_i, q' i, b_i$ dans son état interne (d_i sur 2 bits+';')
- si $d_i = \rightarrow$, on recopie $b_i q' i$ et on laisse la tête sous $q' \dots$ en faisant des allers-retours entre les zones avoisinantes délimitées par ';'.

Apply: la machine U

- ❖ A toute étape de la simulation de M (en entrée) par U , si la configuration de M est (w, q, w') [ww' commençant par $\$$] alors la bande de travail de U contient $\$wqw'$ [en gras=codé sur 0,1,;] [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]

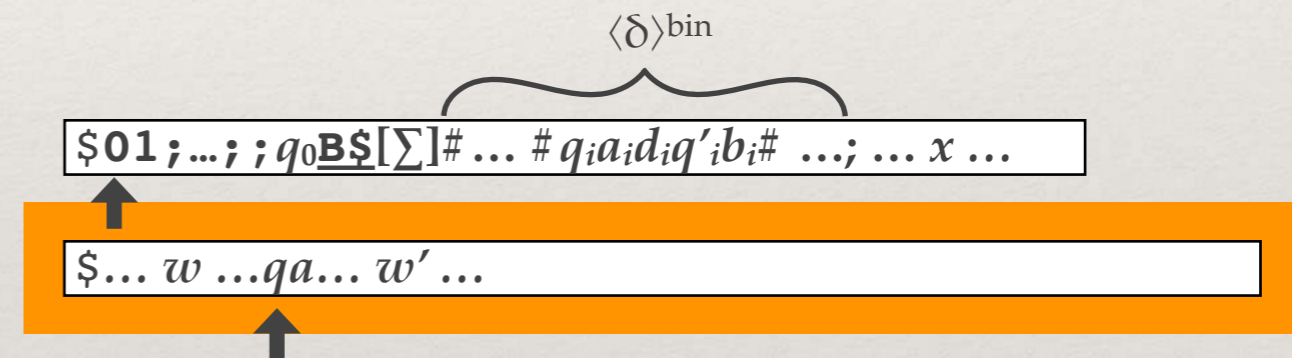


Pour simuler une étape de M , U :

- parcourt son entrée jusqu'à trouver $\# q_i a_i q' i b_i d_i$ avec $q_i a_i = q a$ et mémorise $d_i, q' i, b_i$ dans son état interne (d_i sur 2 bits+';')
- si $d_i = \rightarrow$, on recopie $b_i q' i$ et on laisse la tête sous $q' \dots$ en faisant des allers-retours entre les zones avoisinantes délimitées par ';'.

Apply: la machine U

- ❖ A toute étape de la simulation de M (en entrée) par U , si la configuration de M est (w, q, w') [ww' commençant par $\underline{\$}$] alors la bande de travail de U contient $\$wqw'$ [en gras=codé sur 0,1,;] [avec: tête à droite du $\$$ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



Pour simuler une étape de M , U :

- parcourt son entrée jusqu'à trouver $\# q_i a_i q' i b_i d_i$ avec $q_i a_i = q a$ et mémorise $d_i, q' i, b_i$ dans son état interne (d_i sur 2 bits+;')
- si $d_i = \leftarrow$ ou $d_i = \downarrow$, exercice
- la machine U accepte si $q = \text{bin}_{\Sigma'}(\text{accept})$; rejette si $q = \text{bin}_{\Sigma'}(\text{reject})$;

La machine universelle U

- ❖ Rappel: $(\langle M \rangle; x)^{\text{bin}} \stackrel{\text{def}}{=} 01; ar...;; q_0 \underline{B} \underline{\$} [\text{autres lettres de } \Sigma] \# \langle \delta \rangle^{\text{bin}}; x$
- ❖ Nous avons construit une machine **universelle** U , i.e.:
- ❖ **Théorème.** Pour tout mT M (sur un alphabet Σ quelconque), pour toute entrée $x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*$,
 - U accepte $(\langle M \rangle; x)^{\text{bin}}$ ssi M accepte x
 - U rejette $(\langle M \rangle; x)^{\text{bin}}$ ssi M rejette x .
- ❖ L'alphabet de U est $\{0, 1, ,, , \underline{B}, \underline{\$}\}$

kwote

- ❖ Rappel: $(\langle M \rangle ; x)^{\text{bin}} \stackrel{\text{def}}{=} \langle M \rangle^{\text{bin}} ; x = \langle M \rangle^{\text{bin}} \text{bin}_{\Sigma'} (;x)$
où $\langle M \rangle^{\text{bin}} \stackrel{\text{def}}{=} \mathbf{01;ar...;;q_0\underline{B}\$[autres lettres de \Sigma]\# \langle \delta \rangle^{\text{bin}}$
est un mot sur $\{0,1,;\}$
- ❖ On définit **kwote** de sorte que $\text{kwote} (; \langle M \rangle^{\text{bin}}) = \text{bin}_{\Sigma'} (; \langle M \rangle^{\text{bin}})$
- ❖ Subtilité: non, on ne pas juste appliquer $\text{bin}_{\Sigma'}$:
on ne connaît **pas** Σ' , qui est décrit dans le codage $\langle M \rangle^{\text{bin}}$

kwote

- ❖ Rappel: $(\langle M \rangle ; x)^{\text{bin}} \stackrel{\text{def}}{=} \langle M \rangle^{\text{bin}} ; x = \langle M \rangle^{\text{bin}} \text{bin}_{\Sigma'} (; x)$
où $\langle M \rangle^{\text{bin}} \stackrel{\text{def}}{=} \mathbf{01;ar...;;q_0\underline{B}\$}$ [autres lettres de Σ]# $\langle \delta \rangle^{\text{bin}}$
est un mot sur $\{0,1,;\}$
- ❖ On définit `kwote` de sorte que $\text{kwote} (; \langle M \rangle^{\text{bin}}) = \text{bin}_{\Sigma'} (; \langle M \rangle^{\text{bin}})$
- ❖ `kwote` lit son entrée w
si w est de la forme $;w_0;w_1;w_;;...;;w'$ où $w_0, w_1, w_;, \dots, w'$ sont en binaire,
`kwote` écrit w où:
 - les 0 sont remplacés par w_0
 - les 1 sont remplacés par w_1
 - les ; sont remplacés par $w_;$(sinon, `kwote` répond n'importe quoi — ce ne sera pas important)

```
kwote(;000;001;010;...;;0110) =  
010 000 000 000 010  
   000 000 001 010  
   000 001 000 010 ... 010 010  
   000 001 001 000
```

Le langage L_U

- ❖ $L_U \stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \text{bin}_{\Sigma'} (;x) \mid M \text{ mT sur un alphabet } \Sigma, x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*, \text{ tels que } M \text{ accepte } x \}$

❖ **Proposition.** L_U est r.e.

❖ *Preuve.* C'est le langage accepté par U

Théorème. Pour tout mT M (sur un alphabet Σ quelconque), pour toute entrée $x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*$,
 — U accepte $(\langle M \rangle ; x)^{\text{bin}}$ ssi M accepte x
 — U rejette $(\langle M \rangle ; x)^{\text{bin}}$ ssi M rejette x .

❖ ... ou presque: on doit aussi vérifier le **format** de l'entrée, qui doit être $w_0;w_1;w_;;...;;w';x$ avec $w_0, w_1, w_;; \dots, w', x$ en binaire,

❖ où $w' = \text{bin}_{\Sigma'}(\langle M \rangle)$ pour une mT M → [la liste des codes $w_0;w_1;w_;;...;;$ sert à inverser $\text{bin}_{\Sigma'}$]

❖ $\langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \dots \# q_m a_m d_m q'_m b_m$
 ❖ $\langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{B} \underline{\$} [\Sigma] \# \langle \delta \rangle$

❖ et x code un mot sur $\Sigma - \{\underline{B}, \underline{\$}\}$ (lettres listées au début de w)

Cette vérification de format est donc décidable

Le problème de l'acceptation

❖ $L_U \stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \text{ bin}_{\Sigma'} (;x) \mid M \text{ mT sur un alphabet } \Sigma, x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*, \text{ tels que } M \text{ accepte } x \}$

❖ **Théorème.** L_U est indécidable.

❖ *Preuve* (1/2). Sinon, le **complémentaire** $\complement L_U$ de L_U est décidable, par une mT M_N (qui termine toujours)

❖ Soit D la mT qui prend en entrée un mot w (en binaire), vérifie que w est du format $\langle M \rangle^{\text{bin}}$ (rejette sinon) puis applique M_N sur w $\text{kwote} (;w)$

❖ $L(D) = \{ \langle M \rangle^{\text{bin}} \mid \langle M \rangle^{\text{bin}} \text{ bin}_{\Sigma'} (; \langle M \rangle^{\text{bin}}) \notin L_U \}$ $(\text{kwote} (; \langle M \rangle^{\text{bin}}) = \text{bin}_{\Sigma'} (; \langle M \rangle^{\text{bin}}))$
 $= \{ \langle M \rangle^{\text{bin}} \mid M \text{ n'accepte pas } \langle M \rangle^{\text{bin}} \}$

Le problème de l'acceptation

- ❖ $L_U \stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \text{ bin}_{\Sigma'} (;x) \mid M \text{ mT sur un alphabet } \Sigma, x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*, \text{ tels que } M \text{ accepte } x \}$
- ❖ **Théorème.** L_U est indécidable.
- ❖ *Preuve (2/2).* Sinon, le **complémentaire** $\complement L_U$ de L_U est décidable, par une mT M_N (qui termine toujours)
- ❖ $L(D) = \{ \langle M \rangle^{\text{bin}} \mid M \text{ n'accepte pas } \langle M \rangle^{\text{bin}} \}$
- ❖ $\langle D \rangle^{\text{bin}} \in L(D)$ ssi D n'accepte pas $\langle D \rangle^{\text{bin}}$ (déf. de $L(D)$, ci-dessus)
ssi $\langle D \rangle^{\text{bin}} \notin L(D)$ (déf.: langage d'une mT)

Contradiction. \square

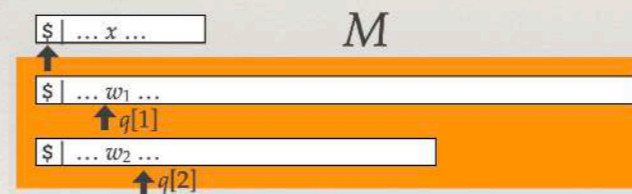
Le problème de l'acceptation

- ❖ $L_U \stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \text{bin}_{\Sigma'} (;x) \mid M \text{ mT sur un alphabet } \Sigma, x \in (\Sigma - \{\underline{B}, \underline{\$}\})^*, \text{ tels que } M \text{ accepte } x \}$

❖ **Théorème.** L_U n'est même pas co-r.e.

❖ *Preuve.* L_U est r.e.

S'il était aussi co-r.e., il serait récursif (décidable). \square



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

Le problème de l'arrêt

- ❖ **Halt** $\stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma, \text{ telle que } M \text{ termine en partant du mot vide } \varepsilon \}$
- ❖ **Théorème.** Halt est indécidable, r.e., et donc pas co-r.e.
- ❖ *Preuve (1/2).* Le fait que Halt soit r.e. se montre comme pour L_U . On définit une machine M_{Halt} qui, sur son entrée:
 - ❖ vérifie qu'elle est du format $\langle M \rangle^{\text{bin}}$
 - ❖ simule M sur l'entrée ε
 - ❖ si jamais M s'arrête, va en **accept** (M_{Halt} ne va jamais en **reject**)
- ❖ Alors $L(M_{\text{Halt}}) = \mathbf{Halt}$.

Le problème de l'arrêt

- ❖ **Halt** $\stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma, x \in (\Sigma - \{ \underline{B}, \underline{\$} \})^*, \text{ tels que } M \text{ termine en partant du mot vide } \varepsilon \}$

- ❖ **Théorème.** Halt est indécidable, r.e., et donc pas co-r.e.

- ❖ *Preuve (2/2).* Supposons **Halt décidé** par une mT M

- ❖ On décide $L_U \stackrel{\text{def}}{=} \{ \langle M \rangle^{\text{bin}} \text{ bin}_{\Sigma'} (;x) \mid M \text{ accepte } x \}$ comme suit:

- ❖ on vérifie le format d'entrée $\langle M \rangle^{\text{bin}} \text{ bin}_{\Sigma'} (;x)$

- ❖ on construit (le code de) une nouvelle mT M_x qui commence par écrire x (mémemorisé dans son état interne) sur sa bande, puis simule M ; si M rejette, alors M_x **boucle** (~forever-loop)

- ❖ On a $\langle M \rangle^{\text{bin}} \text{ bin}_{\Sigma'} (;x) \in L_U$ ssi $\langle M_x \rangle^{\text{bin}} \in \mathbf{Halt}$: non, car L_U **indécidable**. \square

Ceci est une
preuve par
réduction
(de L_U vers **Halt**)

La prochaine fois

La prochaine fois

- ❖ Réductions:
« si $L \rightarrow L'$ et L' est indécidable, alors L aussi »
- ❖ Problème de correspondance de Post
- ❖ Théorème de Rice