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

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
- `(eval ‘`M`)` $\rightarrow^* M$

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
- `(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

- `(eval '`M`)` $\rightarrow^*$ M

eval, apply, make-apply, list

- `(eval '`M`)` $\rightarrow^*$ M
- `(apply '`M `'`N`)` $\rightarrow^*$ $(M\ N)$

  [petit mensonge: en fait `(apply '`M `'(`$N_1 \ldots N_k$`))` $\rightarrow^*$ $(M\ N_1 \ldots N_k)$]

eval, apply, make-apply, list

- `(eval ‘`M`)` $\rightarrow^*$ M
- `(apply ‘`M `’`N`)` $\rightarrow^*$ `(`$M\,N$`)`
  [petit mensonge: en fait `(apply ‘`M `’(`$N_1 \ldots N_k$`))` $\rightarrow^*$ `(`$M\,N_1 \ldots N_k$`)`]
- On va supposer une primitive `make-apply` telle que
  `(make-apply ‘`M `’`N`)` $\rightarrow^*$ `‘(`$M\,N$`)`

eval, apply, make-apply, list

- `(eval ‘`M`)` $\rightarrow^*$ M
- `(apply ‘`M `’`N`)` $\rightarrow^*$ `(`$M\,N$`)`
  [petit mensonge: en fait `(apply ‘`M `’(`$N_1 \ldots N_k$`))` $\rightarrow^*$ `(`$M\,N_1 \ldots N_k$`)`]
- On va supposer une primitive `make-apply` telle que
  `(make-apply ‘`M `’`N`)` $\rightarrow^*$ `‘(`$M\,N$`)`
- En Lisp, `make-apply=list`

eval, apply, make-apply, list

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

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 → données)

telle que `(kwote` *a*`)` →* le `quote` de la valeur de *a*

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 → données)
  telle que `(kwote` *a*`)` →* le `quote` de la valeur de *a*
- Par exemple,
  `(kwote (cons 1 (cons 2 nil)))` →* `‘(1 2)`

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 → données)
  telle que `(kwote` *a*`)` →* le `quote` de la valeur de *a*
- Par exemple,
  `(kwote (cons 1 (cons 2 nil)))` →* `‘(1 2)`
- Aussi, `(kwote ‘`*M*`)` →* `‘‘`*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

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))
```

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)))`

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)))`

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

```
(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)))`

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

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

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

(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)))`

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)))`

(1) si `(turing ‘turing)` ne termine **pas**,
`(haltp ‘(turing ‘turing))` →* `t`

- →* `(cond ((haltp ‘(turing ‘turing))`
  `(forever-loop))`
  `(t 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

```
(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)))`

(1) si `(turing ‘turing)` ne termine **pas**,
`(haltp ‘(turing ‘turing))` →* `t`

- →* `(cond ((haltp ‘(turing ‘turing))`
  `(forever-loop))`
  `(t nil)))`

(2) alors on termine sur `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

```
(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.

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: $\langle M \rangle$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\mathtt{B}}, \underline{\mathtt{\$}})$
- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  et on pose $(q'_i,b_i,d_i) \stackrel{\text{def}}{=} \delta(q_i,a_i)$, $1 \leq i \leq m$

Quote: $\langle M \rangle$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\mathtt{B}}, \underline{\mathtt{\$}})$

Pour le moment, on **fixe** Q, $\sum$

- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  et on pose $(q'_i,b_i,d_i) \stackrel{\text{def}}{=} \delta(q_i,a_i)$, $1 \leq i \leq m$

Quote: $\langle M \rangle$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\mathtt{B}}, \underline{\$})$

Pour le moment, on **fixe** Q, $\sum$

- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  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

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

  sur l'alphabet $Q^+ \uplus \sum \uplus \{\leftarrow, \downarrow, \rightarrow, \#\}$

Quote: $\langle M \rangle$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\mathtt{B}}, \underline{\mathtt{\$}})$

Pour le moment, on **fixe** Q, $\sum$

- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  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
  $\langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
  sur l'alphabet $Q^+ \uplus \sum \uplus \{\leftarrow, \downarrow, \rightarrow, \#\}$

- On pose $\langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathtt{B\$}}[\text{autres lettres de } \sum] \# \langle \delta \rangle$

Quote: $\langle M \rangle$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\mathtt{B}}, \underline{\mathtt{\$}})$

Pour le moment, on **fixe** Q, $\sum$

- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  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
  $\langle \delta \rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
  sur l'alphabet $Q^+ \uplus \sum \uplus \{\leftarrow, \downarrow, \rightarrow, \#\}$

- On pose $\langle M \rangle \stackrel{\text{def}}{=} q_0 \underline{\mathtt{B\$}}[\text{autres lettres de } \sum]\#\langle \delta \rangle$

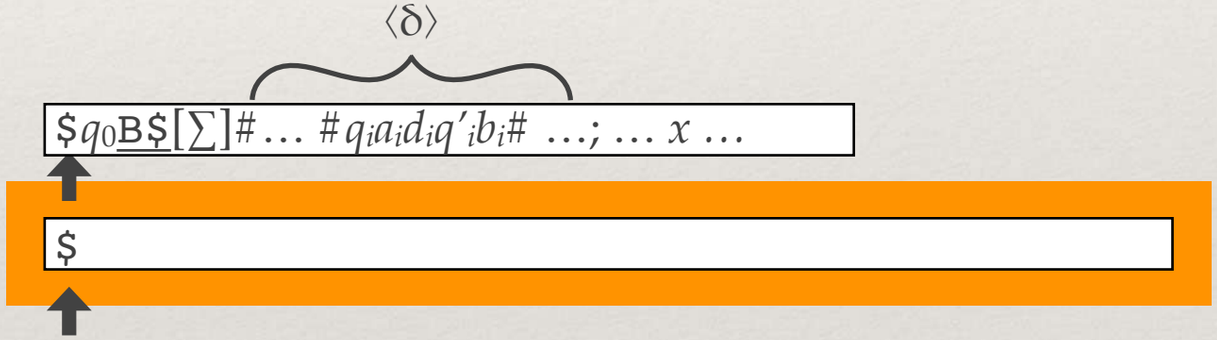
Pas exactement le codage
du théorème 6.5.1 du poly

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



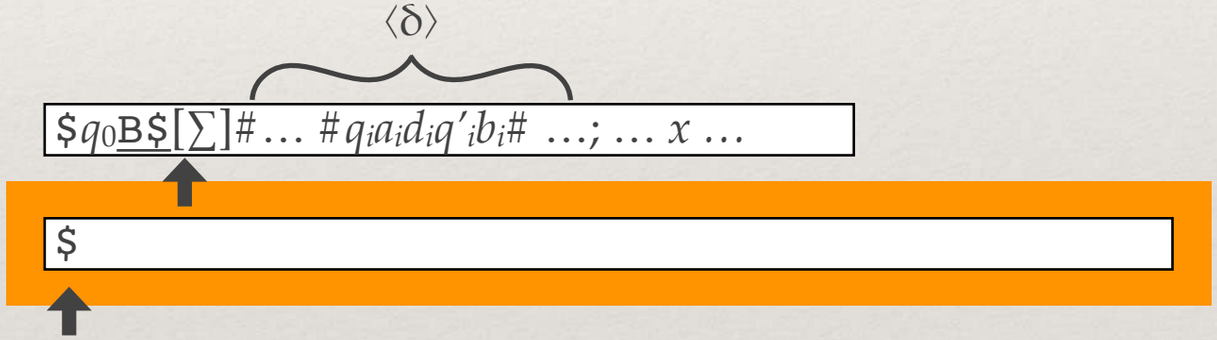
$U_{Q\Sigma}$ commence par:
— lire et mémoriser q_0, $\underline{\texttt{B}}$, $\underline{\texttt{\$}}$ dans son état interne

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B}}\underline{\texttt{\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



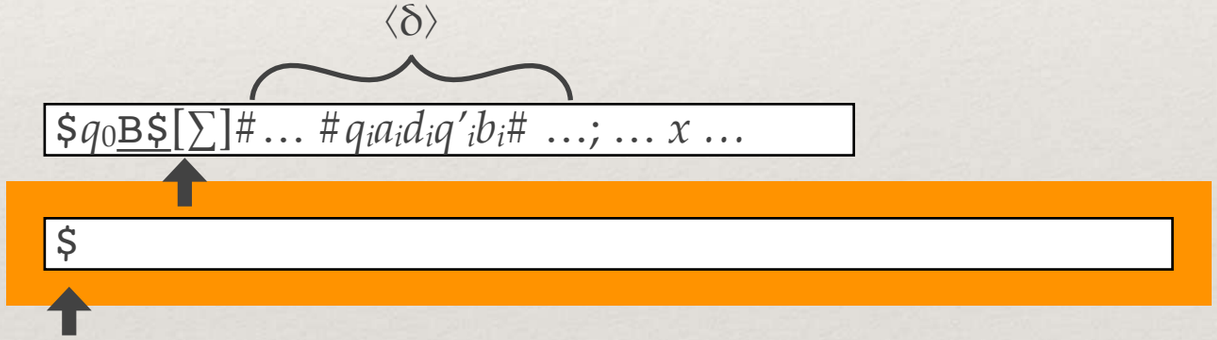
$U_{Q\Sigma}$ commence par:
— lire et mémoriser q_0, $\underline{\texttt{B}}$, $\underline{\texttt{\$}}$ dans son état interne

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

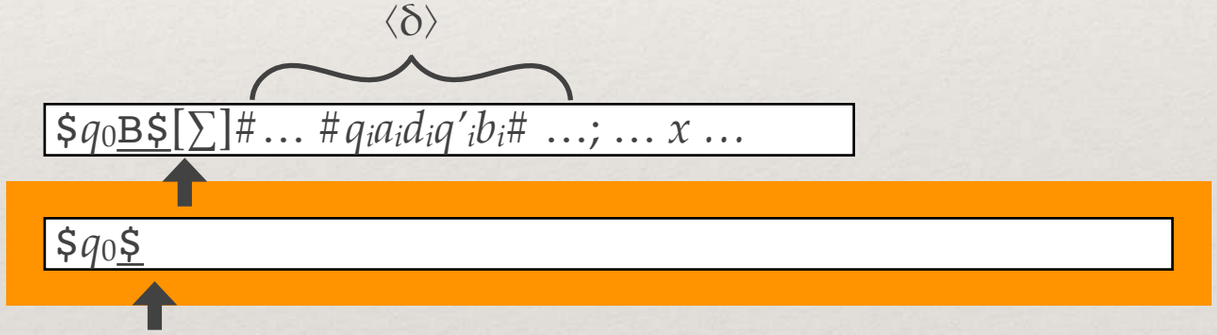
— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

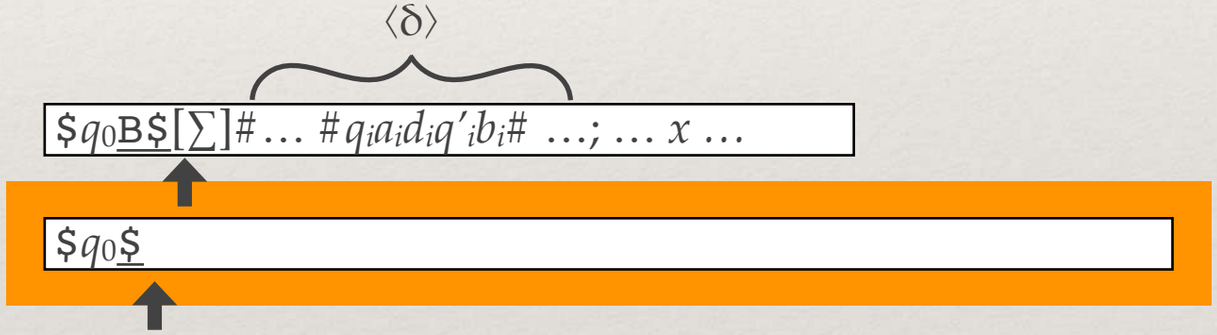
— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

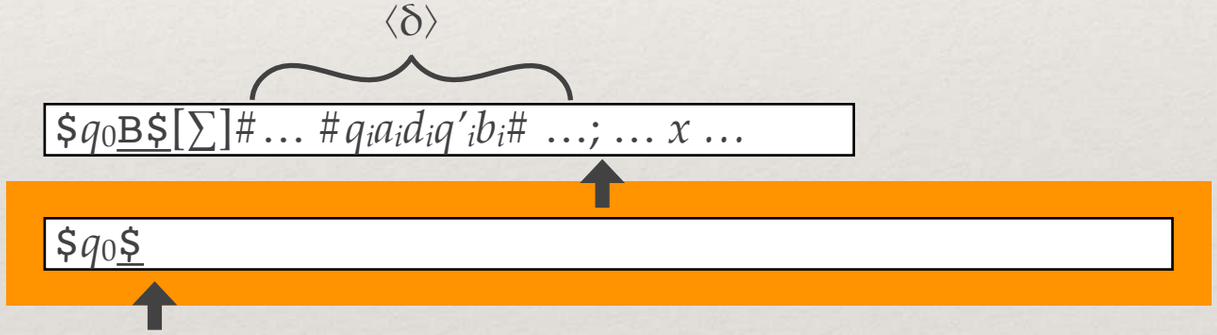
— avancer jusqu'au ;

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

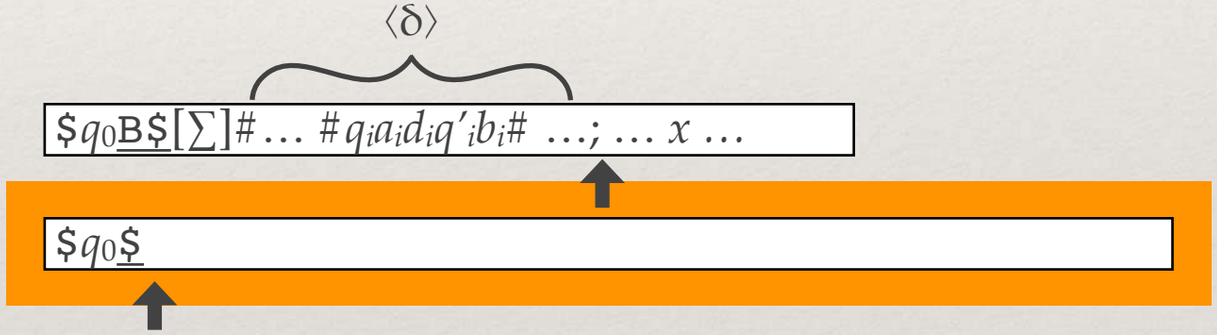
— avancer jusqu'au ;

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

— avancer jusqu'au ;

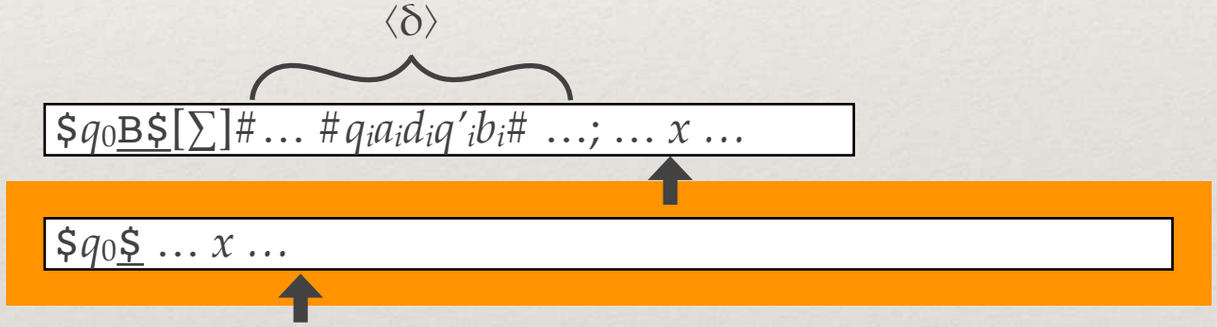
— recopier x sur la bande de travail

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

— avancer jusqu'au ;

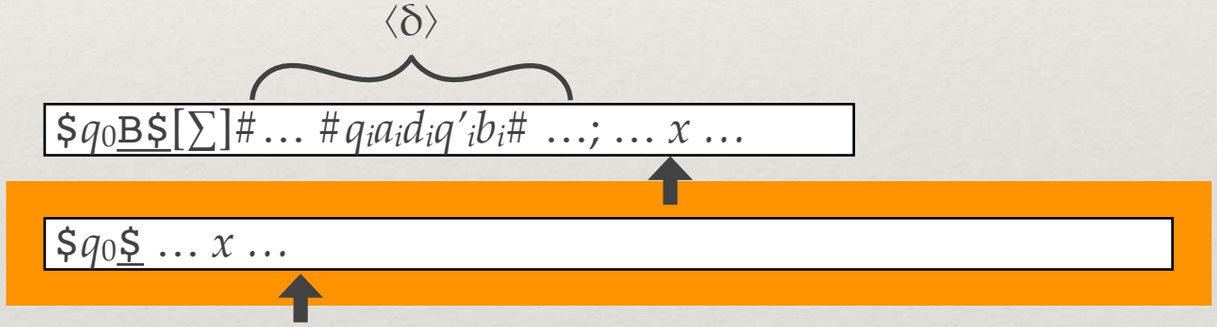
— recopier x sur la bande de travail

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

— avancer jusqu'au ;

— recopier x sur la bande de travail

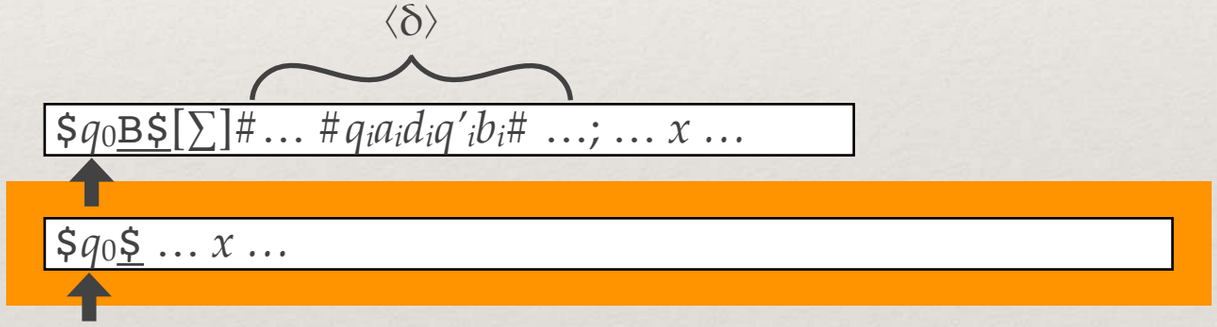
— nettoyer: ramener les têtes en début de bande

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1 a_1 d_1 q'_1 b_1 \# q_2 a_2 d_2 q'_2 b_2 \# \ldots \# q_m a_m d_m q'_m b_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0 \underline{\texttt{B\$}}[\sum]\#\langle\delta\rangle$
- alphabet $Q^+ \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$

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

- En entrée: $\langle M\rangle$; x



$U_{Q\Sigma}$ commence par:

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

— écrire q_0, $\underline{\texttt{\$}}$ sur la bande de travail

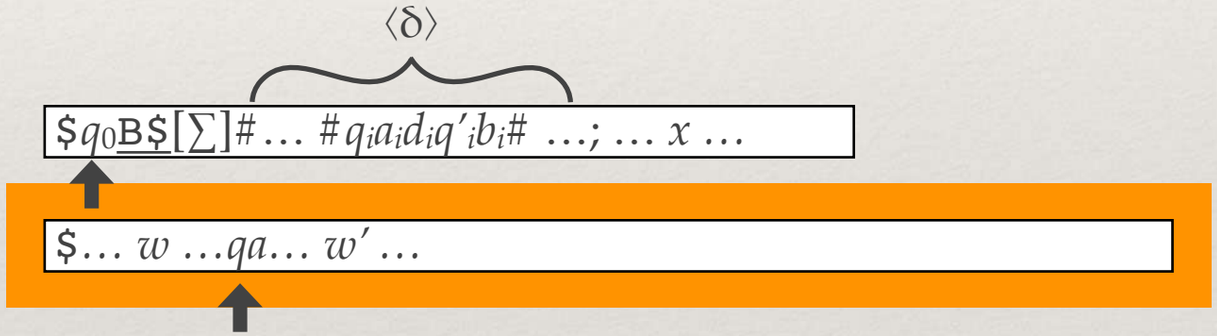
— avancer jusqu'au ;

— recopier x sur la bande de travail

— nettoyer: ramener les têtes en début de bande (après les $)

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

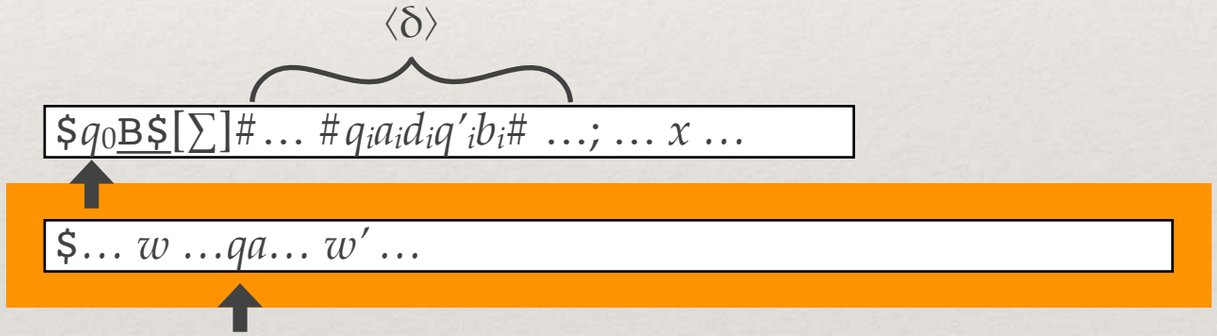
- A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$,
  si la configuration de M est (w,q,w') [ww' commençant par $\underline{\$}$]
  alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$
  [avec: tête à droite du $ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



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

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 $\underline{\texttt{\$}}$]
  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]

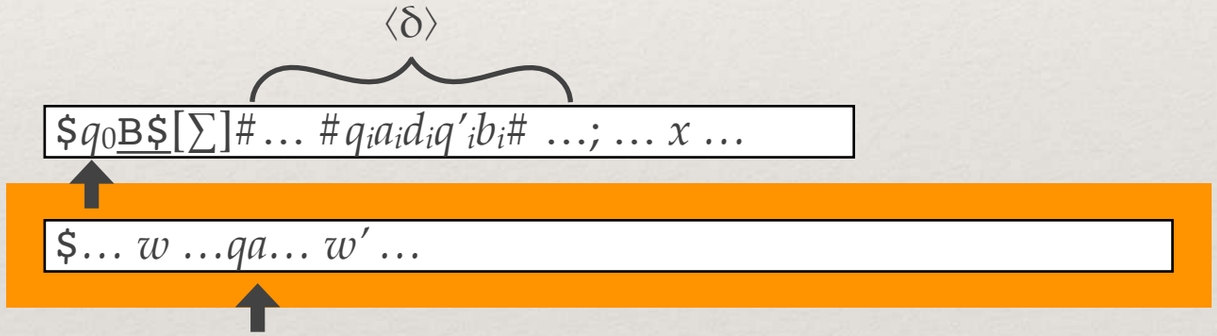


Pour simuler une étape de M, $U_{Q\Sigma}$:
— lit q et a sur la bande de travail, et les mémorise dans son état interne

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

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

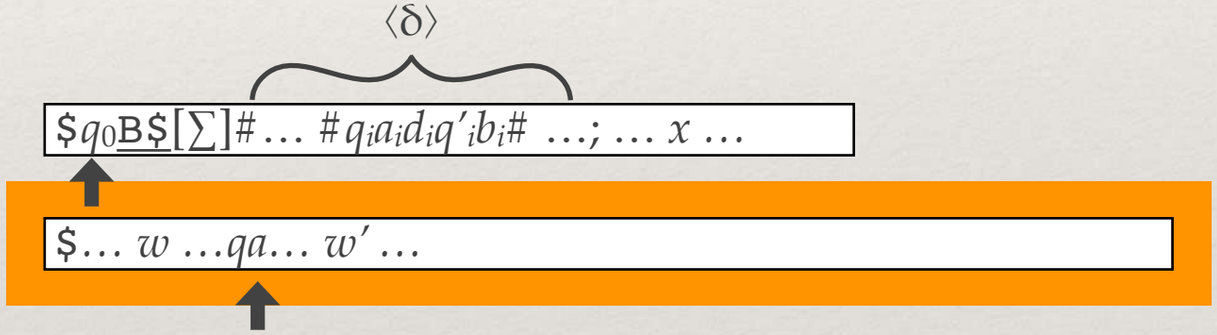
- A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$,
  si la configuration de M est (w,q,w') [ww' commençant par $\underline{\$}$]
  alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$
  [avec: tête à droite du $ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



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

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

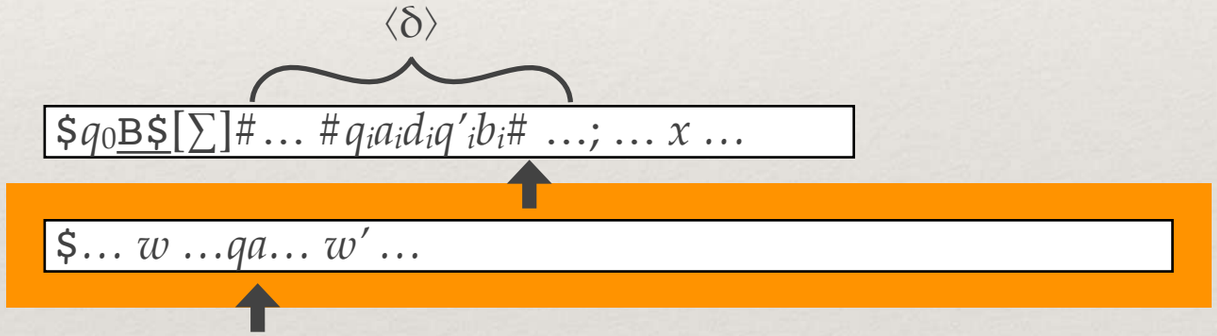
- A toute étape de la simulation de M (en entrée) par $U_{Q\Sigma}$,
  si la configuration de M est (w,q,w') [ww' commençant par $\underline{\$}$]
  alors la bande de travail de $U_{Q\Sigma}$ contient $\$wqw'$
  [avec: tête à droite du $ sur bande d'entrée; tête sur la lettre q sur la bande de travail]



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 $\#q_ia_id_iq'_ib_i$ 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, $U_{Q\Sigma}$:

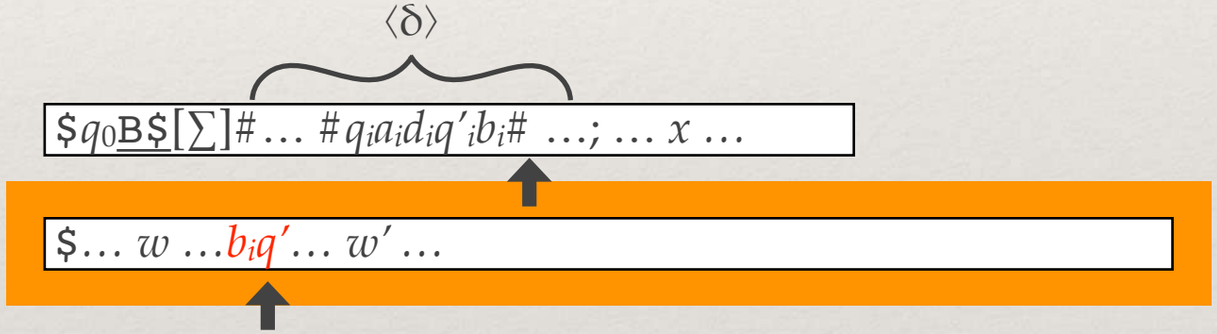
— lit q et a sur la bande de travail, et les mémorise dans son état interne

— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ 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 $\$wqw'$
  [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, $U_{Q\Sigma}$:

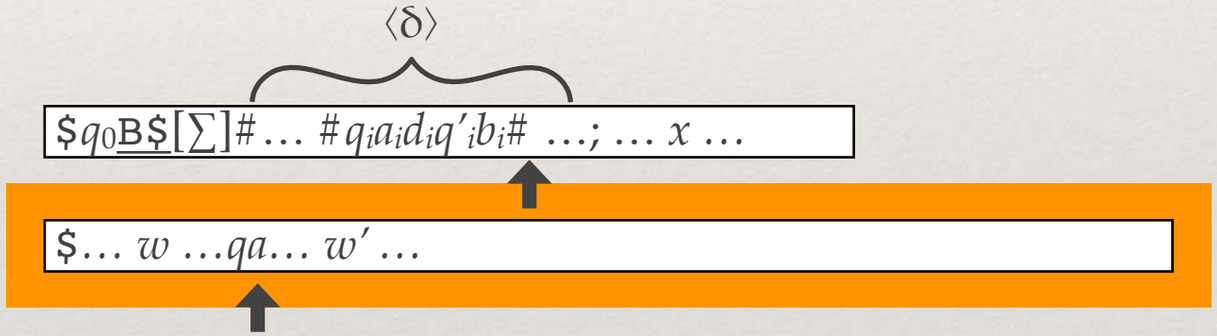
— lit q et a sur la bande de travail, et les mémorise dans son état interne

— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ 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, $U_{Q\Sigma}$:

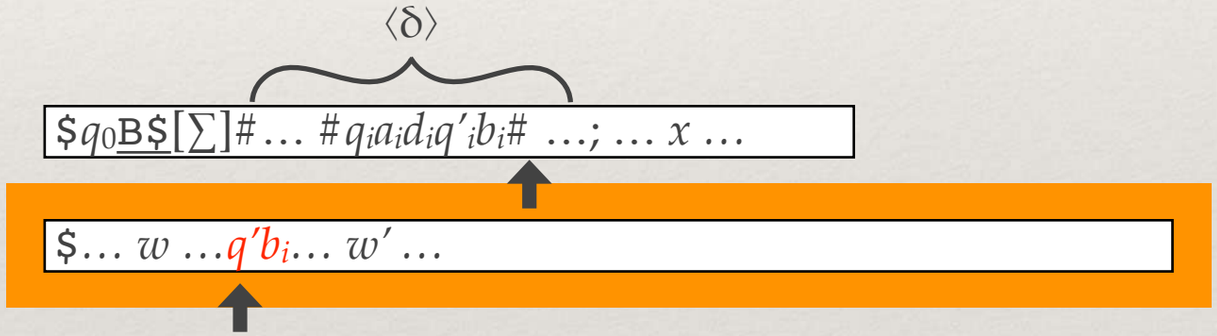
— lit q et a sur la bande de travail, et les mémorise dans son état interne

— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ 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, $U_{Q\Sigma}$:

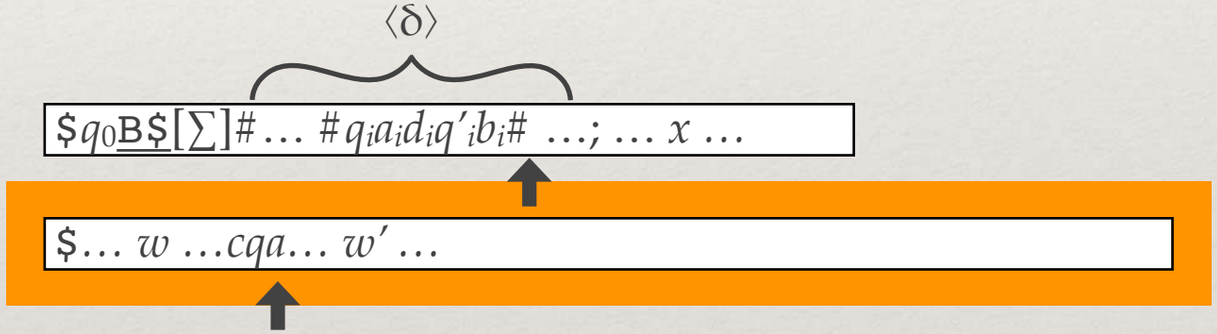
— lit q et a sur la bande de travail, et les mémorise dans son état interne

— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ 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, $U_{Q\Sigma}$:

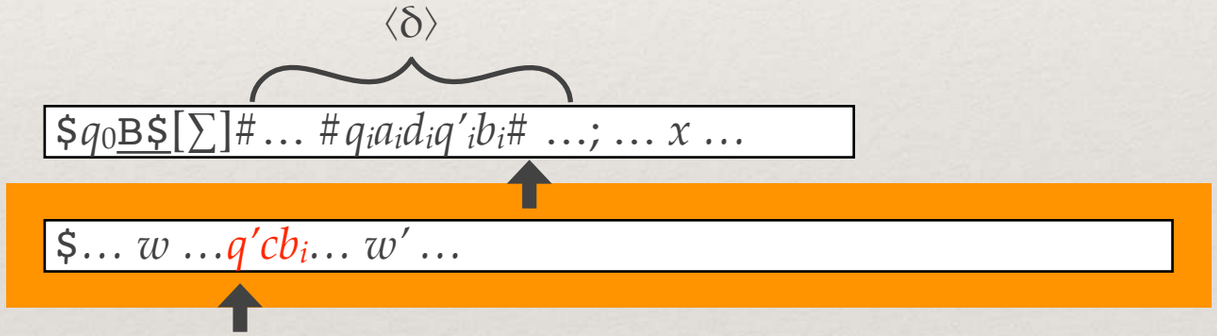
— lit q et a sur la bande de travail, et les mémorise dans son état interne

— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ 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, $U_{Q\Sigma}$:

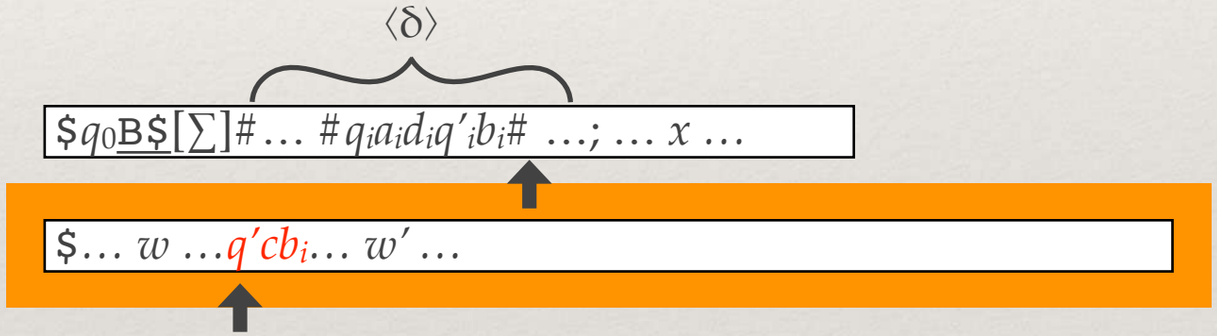
— lit q et a sur la bande de travail, et les mémorise dans son état interne

— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ 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'

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

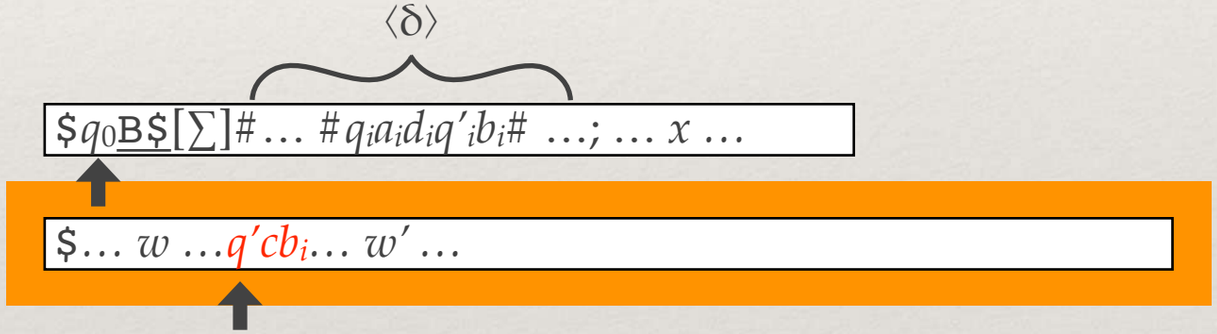
— parcourt son entrée jusqu'à trouver $\#q_ia_id_iq'_ib_i$ avec $q_i=q$, $a_i=a$
et mémorise d_i, q'_i, b_i, dans son état interne

— 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 $\#q_ia_id_iq'_ib_i$ 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 $q' \neq$ **accept**, **reject**
- Sinon, $U_{Q\Sigma}$ accepte, ou rejette, selon le cas

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

- Dernier point: on fait tout ceci uniquement si $q' \neq$ **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 $x \in (\Sigma - \{\underline{\mathtt{B}}, \underline{\mathtt{\$}}\})^*$,
— $U_{Q\Sigma}$ accepte $\langle M \rangle$; x ssi M accepte x
— $U_{Q\Sigma}$ rejette $\langle M \rangle$; x ssi M rejette x.

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

- Dernier point: on fait tout ceci uniquement si $q' \neq$ **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 $x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
  — $U_{Q\Sigma}$ 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 $\Sigma' \stackrel{\text{def}}{=} Q^+ \uplus \Sigma \uplus \{\leftarrow, \downarrow, \rightarrow, \#, \texttt{B}, \texttt{\$}\}$, pas Σ

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

- Dernier point: on fait tout ceci uniquement si $q' \neq$ **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 $x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
  — $U_{Q\Sigma}$ 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 $\Sigma' \stackrel{\text{def}}{=} Q^+ \uplus \Sigma \uplus \{\leftarrow, \downarrow, \rightarrow, \#, \texttt{B}, \texttt{\$}\}$, pas Σ
- Pour corriger ce problème…

Réduire à deux lettres (plus ;)

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

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

Réduire à deux lettres (plus ;)

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

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

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

Réduire à deux lettres (plus ;)

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

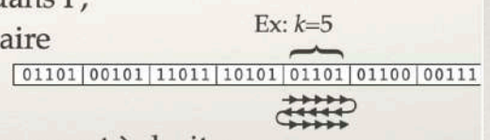
$$(\langle M\rangle;x)^{\text{bin}} \triangleq \text{bin}_{\Sigma'}(0);\text{bin}_{\Sigma'}(1);\text{bin}_{\Sigma'}(;);\text{bin}_{\Sigma'}(\textbf{accept});\text{bin}_{\Sigma'}(\textbf{reject});\ldots;;\text{bin}_{\Sigma'}(\langle M\rangle;x)$$

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

- La machine U **simule** chaque étape de calcul de U_Σ comme avant…

Réduire le nombre de lettres

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



- 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]

Réduire à deux lettres (plus ;)

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

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

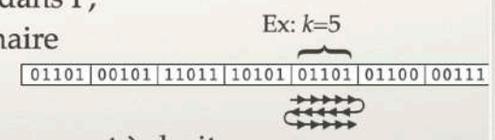
Point subtil: la fonction $\text{bin}_{\Sigma'}$ n'est pas donnée de l'extérieur, mais est codée par le préfixe $\text{bin}_{\Sigma'}(0);\text{bin}_{\Sigma'}(1);\text{bin}_{\Sigma'}(;);\text{bin}_{\Sigma'}(\textbf{accept});\text{bin}_{\Sigma'}(\textbf{reject});\ldots;;$

- 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 $\langle M\rangle;x$

Réduire le nombre de lettres

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



- 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]

Réduire à deux lettres (plus ;)

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

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

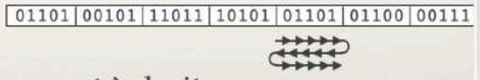
Point subtil: la fonction $\text{bin}_{\Sigma'}$ n'est pas donnée de l'extérieur, mais est codée par le préfixe $\text{bin}_{\Sigma'}(0);\text{bin}_{\Sigma'}(1);\text{bin}_{\Sigma'}(;);\text{bin}_{\Sigma'}(\textbf{accept});\text{bin}_{\Sigma'}(\textbf{reject});\ldots;;$

- 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 $k \triangleq \Theta(\log |\Sigma|)$ 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 $O(\log |\Sigma'|)$)

Codage binaire (avec séparateur ;)

- Pour chaque Σ', on choisit un code binaire telle que toutes les lettres a aient un code $\mathrm{bin}_{\Sigma'}(a)$ de la **même longueur** (typiquement $O(\log |\Sigma'|)$)
- Pour un mot $w=a_1\, a_2 \ldots a_n \in \Sigma'^*$, (et notamment $\langle M\rangle;x$)

$$\mathrm{bin}_{\Sigma'}(w) \stackrel{\text{def}}{=} \mathrm{bin}_{\Sigma'}(a_1);\mathrm{bin}_{\Sigma'}(a_2);\ldots;\mathrm{bin}_{\Sigma'}(a_n);$$

Codage binaire (avec séparateur ;)

- Pour chaque Σ', on choisit un code binaire telle que toutes les lettres a aient un code $\mathrm{bin}_{\Sigma'}(a)$ de la **même longueur** (typiquement $O(\log |\Sigma'|)$)
- Pour un mot $w=a_1\, a_2 \ldots a_n \in \Sigma'^*$, (et notamment $\langle M\rangle;x$)

  $\mathrm{bin}_{\Sigma'}(w) \stackrel{\text{def}}{=} \mathrm{bin}_{\Sigma'}(a_1);\mathrm{bin}_{\Sigma'}(a_2);\ldots;\mathrm{bin}_{\Sigma'}(a_n);$
- Dans la suite, j'écrirai $\boldsymbol{w}$ (en gras) pour $\mathrm{bin}_{\Sigma'}(w)$

  [sinon, on aura des expressions immenses]

Codage binaire (avec séparateur ;)

- Pour chaque Σ', on choisit un code binaire telle que toutes les lettres a aient un code $\mathrm{bin}_{\Sigma'}(a)$ de la **même longueur** (typiquement $O(\log |\Sigma'|)$)
- Pour un mot $w=a_1\, a_2 \ldots a_n \in \Sigma'^*$, (et notamment $\langle M\rangle;x$)
  $\mathrm{bin}_{\Sigma'}(w) \stackrel{\text{def}}{=} \mathrm{bin}_{\Sigma'}(a_1);\mathrm{bin}_{\Sigma'}(a_2);\ldots;\mathrm{bin}_{\Sigma'}(a_n);$
- Dans la suite, j'écrirai $\boldsymbol{w}$ (en gras) pour $\mathrm{bin}_{\Sigma'}(w)$
  [sinon, on aura des expressions immenses]
- Par exemple, si $\mathrm{bin}_{\Sigma'}(a)=010$, $\mathrm{bin}_{\Sigma'}(b)=001$, $\mathrm{bin}_{\Sigma'}(c)=110$,
  $\boldsymbol{abbcbc}$ = 010;001;001;110;001;110;

Codage binaire (avec séparateur ;)

- Pour chaque Σ', on choisit un code binaire telle que toutes les lettres a aient un code $\mathrm{bin}_{\Sigma'}(a)$ de la **même longueur** (typiquement $O(\log |\Sigma'|)$)
- Pour un mot $w=a_1\, a_2 \ldots a_n \in \Sigma'^*$, (et notamment $\langle M\rangle;x$)

  $\mathrm{bin}_{\Sigma'}(w) \stackrel{\text{def}}{=} \mathrm{bin}_{\Sigma'}(a_1);\mathrm{bin}_{\Sigma'}(a_2);\ldots;\mathrm{bin}_{\Sigma'}(a_n);$
- Dans la suite, j'écrirai $\boldsymbol{w}$ (en gras) pour $\mathrm{bin}_{\Sigma'}(w)$

  [sinon, on aura des expressions immenses]
- Par exemple, si $\mathrm{bin}_{\Sigma'}(a)=010$, $\mathrm{bin}_{\Sigma'}(b)=001$, $\mathrm{bin}_{\Sigma'}(c)=110$,

  $\boldsymbol{abbcbc} = 010;001;001;110;001;110;$
- Avec cette convention, on a donc:

  $(\langle M\rangle;x)^{\mathrm{bin}} \stackrel{\text{def}}{=} \mathrm{bin}_{\Sigma'}(0);\mathrm{bin}_{\Sigma'}(1);\mathrm{bin}_{\Sigma'}(;);\mathrm{bin}_{\Sigma'}(a);\mathrm{bin}_{\Sigma'}(r);\ldots;;\mathrm{bin}_{\Sigma'}(\langle M\rangle;x)$

  $= \mathbf{01;ar\ldots;}\boldsymbol{\langle M\rangle;x}$

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

Quote: $\langle M \rangle_{\text{bin}}$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\texttt{B}}, \underline{\texttt{\$}})$
- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  et on pose $(q'_i,b_i,d_i) \stackrel{\text{def}}{=} \delta(q_i,a_i)$, $1 \leq i \leq m$

Quote: $\langle M \rangle_{\text{bin}}$

- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\mathtt{B}}, \underline{\$})$
- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$ 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

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

  sur l'alphabet ~~$Q^+ \uplus \sum \uplus \{\leftarrow, \downarrow, \rightarrow, \#\}$~~ $\{0,1,;\} \subseteq \sum$

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

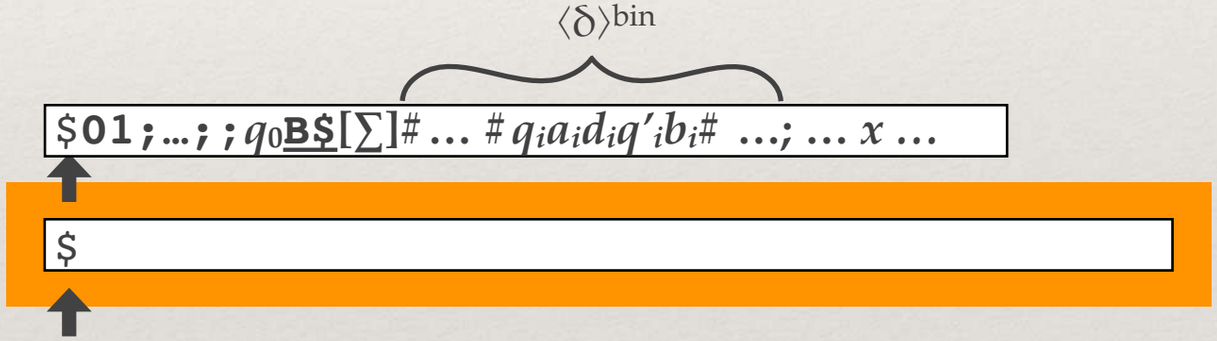
- Soit $M \stackrel{\text{def}}{=} (Q, q_0, \sum, \delta, \underline{\texttt{B}}, \underline{\texttt{\$}})$
- On énumère $Q \times \sum \stackrel{\text{def}}{=} \{(q_1,a_1),\ldots,(q_m,a_m)\}$
  et on pose $(q'_i,b_i,d_i) \stackrel{\text{def}}{=} \delta(q_i,a_i)$, $1 \leq i \leq m$
- On code δ par le mot
  $\langle \delta \rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
  sur l'alphabet ~~$Q \uplus \sum \uplus \{\leftarrow,\downarrow,\rightarrow,\#\}$~~ $\{0,1,;\} \subseteq \sum$
- $(\langle M \rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;q_0<u>B$</u>[autres lettres de $\sum$]#** $\langle \delta \rangle^{\text{bin}}$**;$x$**

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)

- $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
- $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]# $\langle\delta\rangle$;x
- alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$



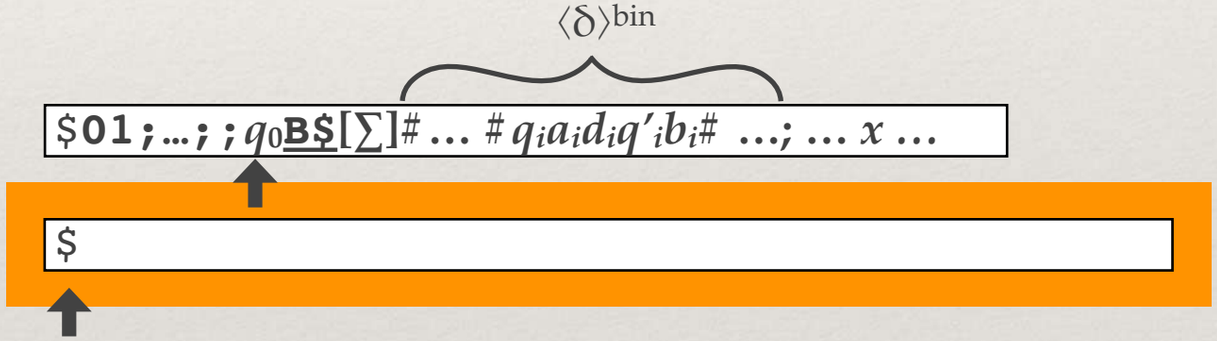
U commence par:
— lire ~~et mémoriser~~ q_0, B, $ ~~dans son état interne~~

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)

  - $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
  - $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]#$\,\langle\delta\rangle$**;**$x$
  - alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$



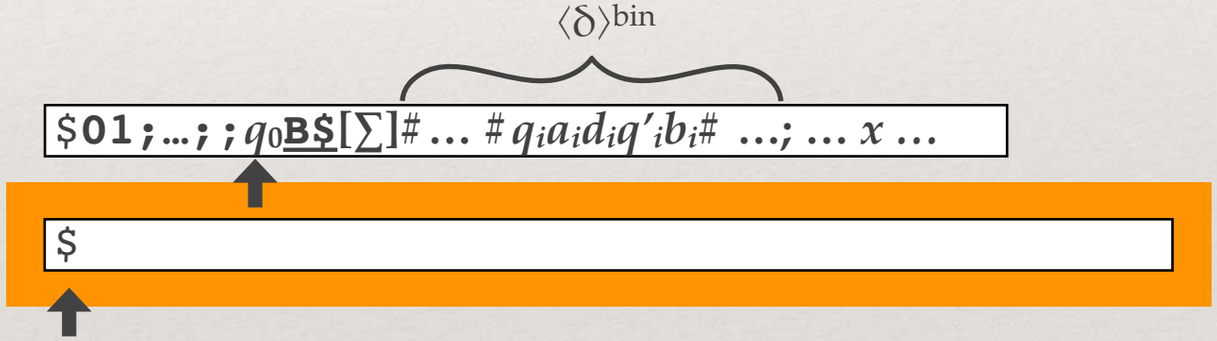
U commence par:
— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

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)

- $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
- $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]#$\langle\delta\rangle$**;**$x$
- alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$



U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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)

  - $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
  - $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]#$\langle\delta\rangle$**;**$x$
  - alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$

U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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)

  - $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
  - $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]#$\langle\delta\rangle$**;**$x$
  - alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$

U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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)

  - $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
  - $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]# $\langle\delta\rangle$**;**x
  - alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$

U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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)

- $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
- $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**$[\sum]\#\,\langle\delta\rangle;x$
- alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$

U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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 $\text{bin}_{\Sigma'}(;)$)

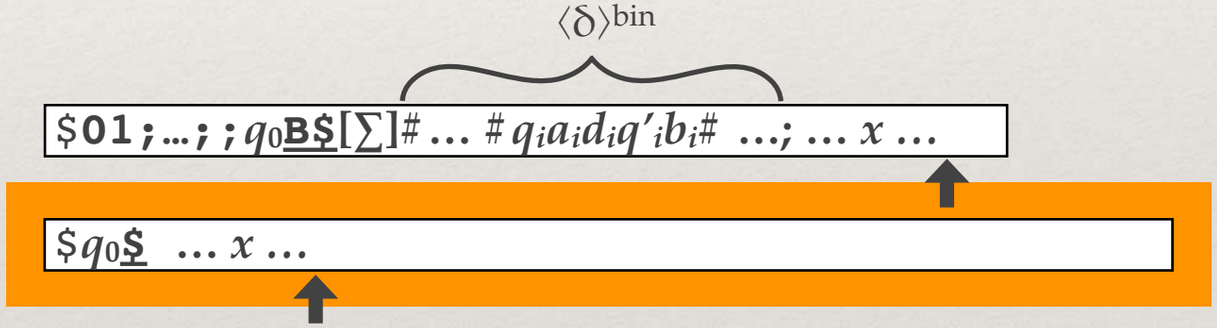
— recopier x sur la bande de travail

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)

- $\langle\delta\rangle^{\text{bin}} \triangleq q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
- $(\langle M\rangle;x)^{\text{bin}} \triangleq$ **01;ar…;;**q_0**B$**[∑]#$\,\langle\delta\rangle$**;**$x$
- alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$



U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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 $\text{bin}_{\Sigma'}(;)$)

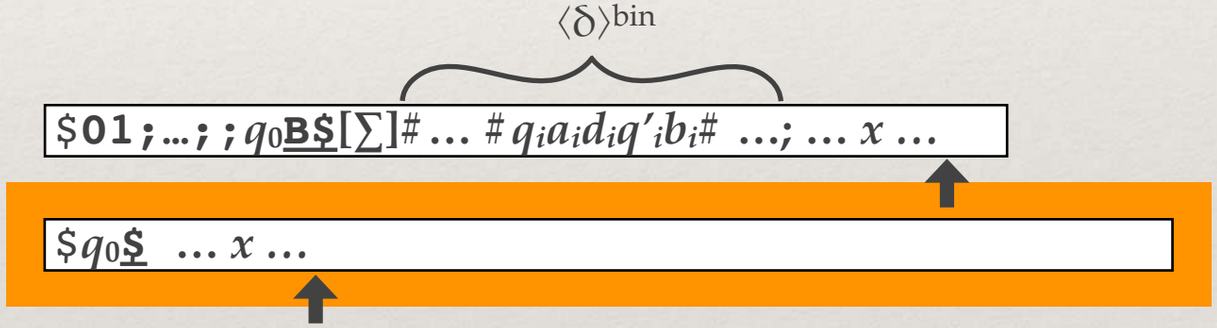
— recopier x sur la bande de travail

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)

  - $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
  - $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]# $\langle\delta\rangle$**;**x
  - alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$



U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** 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

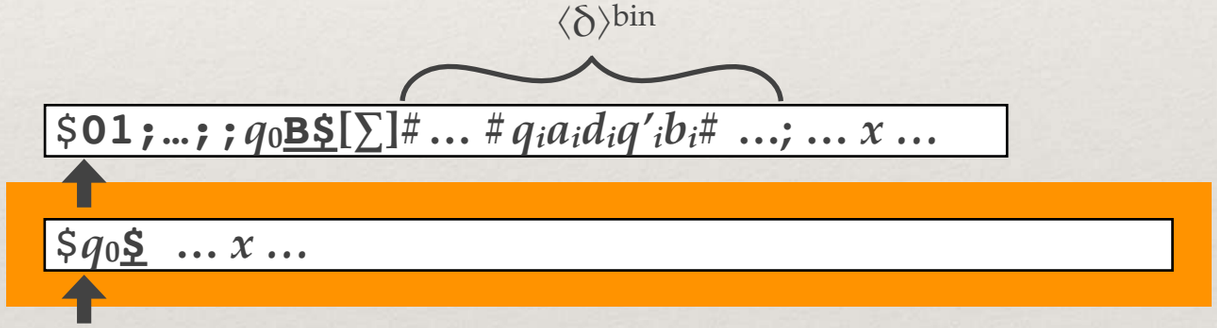
— nettoyer: ramener les têtes en début de bande

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)

- $\langle\delta\rangle^{\text{bin}} \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#\, q_2a_2d_2q'_2b_2\#\, \ldots\#\, q_ma_md_mq'_mb_m$
- $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;**q_0**B$**[∑]# $\langle\delta\rangle$**;**x
- alphabet {0,1,;}

- En entrée: $(\langle M\rangle;x)^{\text{bin}}$



U commence par:

— lire ~~et mémoriser~~ q_0, **B**, **$** ~~dans son état interne~~ (après le **;;**)

— écrire q_0**$** sur la bande de travail
(= tout jusqu'au premier ';', puis tout entre le 2ème et le 3ème)

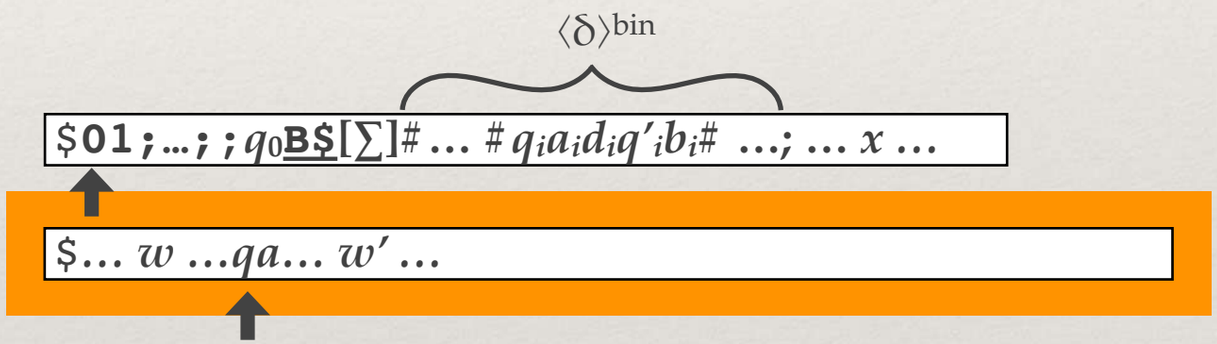
— avancer jusqu'au **;** (en gras, = la suite de lettres $\text{bin}_{\Sigma'}(;)$)

— 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 $]
  alors la bande de travail de *U* contient **$*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]

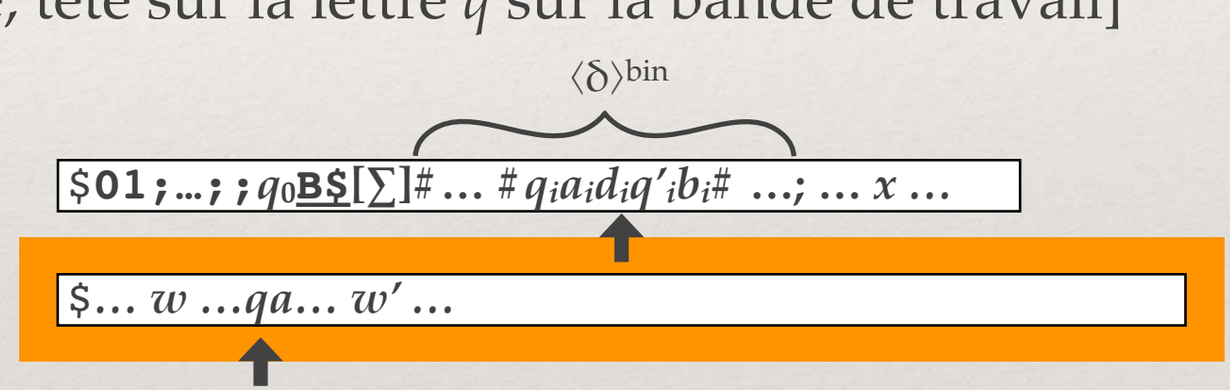


Pour simuler une étape de *M*, *U:*

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

Apply: la machine *U*

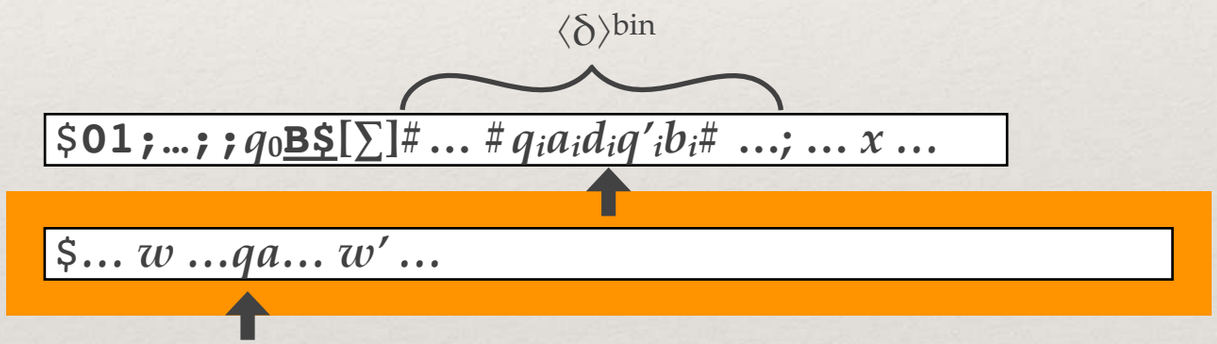
- A toute étape de la simulation de *M* (en entrée) par *U*,
  si la configuration de *M* est (*w*,*q*,*w'*) [*ww'* commençant par $]
  alors la bande de travail de *U* contient **$*wqw'*** [en gras=codé sur 0,1,;]
  [avec: tête à droite du $ sur bande d'entrée; tête sur la lettre *q* sur la bande de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



Pour simuler une étape de *M, U:*

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]

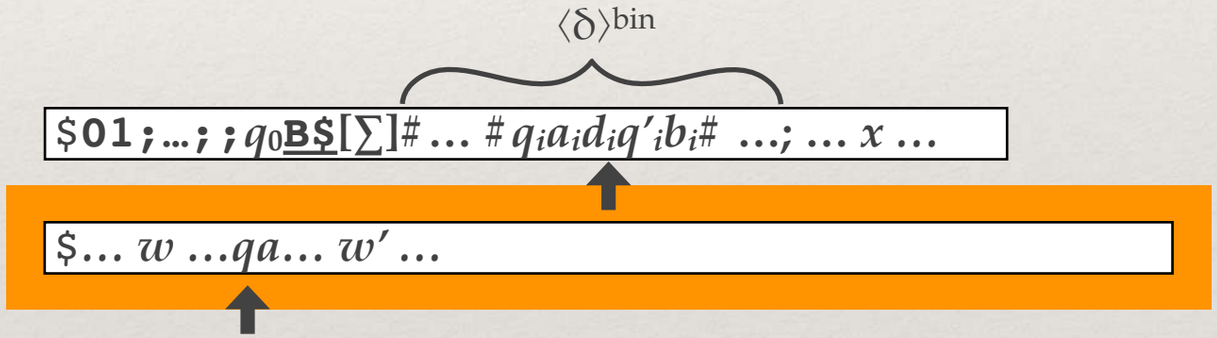


Pour simuler une étape de M, U:

— parcourt son entrée jusqu'à trouver **$\# q_i a_i q'_i b_i d_i$** avec **$q_i a_i = qa$**
et mémorise d_i , ~~q'_i, b_i,~~ dans son état interne (d_i sur 2 bits+';')

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 $\mathtt{\$}\boldsymbol{wqw'}$ [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]



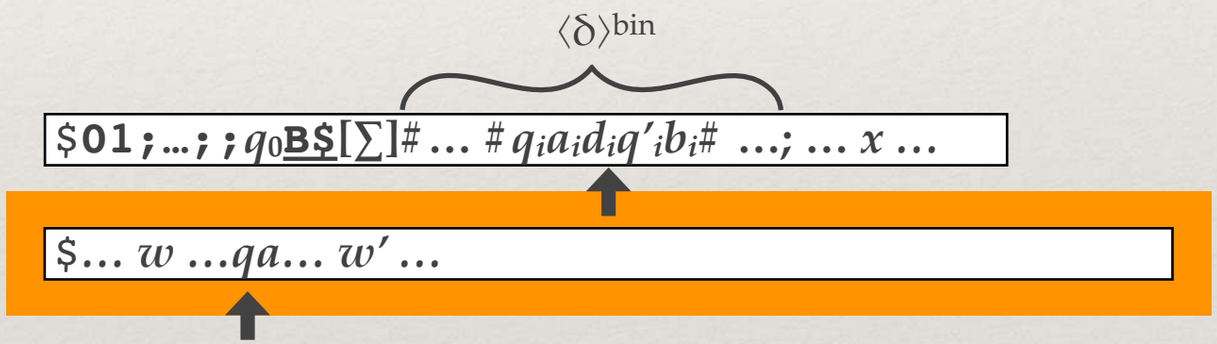
Pour simuler une étape de M, U:

— parcourt son entrée jusqu'à trouver $\#\, q_ia_iq'_ib_id_i$ avec $q_ia_i=qa$
et mémorise d_i , ~~q'_i, b_i,~~ dans son état interne (d_i sur 2 bits+';')

il suffit de comparer en parallèle
$\boldsymbol{qa}=\mathrm{bin}_{\Sigma'}(q);\mathrm{bin}_{\Sigma'}(a)$; avec $\boldsymbol{q_ia_i}=\mathrm{bin}_{\Sigma'}(q_i);\mathrm{bin}_{\Sigma'}(a_i)$;
— si on atteint le 2ème ';' en même temps, on a trouvé
— sinon, avancer la tête d'entrée jusqu'au prochain #
$(=\mathrm{bin}_{\Sigma'}(\#);)$ et ramener la tête de travail 2 ';' en arrière

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 $\$\boldsymbol{wqw'}$ [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:

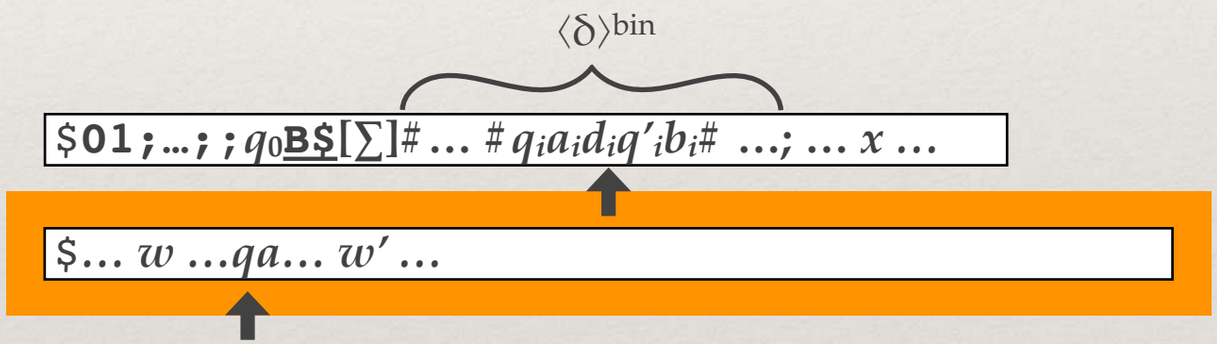
— parcourt son entrée jusqu'à trouver $\#\, \boldsymbol{q_i a_i q'_i b_i d_i}$ avec $\boldsymbol{q_i a_i = qa}$
et mémorise d_i , ~~q'_i, b_i,~~ dans son état interne (d_i sur 2 bits+';')

il suffit de comparer en parallèle
$\boldsymbol{qa}=\text{bin}_{\Sigma'}(q);\text{bin}_{\Sigma'}(a)$; avec $\boldsymbol{q_i a_i}=\text{bin}_{\Sigma'}(q_i);\text{bin}_{\Sigma'}(a_i)$;
— si on atteint le 2ème ';' en même temps, on a trouvé
— sinon, avancer la tête d'entrée jusqu'au prochain #
(=$\text{bin}_{\Sigma'}(\#)$;) et ramener la tête de travail 2 ';' en arrière

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

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]



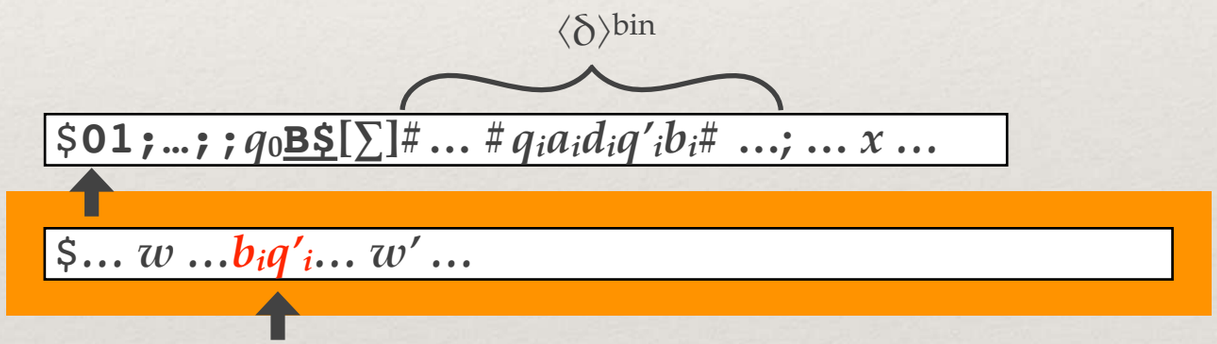
Pour simuler une étape de M, U:

— parcourt son entrée jusqu'à trouver **$\# q_i a_i q'_i b_i d_i$** avec **$q_i a_i = qa$**
  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'…
  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]



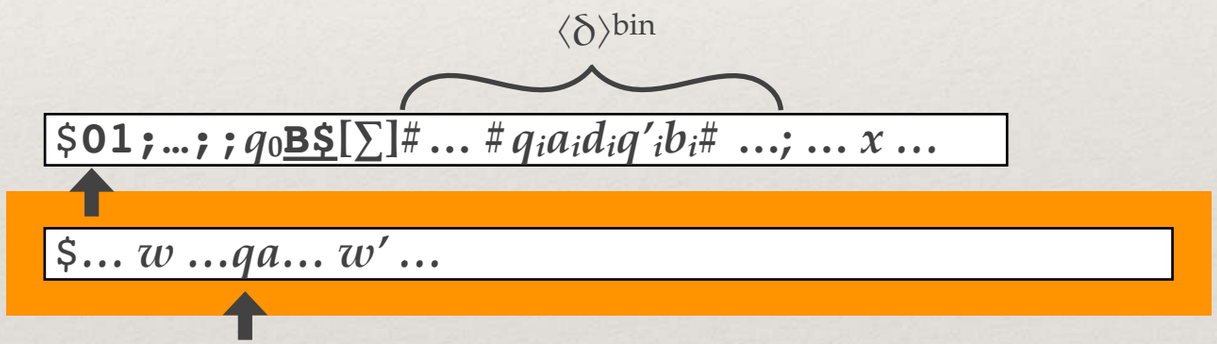
Pour simuler une étape de M, U:

— parcourt son entrée jusqu'à trouver **$\# q_i a_i q'_i b_i d_i$** avec **$q_i a_i = qa$**
  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'…
  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]



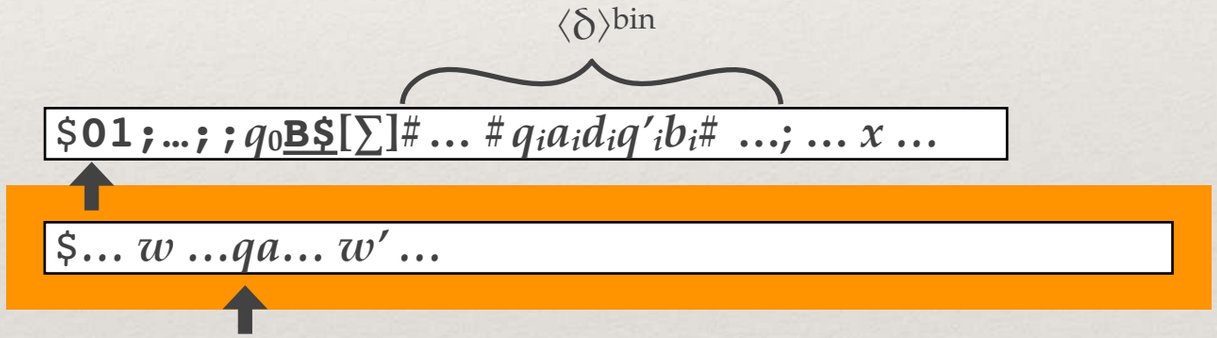
Pour simuler une étape de M, U:

— parcourt son entrée jusqu'à trouver $\#\, q_i a_i q'_i b_i d_i$ avec $q_i a_i = qa$
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

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 $\$\boldsymbol{wqw'}$ [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 $\#\, \boldsymbol{q_i a_i q'_i b_i d_i}$ avec $\boldsymbol{q_i a_i = qa}$
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 $\boldsymbol{q}$=bin$_{\Sigma'}$(**accept**);, rejette si $\boldsymbol{q}$=bin$_{\Sigma'}$(**reject**);

La machine universelle U

- Rappel: $(\langle M\rangle;x)^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;q_0B$[autres lettres de Σ]#** $\langle\delta\rangle^{\text{bin}}$**;$x$**
- Nous avons construit une machine **universelle** U, i.e.:
- **Théorème.** Pour tout mT M (sur un alphabet Σ **quelconque**), pour toute entrée $x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
  — 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,;,B,$}

kwote

- Rappel: $(\langle M\rangle;x)^{\text{bin}} \triangleq \langle M\rangle^{\text{bin}};x = \langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x)$
  où $\langle M\rangle^{\text{bin}} \triangleq$ **01;ar…;;q_0B$[autres lettres de Σ]#** $\langle\delta\rangle^{\text{bin}}$
  est un mot sur {0,1,;}

kwote

- Rappel: $(\langle M\rangle;x)^{\text{bin}} \triangleq \langle M\rangle^{\text{bin}};x = \langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x)$
  où $\langle M\rangle^{\text{bin}} \triangleq$ **01;ar…;;q_0B$[autres lettres de Σ]#** $\langle\delta\rangle^{\text{bin}}$
  est un mot sur {0,1,;}
- On définit `kwote` de sorte que $\texttt{kwote}(;\langle M\rangle^{\text{bin}}) = \text{bin}_{\Sigma'}(;\langle M\rangle^{\text{bin}})$

kwote

- Rappel: $(\langle M\rangle;x)^{\text{bin}} \triangleq \langle M\rangle^{\text{bin}};x = \langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x)$
  où $\langle M\rangle^{\text{bin}} \triangleq$ **01;ar…;;q_0B$[autres lettres de Σ]#** $\langle\delta\rangle^{\text{bin}}$
  est un mot sur {0,1,;}

- On définit `kwote` de sorte que $\texttt{kwote}(;\langle M\rangle^{\text{bin}}) = \text{bin}_{\Sigma'}(;\langle M\rangle^{\text{bin}})$

- 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 $\langle M\rangle^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;q_0B$[autres lettres de ∑]#** $\langle\delta\rangle^{\text{bin}}$
  est un mot sur {0,1,;}

- On définit `kwote` de sorte que $\texttt{kwote}(;\langle M\rangle^{\text{bin}}) = \text{bin}_{\Sigma'}(;\langle M\rangle^{\text{bin}})$

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 $\langle M\rangle^{\text{bin}} \stackrel{\text{def}}{=}$ **01;ar…;;q_0B$[autres lettres de ∑]#** $\langle\delta\rangle^{\text{bin}}$
  est un mot sur {0,1,;}

- On définit `kwote` de sorte que `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 $;w_0;w_1;w_;;\ldots;;w'$ où w_0, w_1, $w_;$, …, 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

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

- **Proposition.** L_U est r.e.

Le langage L_U

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

Le langage L_U

- $L_U \stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x) \mid M \text{ mT sur un alphabet } \Sigma,\ x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*,$ tels que M 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 $x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
— 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 $w_0;w_1;w_;;\ldots;;w';x$ avec w_0, w_1, $w_;$, …, w', x en binaire,

Le langage L_U

- $L_U \stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x) \mid M \text{ mT sur un alphabet } \Sigma,\ x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*,$ tels que M 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 $x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
— 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 $w_0;w_1;w_;;\ldots;;w';x$ avec w_0, w_1, $w_;$, …, w', x en binaire,

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

Le langage L_U

- $L_U \stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x) \mid M \text{ mT sur un alphabet } \Sigma,\ x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*,$ tels que M 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 $x \in (\Sigma - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
— 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 $w_0;w_1;w_;;\ldots;;w';x$ avec w_0, w_1, $w_;$, …, w', x en binaire,

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

  - $\langle\delta\rangle \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#q_2a_2d_2q'_2b_2\#\ldots\#q_ma_md_mq'_mb_m$
  - $\langle M\rangle \stackrel{\text{def}}{=} q_0\underline{\texttt{B\$}}[\Sigma]\#\langle\delta\rangle$

Le langage L_U

- $L_U \stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x) \mid M \text{ mT sur un alphabet } \Sigma,\ x \in (\Sigma - \{\underline{\texttt{B}},\underline{\texttt{\$}}\})^*,$ tels que M 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 $x \in (\Sigma - \{\underline{\texttt{B}},\underline{\texttt{\$}}\})^*$,
— 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 $w_0;w_1;w_;;\ldots;;w';x$ avec w_0, w_1, $w_;$, …, w', x en binaire,

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

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#q_2a_2d_2q'_2b_2\#\ldots\#q_ma_md_mq'_mb_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0\underline{\texttt{B}}\underline{\texttt{\$}}[\Sigma]\#\langle\delta\rangle$

Le langage L_U

- $L_U \stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}}\ \text{bin}_{\Sigma'}(;x) \mid M \text{ mT sur un alphabet } \Sigma,\ x \in (\Sigma - \{\underline{\texttt{B}},\underline{\texttt{\$}}\})^*,$ tels que M 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 $x \in (\Sigma - \{\underline{\texttt{B}},\underline{\texttt{\$}}\})^*$,
— 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 $w_0;w_1;w_;;\ldots;;w';x$ avec w_0, w_1, $w_;$, …, w', x en binaire,

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

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

- $\langle\delta\rangle \stackrel{\text{def}}{=} q_1a_1d_1q'_1b_1\#q_2a_2d_2q'_2b_2\#\ldots\#q_ma_md_mq'_mb_m$
- $\langle M\rangle \stackrel{\text{def}}{=} q_0\underline{\texttt{B}}\underline{\texttt{\$}}[\Sigma]\#\langle\delta\rangle$

Cette vérification de format est donc décidable

Le problème de l'acceptation

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

- **Théorème.** L_U est indécidable.

Le problème de l'acceptation

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

Le problème de l'acceptation

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

Le problème de l'acceptation

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

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

$(\texttt{kwote}(;\langle M\rangle^{\text{bin}}) = \text{bin}_{\Sigma'}(;\langle M\rangle^{\text{bin}})$

Le problème de l'acceptation

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

- $L(D) = \{\langle M\rangle^{\text{bin}} \mid M \text{ n'accepte } \textbf{pas } \langle M\rangle^{\text{bin}}\}$

Le problème de l'acceptation

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

- $L(D) = \{\langle M\rangle^{\text{bin}} \mid M \text{ n'accepte } \textbf{pas } \langle M\rangle^{\text{bin}}\}$

- $\langle D\rangle^{\text{bin}} \in L(D)$ 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. □

Le problème de l'acceptation

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

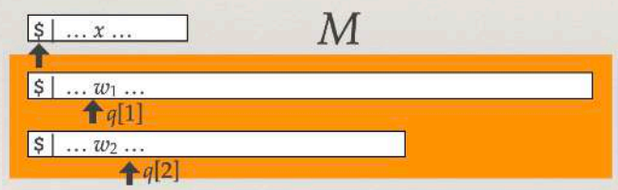
- **Théorème.** L_U n'est même pas co-r.e.

Le problème de l'acceptation

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



$\delta(q_0,(a, a_1, a_2)) = (q_0,(a,a),(\rightarrow,\rightarrow,\rightarrow))$ si $a \neq \texttt{B}$
$\delta(q_0,(\texttt{B}, a_1,a_2)) = (q_1,(a_1,a_2),(\leftarrow,\leftarrow,\leftarrow))$
$\delta(q_1,(a, a_1,a_2)) = (q_1,(a_1,a_2),(\leftarrow,\leftarrow,\leftarrow))$ 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 $q'[1] \neq \textbf{accept}[1]$, $q'[2] \neq \textbf{accept}[2]$
$\delta((q[1],q[2]),(a,a_1,a_2)) = (\textbf{accept},(a_1,a_2),(\downarrow,\downarrow,\downarrow))$
si $q'[1]=\textbf{accept}[1]$
$\delta((q[1],q[2]),(a,a_1,a_2)) = (\textbf{reject},(a_1,a_2),(\downarrow,\downarrow,\downarrow))$
si $q'[2]=\textbf{accept}[2]$ (et $q'[1] \neq \textbf{accept}[1]$)

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=} \{\langle M \rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma,$
  telle que M **termine** en partant du mot vide $\varepsilon\}$

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

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=} \{\langle M \rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma,$
  telle que M **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 $M_{\textbf{Halt}}$ qui, sur son entrée:

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=} \{\langle M \rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma,$
  telle que M **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 $M_{\textbf{Halt}}$ qui, sur son entrée:

  - vérifie qu'elle est du format $\langle M \rangle^{\text{bin}}$

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma,$
  telle que M **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 $M_{\textbf{Halt}}$ qui, sur son entrée:

  - vérifie qu'elle est du format $\langle M\rangle^{\text{bin}}$

  - simule M sur l'entrée ε

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=} \{\langle M \rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma,$
  telle que M **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 $M_{\textbf{Halt}}$ 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**
    ($M_{\textbf{Halt}}$ ne va jamais en **reject**)

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}} \mid M \text{ mT sur un alphabet } \Sigma,$
  telle que M **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 $M_{\textbf{Halt}}$ 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**
    ($M_{\textbf{Halt}}$ ne va jamais en **reject**)
- Alors $L(M_{\textbf{Halt}}) =$ **Halt**.

Le problème de l'arrêt

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

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

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=}$ $\{\langle M\rangle^{\text{bin}} \mid M$ mT sur un alphabet $\sum$, $x \in (\sum - \{\underline{\text{B}}, \underline{\$}\})^*$,
  tels que M **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

Le problème de l'arrêt

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

Le problème de l'arrêt

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

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

Le problème de l'arrêt

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

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

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

Le problème de l'arrêt

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

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

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

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

Le problème de l'arrêt

- **Halt** $\stackrel{\text{def}}{=}$ $\{\langle M\rangle^{\text{bin}} \mid M$ mT sur un alphabet $\sum$, $x \in (\sum - \{\underline{\texttt{B}}, \underline{\texttt{\$}}\})^*$,
  tels que M **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 $L_U \stackrel{\text{def}}{=} \{\langle M\rangle^{\text{bin}}\ \text{bin}_{\sum'}(;x) \mid M \text{ accepte } x\}$ comme suit:
  - on vérifie le format d'entrée $\langle M\rangle^{\text{bin}}\ \text{bin}_{\sum'}(;x)$
  - on construit (le code de) une nouvelle mT M_x qui commence par écrire x (mémorisé dans son état interne) sur sa bande, puis simule M; si M rejette, alors M_x **boucle** (`~forever-loop`)

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

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