Vers l'indécidabilité :
Ensembles (in)dénombrables, machine universelle, et problème de l'arrêt

Calculabilité et Complexité

Paul Brunet

Vers l'indécidabilité
Plan

1. Ensembles dénombrables
2. Les classes **R** et **RE**
3. Machine universelle
4. Le problème de l'arrêt

Vers l'indécidabilité
Plan

1. Ensembles dénombrables
2. Les classes **R** et **RE**
3. Machine universelle
4. Le problème de l'arrêt

Ensembles dénombrables
Définition

définition

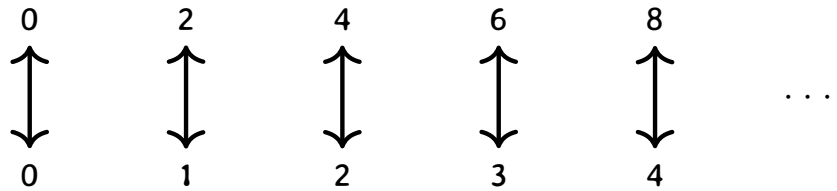
Un ensemble infini est **dénombrable** si il existe une bijection entre cet ensemble et l'ensemble des nombres naturels $\mathbb{N}$.

remarque

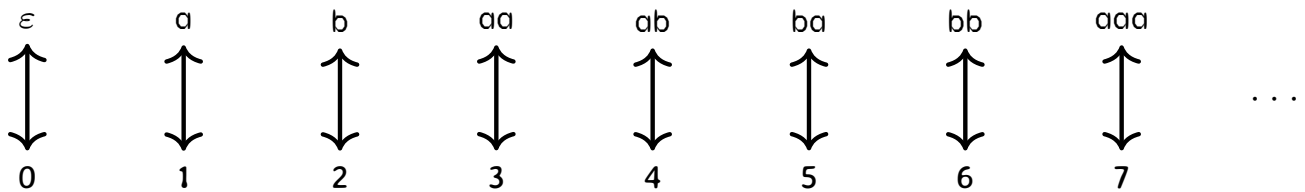
Les ensembles finis sont dénombrables au sens usuel, mais il est souvent confortable en mathématique de restreindre le sens du mot « dénombrable » pour signifier exactement « infini dénombrable ».

Ensembles dénombrables
Exemples

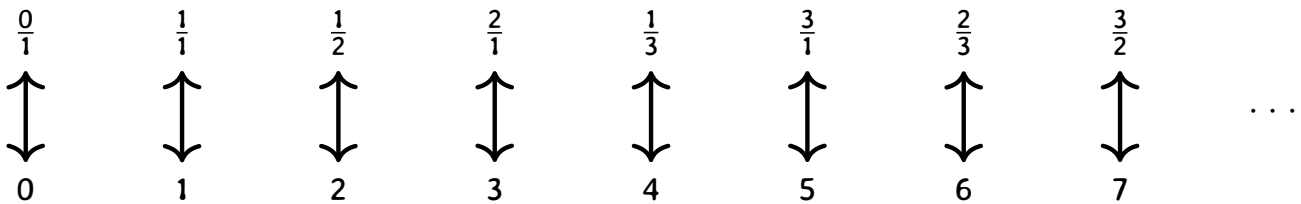
☞ L'ensemble des entiers pairs :



☞ L'ensemble des mots sur $\{a, b\}$:



☞ L'ensemble des nombres rationnels :



☞ L'ensemble des expressions régulières est dénombrable.

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

☞ Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

- Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.
- Pour chaque élément $x \in E$, $f(x)$ est un sous-ensemble de E. On peut donc se demander si $x \in f(x)$?

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

- Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.
- Pour chaque élément $x \in E$, $f(x)$ est un sous-ensemble de E. On peut donc se demander si $x \in f(x)$?
- On définit $D := \{x \in E \mid x \notin f(x)\}$.

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

- Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.
- Pour chaque élément $x \in E$, $f(x)$ est un sous-ensemble de E. On peut donc se demander si $x \in f(x)$?
- On définit $D := \{x \in E \mid x \notin f(x)\}$.
- Comme f est bijective et $D \in \mathcal{P}(E)$, il existe x_D tel que $f(x_D) = D$.

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

- Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.
- Pour chaque élément $x \in E$, $f(x)$ est un sous-ensemble de E. On peut donc se demander si $x \in f(x)$?
- On définit $D := \{x \in E \mid x \notin f(x)\}$.
- Comme f est bijective et $D \in \mathcal{P}(E)$, il existe x_D tel que $f(x_D) = D$.
- On arrive à une contradiction en se posant la question $x_D \in D$?:

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

- Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.
- Pour chaque élément $x \in E$, $f(x)$ est un sous-ensemble de E. On peut donc se demander si $x \in f(x)$?
- On définit $D := \{x \in E \mid x \notin f(x)\}$.
- Comme f est bijective et $D \in \mathcal{P}(E)$, il existe x_D tel que $f(x_D) = D$.
- On arrive à une contradiction en se posant la question $x_D \in D$?:
  - si $x_D \in D$, alors comme $D = f(x_D)$, par définition $x_D \notin D$;

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties. Autrement dit, pour tout ensemble E, il n'existe pas de bijection entre E et $\mathcal{P}(E)$.

Preuve.

- Par contradiction, on suppose qu'il existe une fonction bijective $f : E \to \mathcal{P}(E)$.
- Pour chaque élément $x \in E$, $f(x)$ est un sous-ensemble de E. On peut donc se demander si $x \in f(x)$?
- On définit $D := \{x \in E \mid x \notin f(x)\}$.
- Comme f est bijective et $D \in \mathcal{P}(E)$, il existe x_D tel que $f(x_D) = D$.
- On arrive à une contradiction en se posant la question $x_D \in D$?:
  - si $x_D \in D$, alors comme $D = f(x_D)$, par définition $x_D \notin D$;
  - si $x_D \notin D$, alors comme $D = f(x_D)$, par définition $x_D \in D$!

□

Théorème de Cantor

théorème

Aucun ensemble n'est en bijection avec l'ensemble de ses parties.

corollaire

L'ensemble des parties d'un ensemble dénombrable est indénombrable.

- L'ensemble des mots sur $\{a, b\}$ est dénombrable.
- Donc l'ensemble des langages sur $\{a, b\}$ est indénombrable.
- L'ensemble des expressions régulières est dénombrable, donc l'ensemble des langages réguliers est dénombrable.
- Il y a donc beaucoup plus de langages non-réguliers que de langages réguliers !

Vers l'indécidabilité
Plan

1. Ensembles dénombrables

2. Les classes **R** et **RE**

3. Machine universelle

4. Le problème de l'arrêt

La classe R

On formalise les notions vues dans les cours précédents :

définition

La classe R est l'ensemble des langages acceptés par une machine de Turing à décider.

C'est l'ensemble des langages

- ☞ décidés par une machine de Turing ;
- ☞ récursifs, décidables, calculables ;
- ☞ solubles algorithmiquement.

La classe RE

définition

La classe **RE** est l'ensemble des langages reconnus par une machine de Turing.

Ce sont les langages (problèmes) qui sont :

- reconnus/acceptés par une machine de Turing ;
- semi-récursifs, semi-décidables, semi-calculables ;
- partiellement solubles algorithmiquement ;
- récursivement énumérables.

lemme

La classe **R** est contenue dans la classe **RE**.

Un premier langage impossible

- L'ensemble $\mathbb{M}$ des machines de Turing sur l'alphabet $\{0,1\}$ est dénombrable.
- On a donc $f : \mathbb{M} \to \mathbb{N}$ une fonction bijective, que l'on peut utiliser pour coder les machines de Turing sur l'alphabet $\{0,1\}$:

$$[\mathcal{M}]_{code} := [f(\mathcal{M})]_2$$

- Soit $\mathcal{M} \in \mathbb{M}$, on peut donc demander si $[\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{M})$?
- On définit le langage suivant :

$$\begin{aligned} L_0 &:= \left\{ [\mathcal{M}]_{code} \;\middle|\; [\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{M}) \right\} \\ &= \left\{ w \in \{0,1\}^\star \;\middle|\; \exists \mathcal{M} \in \mathbb{M} : w = [\mathcal{M}]_{code} \wedge w \notin \mathcal{L}(\mathcal{M}) \right\} \end{aligned}$$

- Si ce langage appartenait à la classe **RE**, il existerait une machine $\mathcal{M}_0$ telle que $\mathcal{L}(\mathcal{M}_0) = L_0$.
- Or c'est impossible, car :

$$\begin{aligned} [\mathcal{M}_0]_{code} \in \mathcal{L}(\mathcal{M}_0) &\Rightarrow [\mathcal{M}_0]_{code} \notin \mathcal{L}(\mathcal{M}_0); \\ [\mathcal{M}_0]_{code} \notin \mathcal{L}(\mathcal{M}_0) &\Rightarrow [\mathcal{M}_0]_{code} \in \mathcal{L}(\mathcal{M}_0). \end{aligned}$$

Critique du résultat précédent

Ouais, mais monsieur, il est pas un peu pérave votre langage impossible ?

Vers l'indécidabilité
Plan

1. Ensembles dénombrables
2. Les classes **R** et **RE**

3. Machine universelle
4. Le problème de l'arrêt

Une machine programmable

Un des attraits des ordinateurs est que ce sont des machines **programmables**.

Une machine programmable

- Un des attraits des ordinateurs est que ce sont des machines **programmables**.
- Ainsi, une même machine peut accomplir une grande variété de tâches, suivant le programme qu'elle exécute.

Une machine programmable

- Un des attraits des ordinateurs est que ce sont des machines **programmables**.
- Ainsi, une même machine peut accomplir une grande variété de tâches, suivant le programme qu'elle exécute.
- Les machines de Turing ont également cette possibilité.

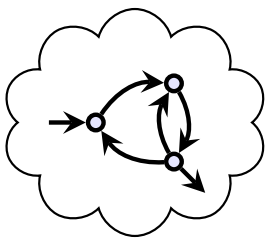
Une machine programmable

- Un des attraits des ordinateurs est que ce sont des machines **programmables**.
- Ainsi, une même machine peut accomplir une grande variété de tâches, suivant le programme qu'elle exécute.
- Les machines de Turing ont également cette possibilité.
- En effet, on va maintenant montrer qu'il existe une machine **déterministe** $\mathcal{U}$ qui prend en entrée une machine (non-déterministe) $\mathcal{M}$ et un mot $w \in \Sigma^\star$, et qui simule l'exécution de $\mathcal{M}$ sur l'entrée w.

Une machine programmable

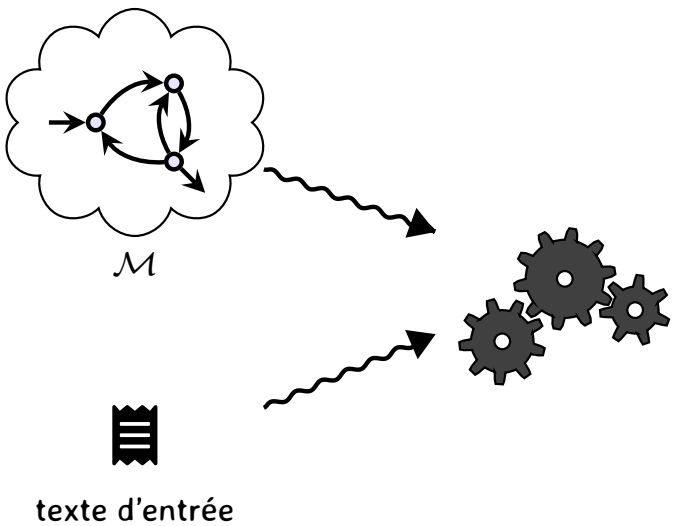
- Un des attraits des ordinateurs est que ce sont des machines **programmables**.
- Ainsi, une même machine peut accomplir une grande variété de tâches, suivant le programme qu'elle exécute.
- Les machines de Turing ont également cette possibilité.
- En effet, on va maintenant montrer qu'il existe une machine **déterministe** $\mathcal{U}$ qui prend en entrée une machine (non-déterministe) $\mathcal{M}$ et un mot $w \in \Sigma^\star$, et qui simule l'exécution de $\mathcal{M}$ sur l'entrée w.
- Conceptuellement, $\mathcal{U}$ est un genre d'interpréteur, comme par exemple la JVM.

La machine universelle
Utilisation habituelle d'une machine de Turing

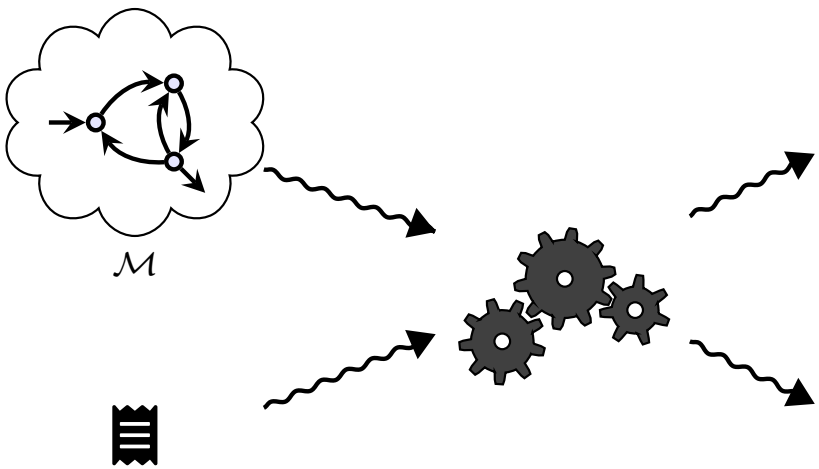


texte d'entrée

La machine universelle
Utilisation habituelle d'une machine de Turing



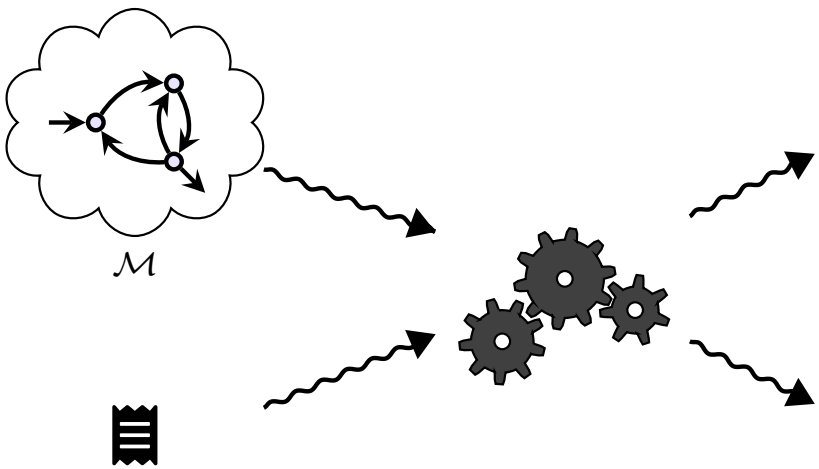
La machine universelle
Utilisation habituelle d'une machine de Turing



il existe une exécution acceptante de $\mathcal{M}$

il n'existe pas d'exécution acceptante de $\mathcal{M}$

La machine universelle
Utilisation habituelle d'une machine de Turing

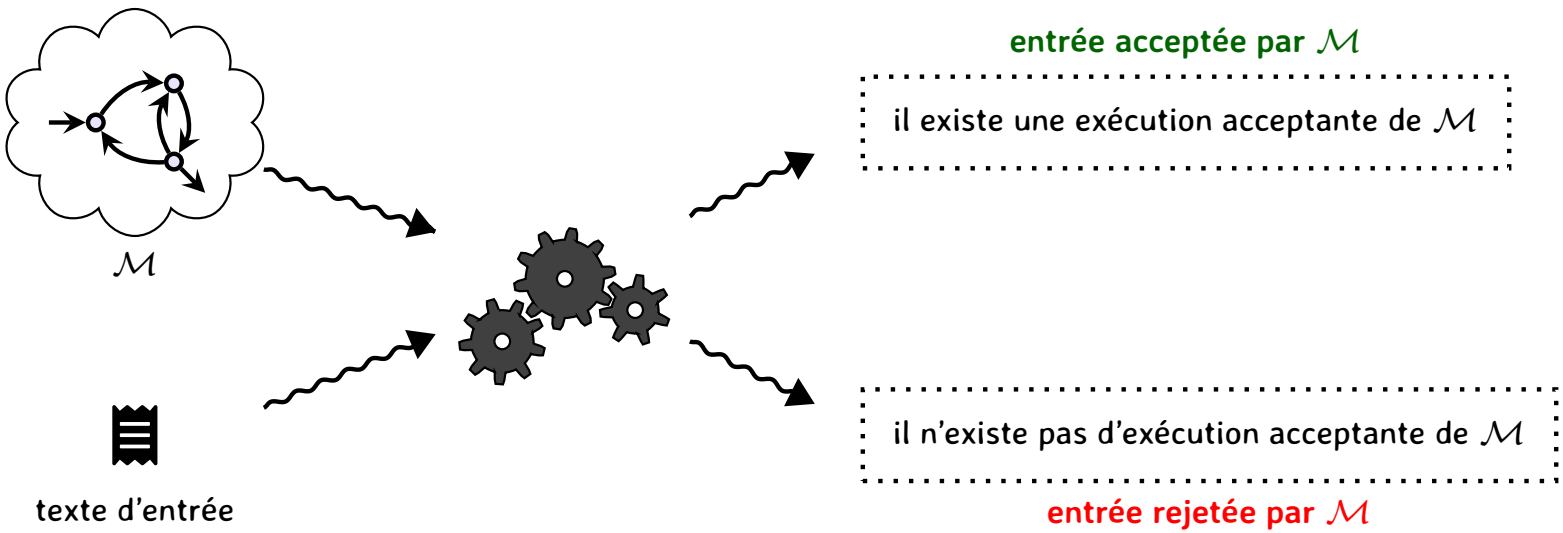


entrée acceptée par $\mathcal{M}$

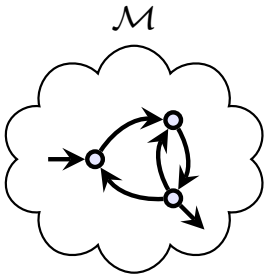
il existe une exécution acceptante de $\mathcal{M}$

il n'existe pas d'exécution acceptante de $\mathcal{M}$

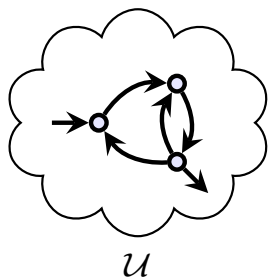
La machine universelle
Utilisation habituelle d'une machine de Turing



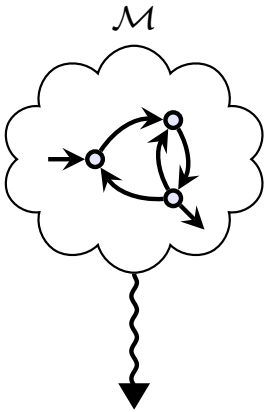
La machine universelle
Utilisation d'une machine de Turing via la machine universelle

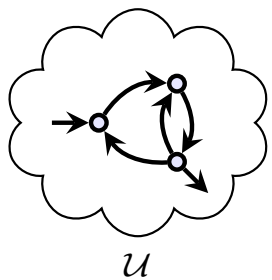


texte d'entrée

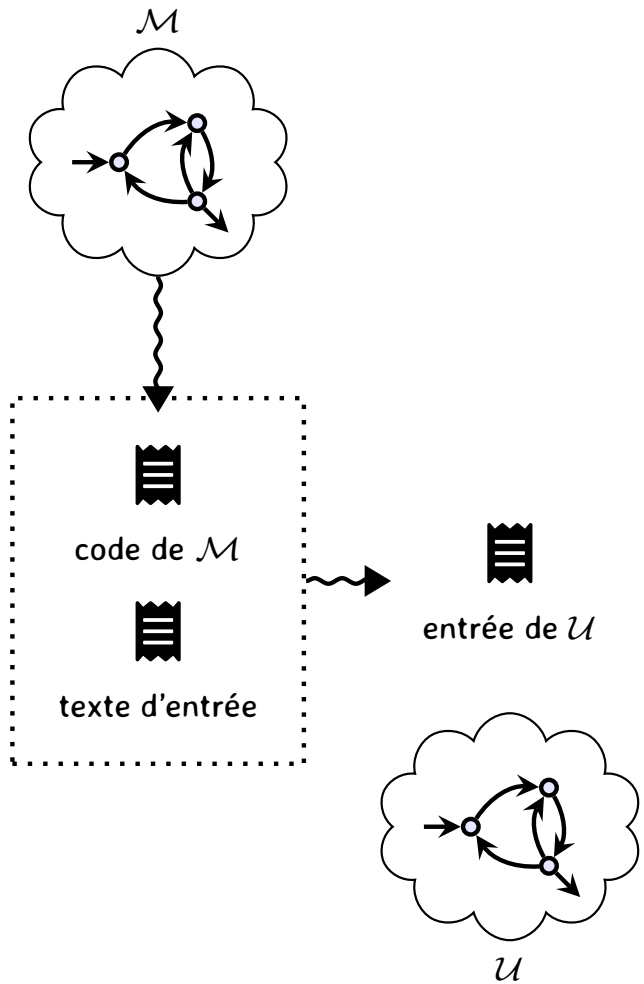


La machine universelle
Utilisation d'une machine de Turing via la machine universelle

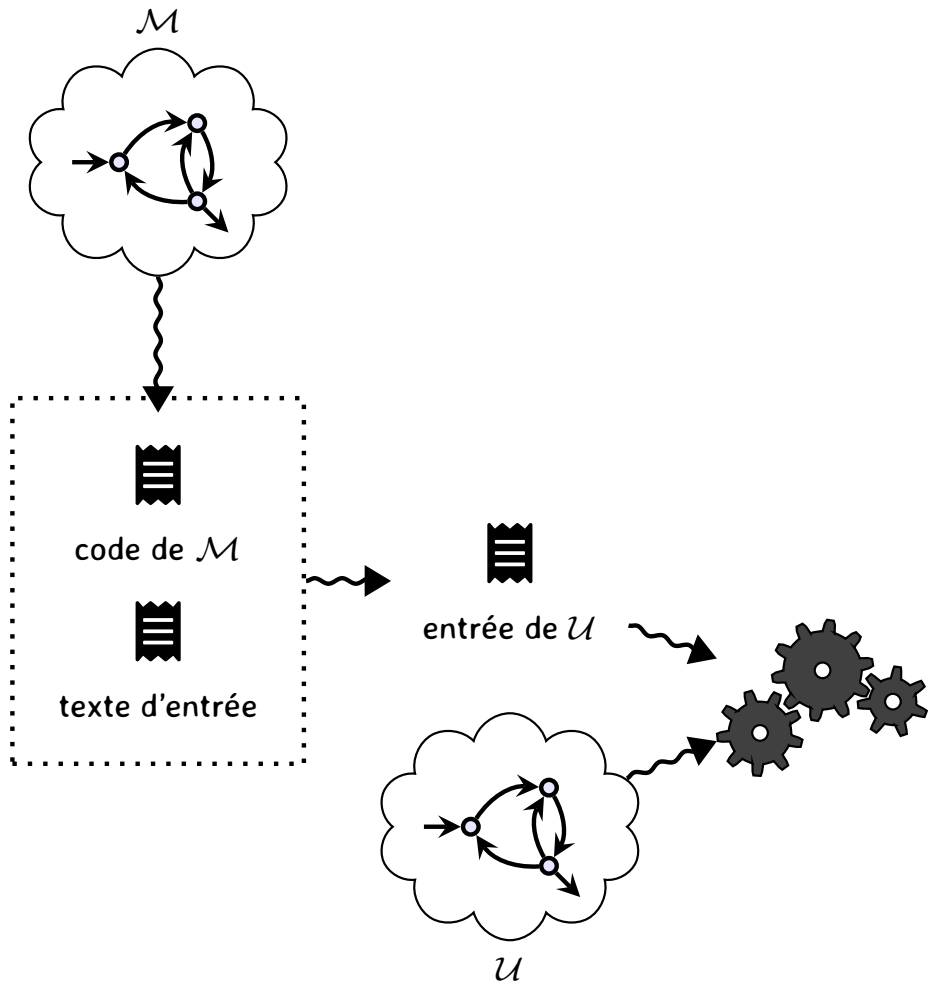




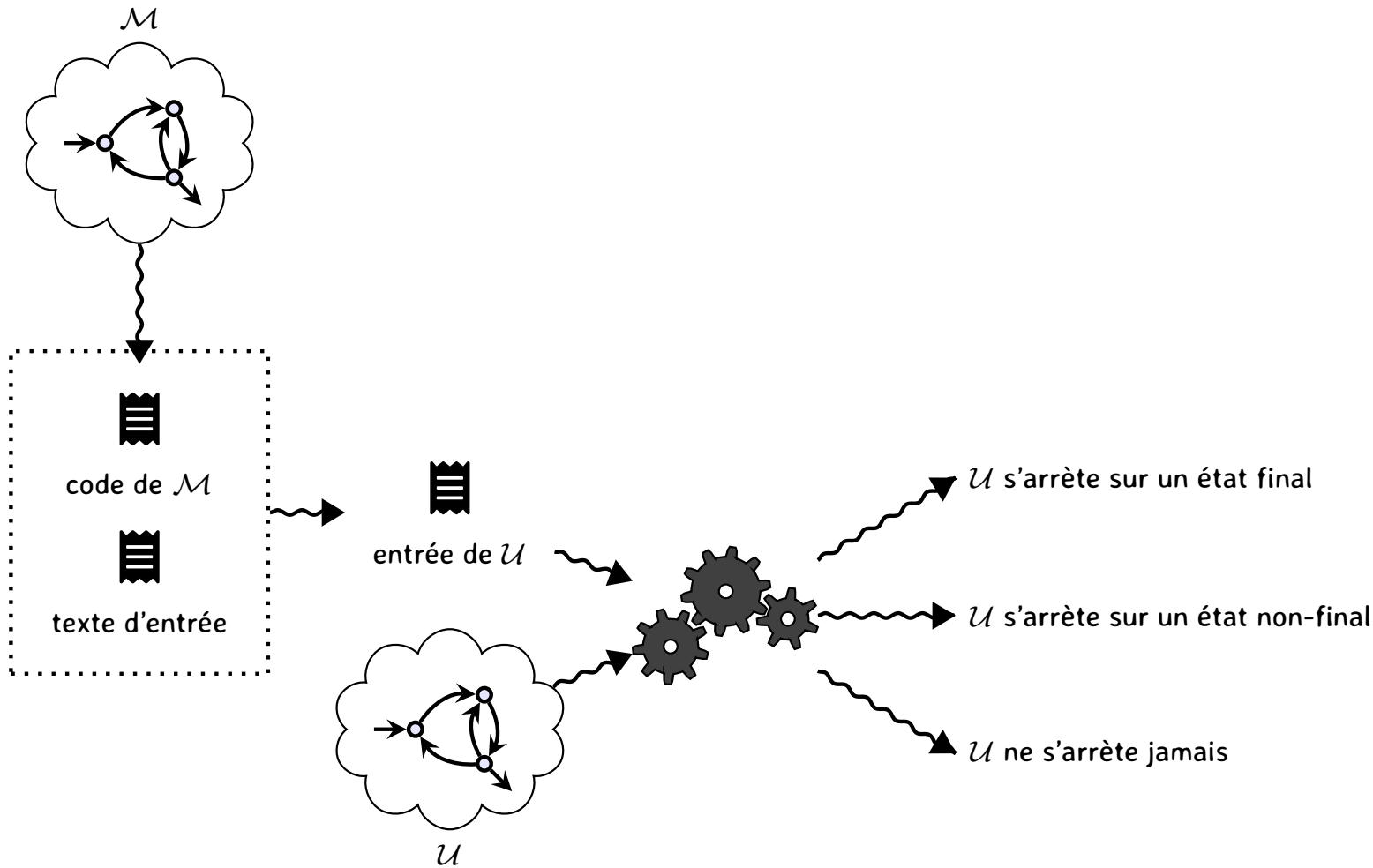
La machine universelle
Utilisation d'une machine de Turing via la machine universelle



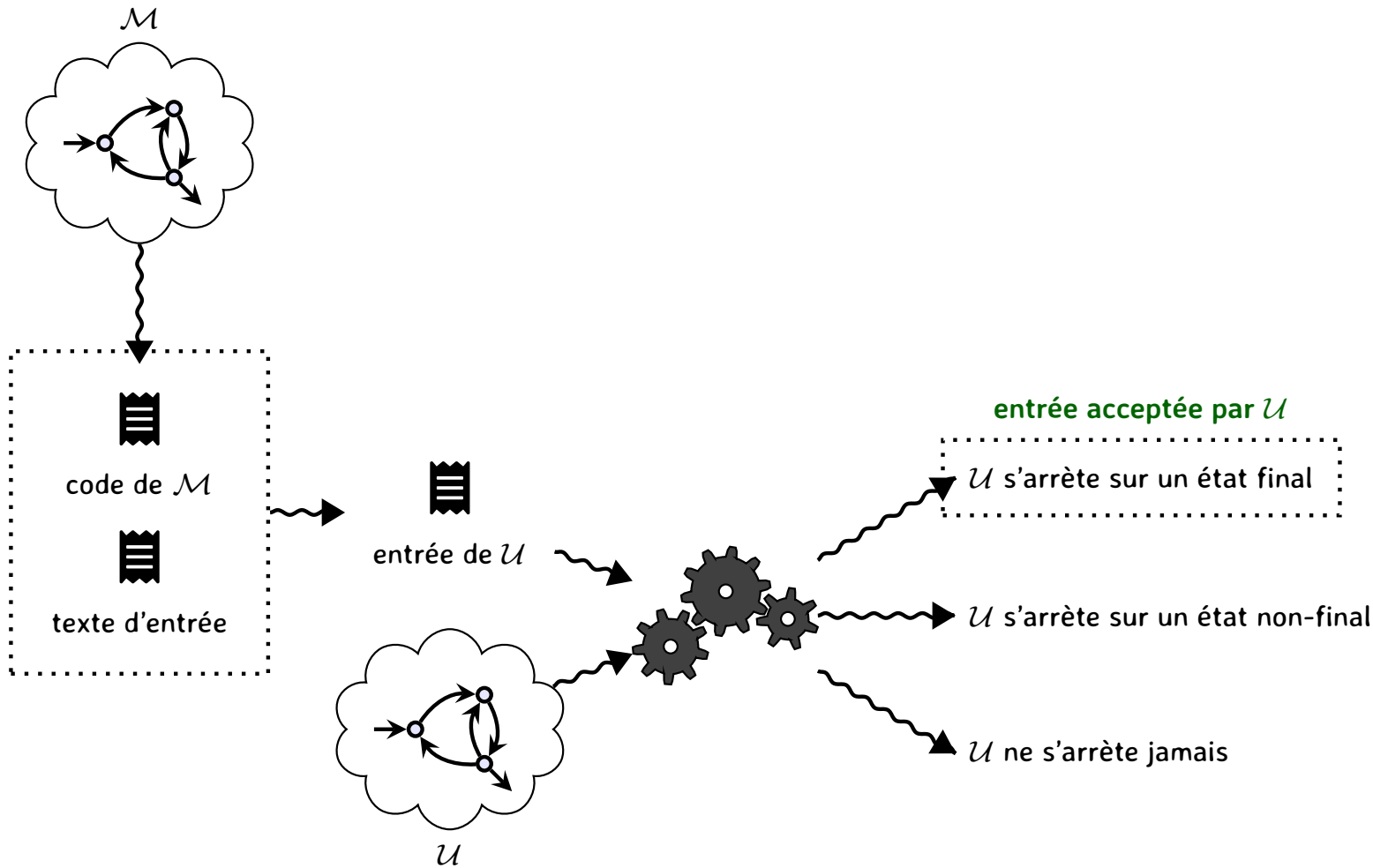
La machine universelle
Utilisation d'une machine de Turing via la machine universelle



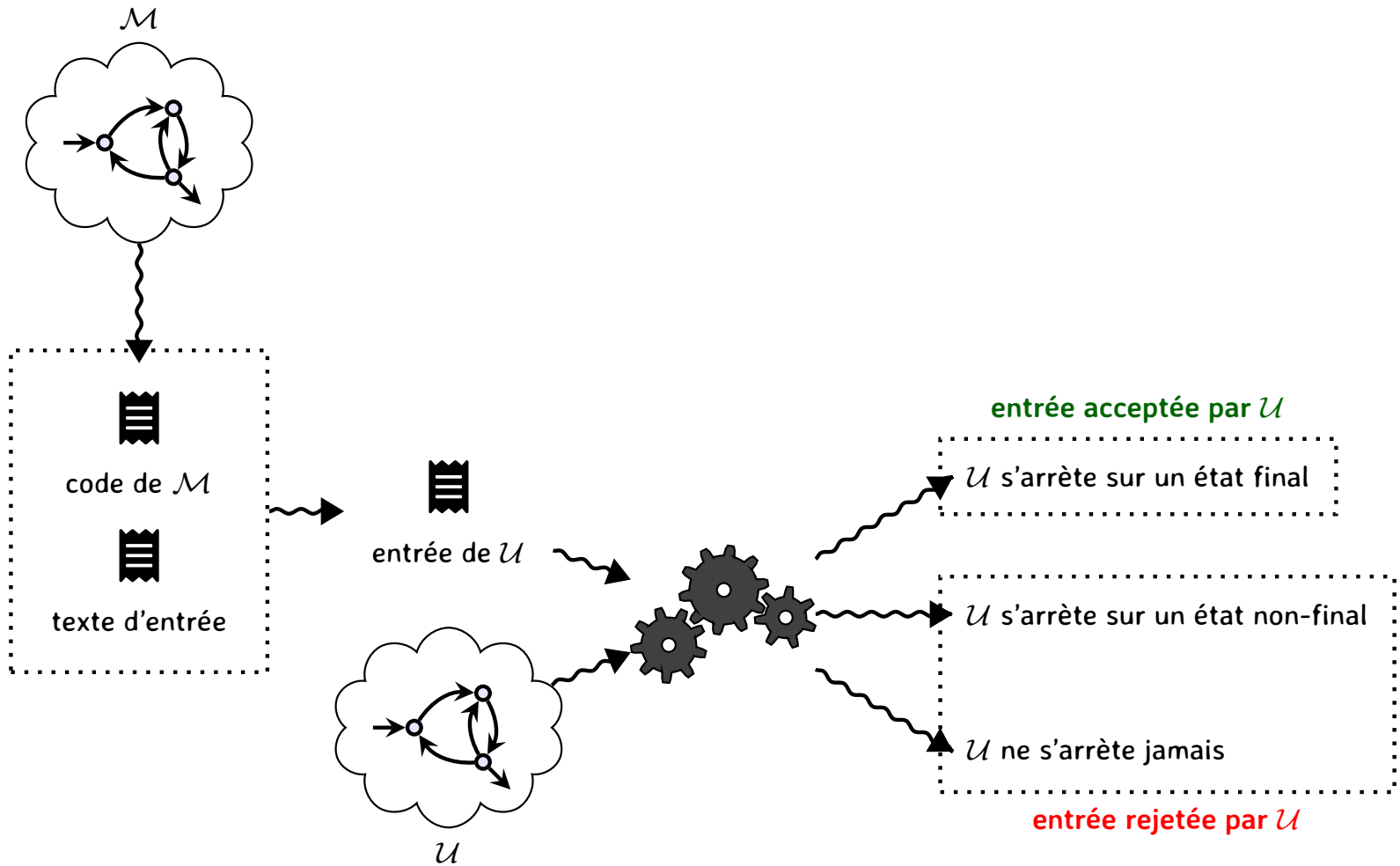
La machine universelle
Utilisation d'une machine de Turing via la machine universelle



La machine universelle
Utilisation d'une machine de Turing via la machine universelle



La machine universelle
Utilisation d'une machine de Turing via la machine universelle



À l'entrée de la machine universelle
Codage de $\mathcal{M}$

$$\mathcal{M} := \langle Q, \Sigma, \Gamma, \Delta, q_0, F \rangle \, .$$

À l'entrée de la machine universelle

Codage de $\mathcal{M}$

états : entiers consécutifs $0, \ldots, n$.

$$\mathcal{M} := \langle Q, \Sigma, \Gamma, \Delta, q_0, F \rangle .$$

☞ On code les entiers en binaire : $[n]_{code} \in \{0,1\}^\star$.

À l'entrée de la machine universelle

Codage de $\mathcal{M}$

états : entiers consécutifs $0, \ldots, n$.

$$\mathcal{M} := \langle Q, \Sigma, \Gamma, \Delta, q_0, F \rangle .$$

alphabets : on fixe $\Sigma = \{0, 1\}$ et $\Gamma = \{0, 1, \boxplus\}$.

- On code les entiers en binaire : $[n]_{code} \in \{0, 1\}^\star$.
- Pour éviter les ambiguités, on distingue le symbole case vide $\boxplus$ de $\mathcal{M}$ du symbole case vide # de la machine universelle.

À l'entrée de la machine universelle

Codage de $\mathcal{M}$

états : entiers consécutifs $0, \ldots, n$.

état initial : on fixe $q_0 = 0$.

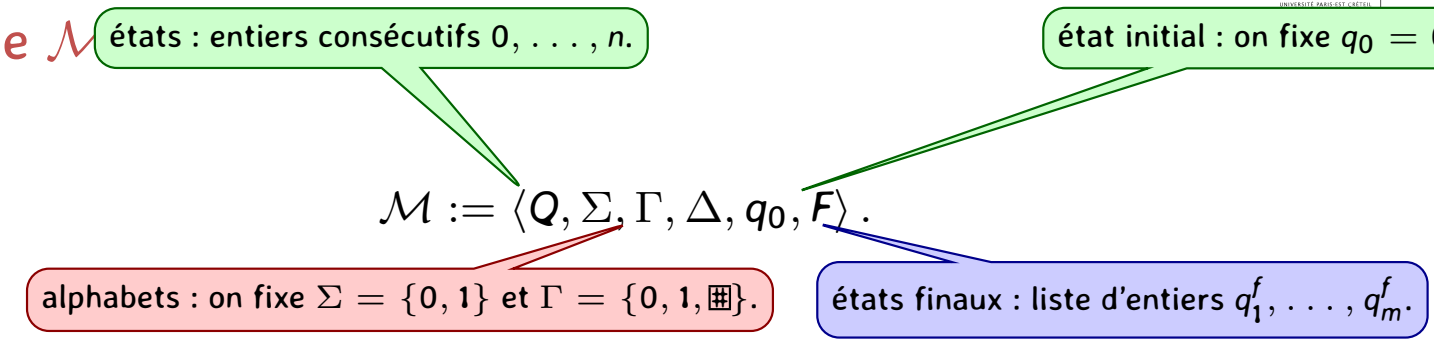
$$\mathcal{M} := \langle Q, \Sigma, \Gamma, \Delta, q_0, F \rangle .$$

alphabets : on fixe $\Sigma = \{0, 1\}$ et $\Gamma = \{0, 1, \boxplus\}$.

- On code les entiers en binaire : $[n]_{code} \in \{0, 1\}^\star$.
- Pour éviter les ambiguités, on distingue le symbole case vide $\boxplus$ de $\mathcal{M}$ du symbole case vide # de la machine universelle.

À l'entrée de la machine universelle

Codage de $\mathcal{M}$



- ☞ On code les entiers en binaire : $[n]_{code} \in \{0, 1\}^\star$.
- ☞ Pour éviter les ambiguités, on distingue le symbole case vide $\boxplus$ de $\mathcal{M}$ du symbole case vide # de la machine universelle.

À l'entrée de la machine universelle

Codage de $\mathcal{M}$

Relation de transition : liste de transitions $t_1, \ldots, t_k$.

états : entiers consécutifs $0, \ldots, n$.

état initial : on fixe $q_0 = 0$.

$$\mathcal{M} := \langle Q, \Sigma, \Gamma, \Delta, q_0, F \rangle .$$

alphabets : on fixe $\Sigma = \{0, 1\}$ et $\Gamma = \{0, 1, \boxplus\}$.

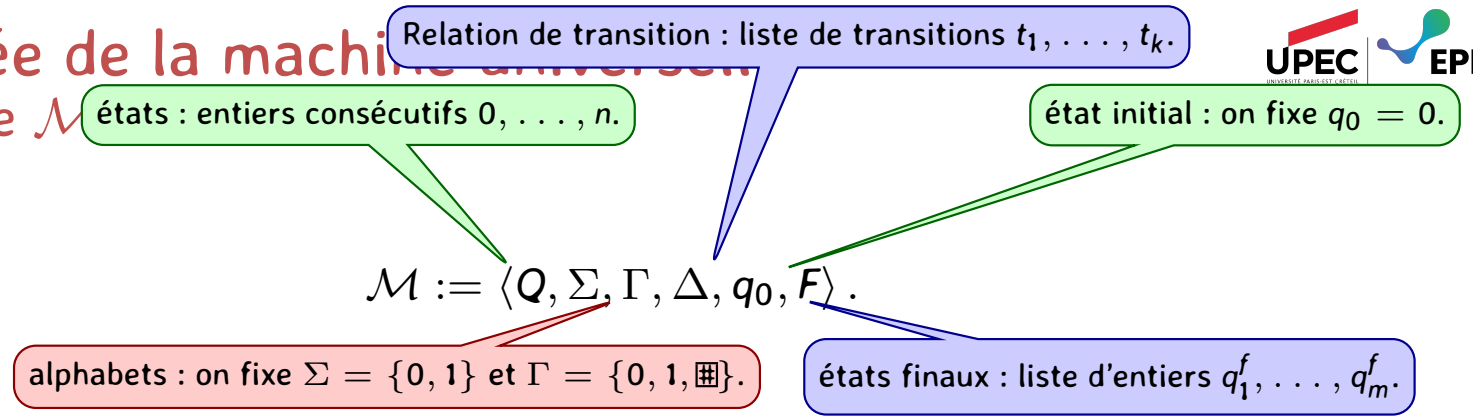
états finaux : liste d'entiers $q_1^f, \ldots, q_m^f$.

- On code les entiers en binaire : $[n]_{code} \in \{0, 1\}^\star$.
- Pour éviter les ambiguités, on distingue le symbole case vide $\boxplus$ de $\mathcal{M}$ du symbole case vide # de la machine universelle.
- Chaque transition : on encode les directions $\{\blacktriangleleft, \blacktriangleright\}$ par les symboles $\{\mathtt{G}, \mathtt{D}\}$, et la transition $q_1 \xrightarrow{a/b,dir} q_2$ par le mot

$$[q_1]_{code} \; \S \; a \; \S \; b \; \S \; [dir]_{code} \; \S \; [q_2]_{code} \in \{0, 1, \boxplus, \mathtt{D}, \mathtt{G}, \S\}^\star .$$

À l'entrée de la machine universelle

Codage de $\mathcal{M}$



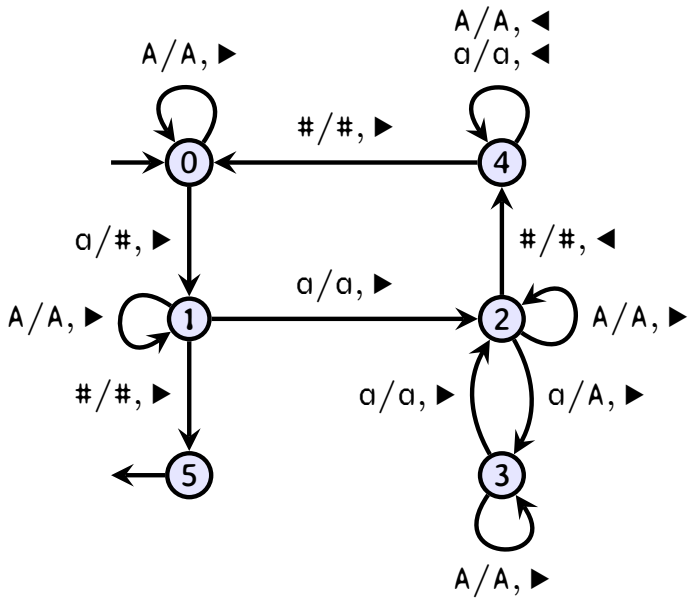
- On code les entiers en binaire : $[n]_{code} \in \{0, 1\}^\star$.
- Pour éviter les ambiguités, on distingue le symbole case vide # de $\mathcal{M}$ du symbole case vide # de la machine universelle.
- Chaque transition : on encode les directions $\{\blacktriangleleft, \blacktriangleright\}$ par les symboles $\{G, D\}$, et la transition $q_1 \xrightarrow{a/b,dir} q_2$ par le mot

$$[q_1]_{code} \; \S \; a \; \S \; b \; \S \; [dir]_{code} \; \S \; [q_2]_{code} \in \{0, 1, \#, D, G, \S\}^\star .$$

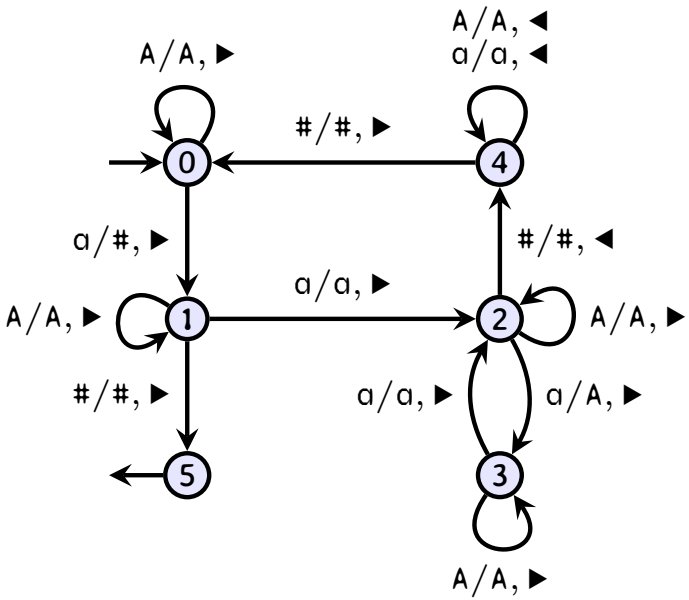
- Finalement on obtient le code de la machine $\mathcal{M}$:

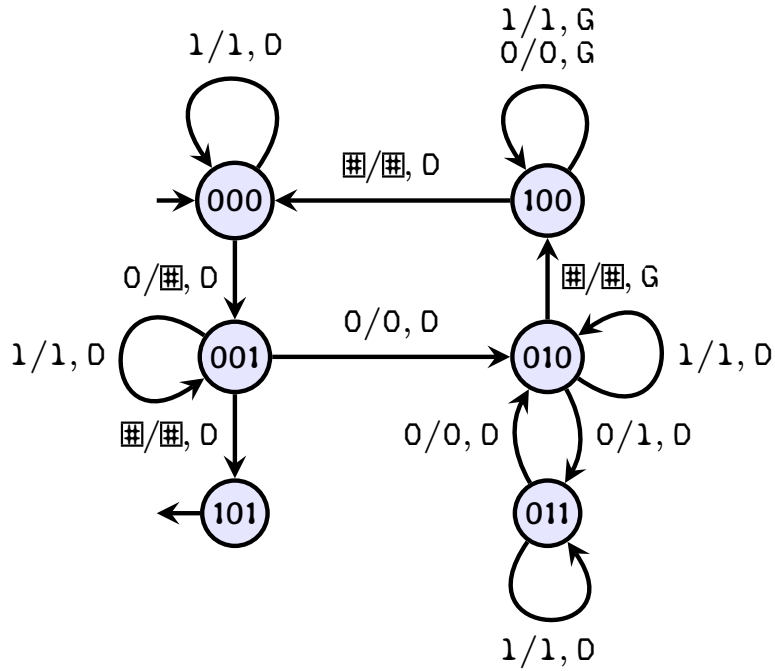
$$[\mathcal{M}]_{code} = \$ \; [t_1]_{code} \mid \ldots \mid [t_k]_{code} \; \$ \; \left[q_1^f\right]_{code} \mid \ldots \mid \left[q_m^f\right]_{code}$$
$$\in \{0, 1, \#, D, G, \S, \$, \mid\}^\star$$

Codage de $\mathcal{M}$
Exemple

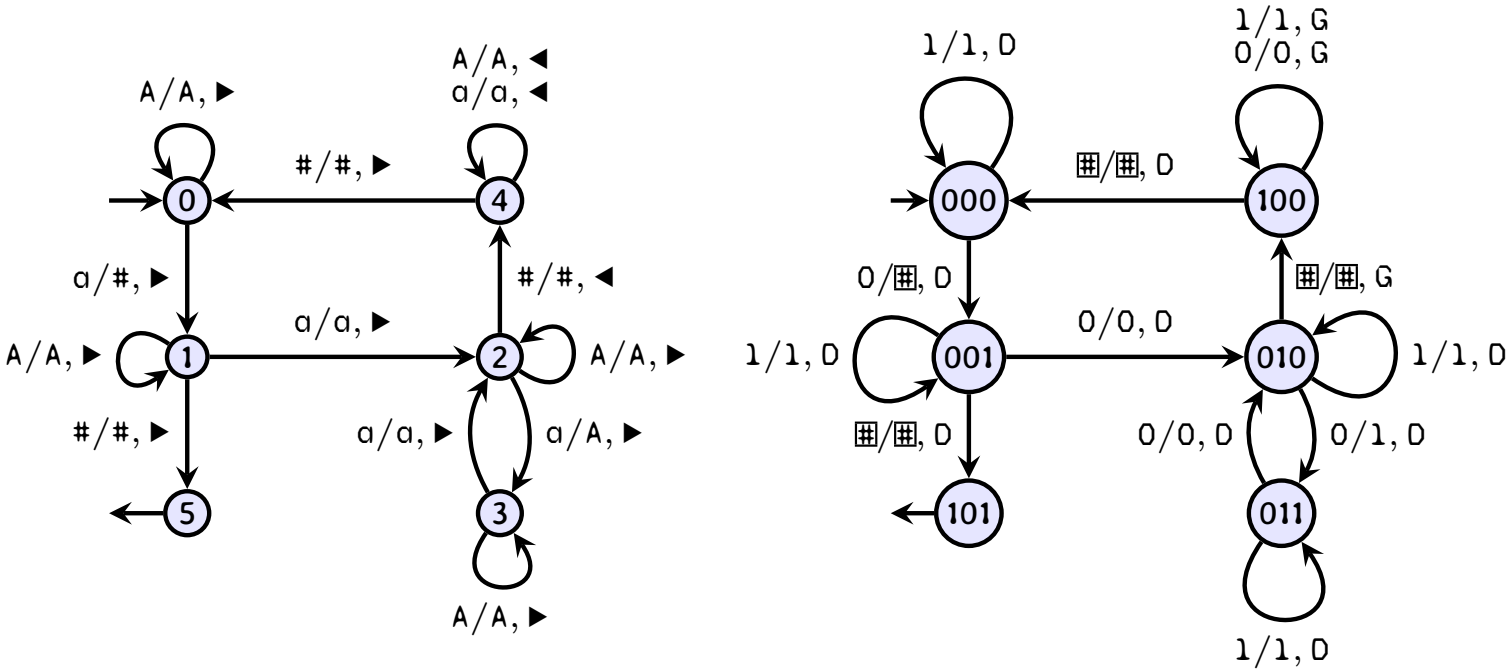


Codage de $\mathcal{M}$
Exemple



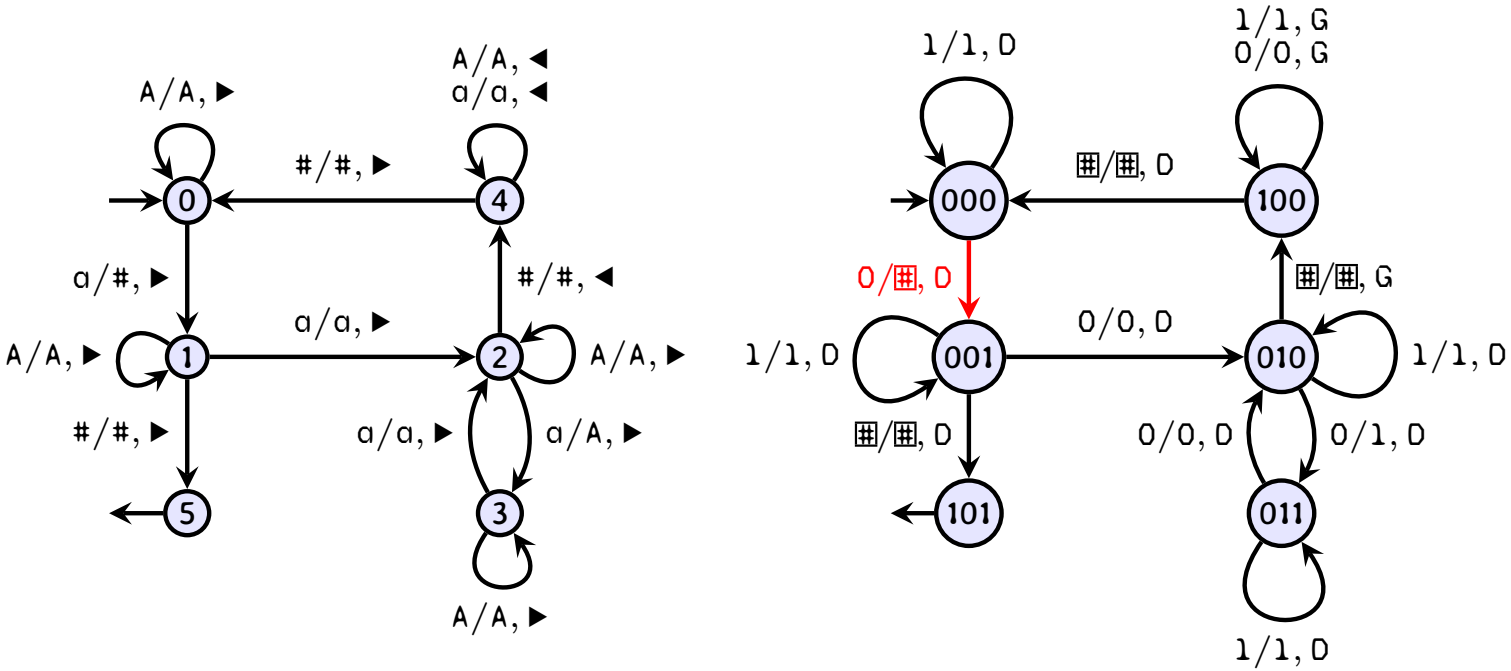


Codage de $\mathcal{M}$
Exemple



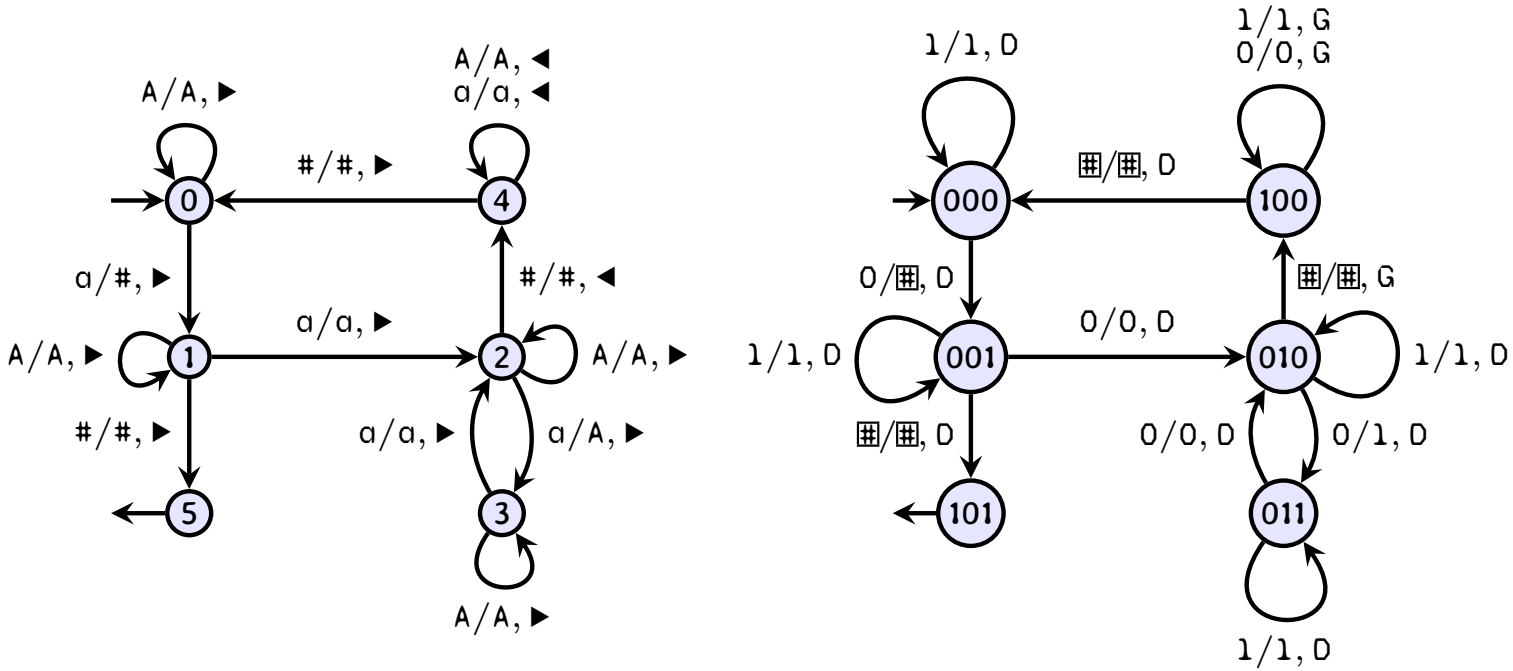
$$[\mathcal{M}]_{code} = \$\,000\S1\S1\S D\S000 \mid 000\S0\S\boxplus\S D\S001 \mid \ldots\ \$101$$

Codage de $\mathcal{M}$
Exemple



$$[\mathcal{M}]_{code} = \$\,000\S1\S1\S D\S000 \mid 000\S0\S\boxplus\S D\S001 \mid \ldots\ \$101$$

Codage de $\mathcal{M}$
Exemple



$$[\mathcal{M}]_{code} = \$\,000\S1\S1\S D\S000 \mid 000\S0\S\boxplus\S D\S001 \mid \ldots\ \$101$$

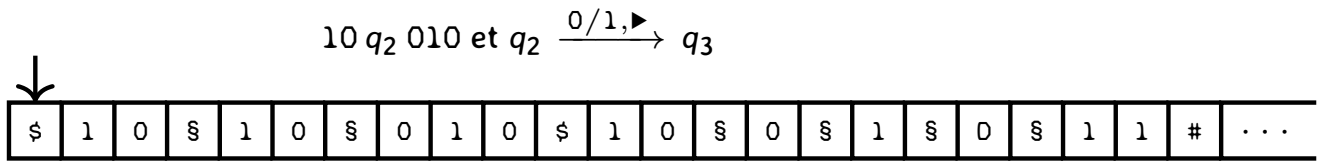
exercice

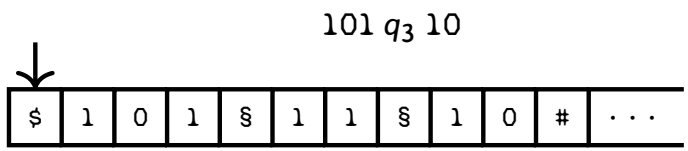
Montrer que le langage $L = \{[\mathcal{M}]_{code} \mid \mathcal{M} \text{ machine de Turing}\}$ est décidable.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition

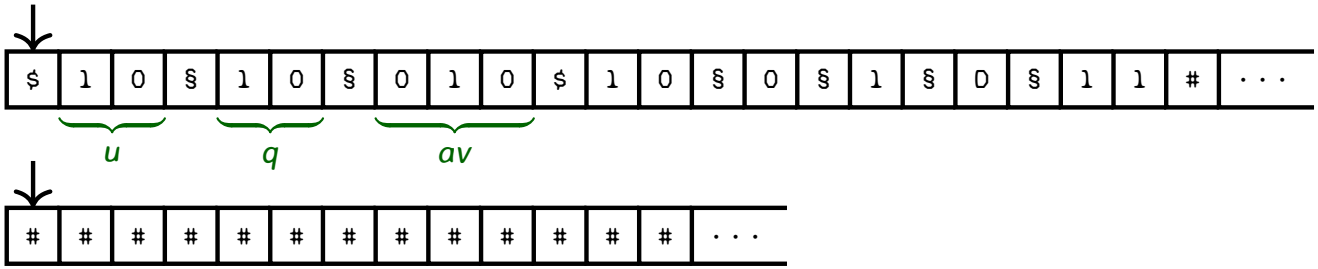
On encode une configuration $u\,q\,v$ par le mot $u\S\,[q]_{code}\,\S v$.
Supposons qu'on nous donne une configuration et une transition, on peut construire un machine $\mathcal{M}_\Delta$ à deux rubans qui :

- rejette si la transition ne peut pas être activée depuis cette configuration ;
- sinon, elle écrit sur son ruban d'entrée la configuration après avoir exécuté la transition, et elle accepte.



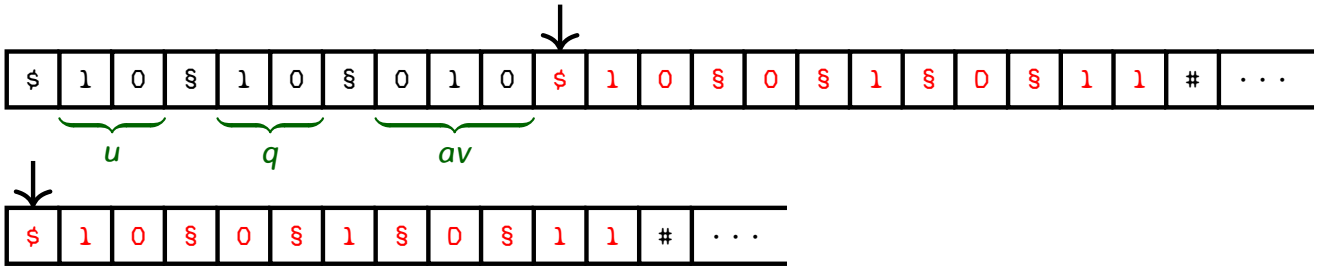


Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



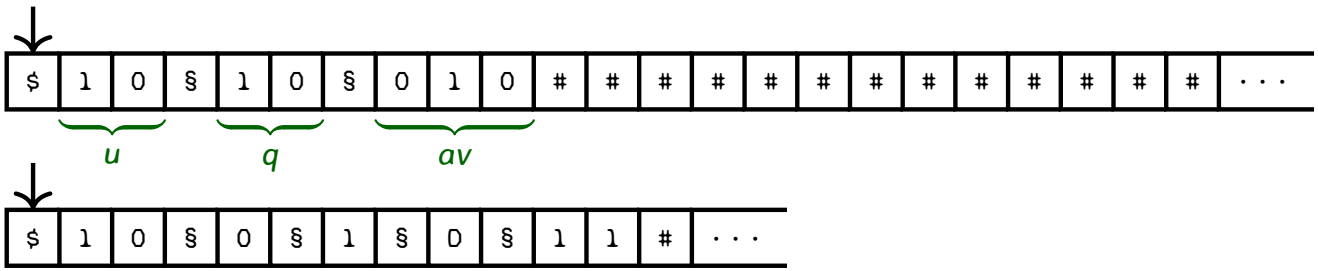
☞ On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



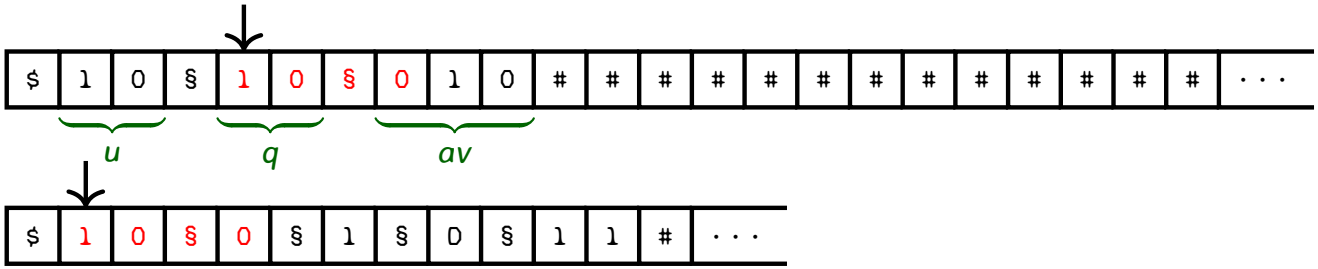
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



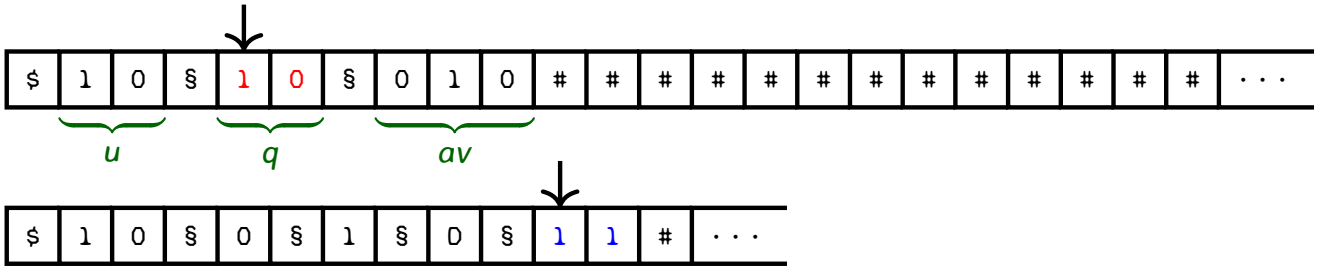
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



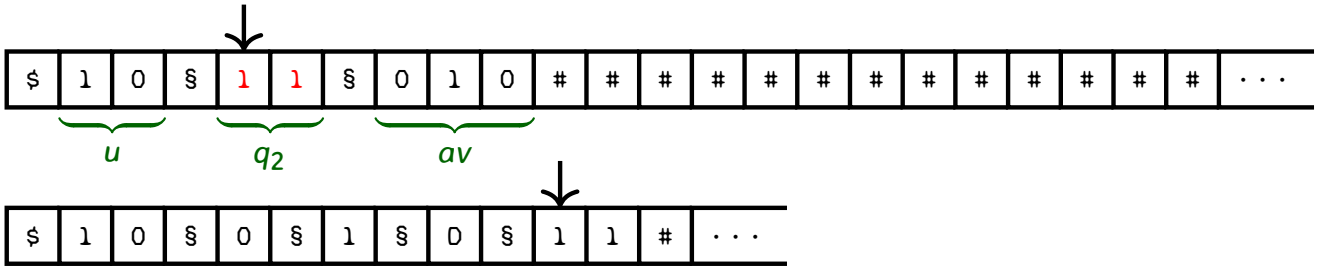
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



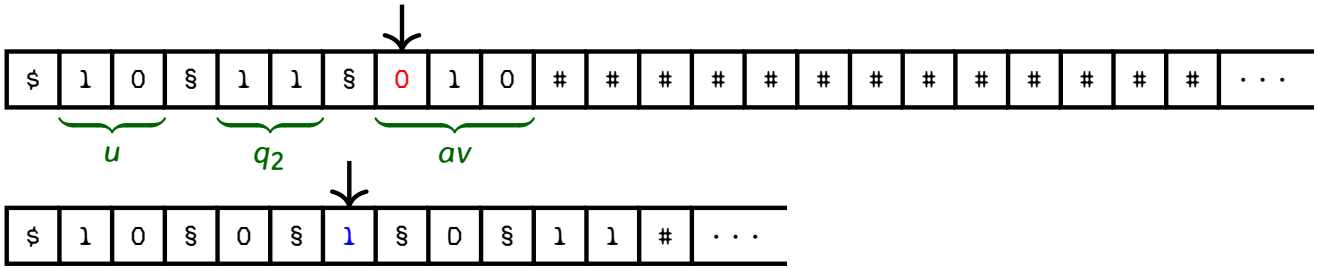
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2,

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



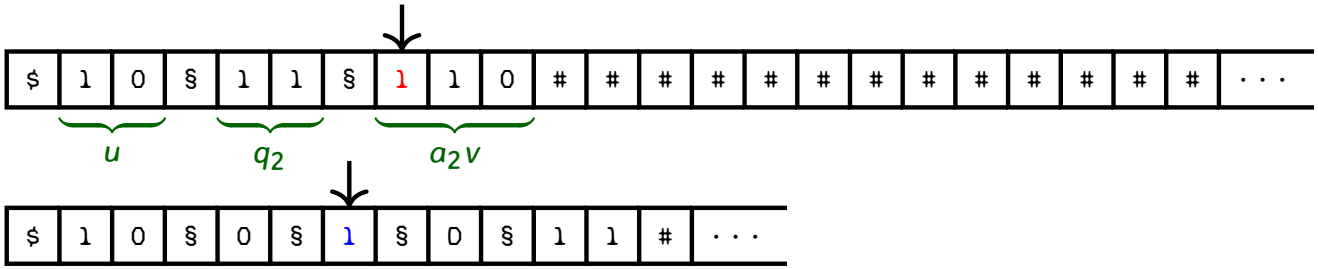
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2\rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2,

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



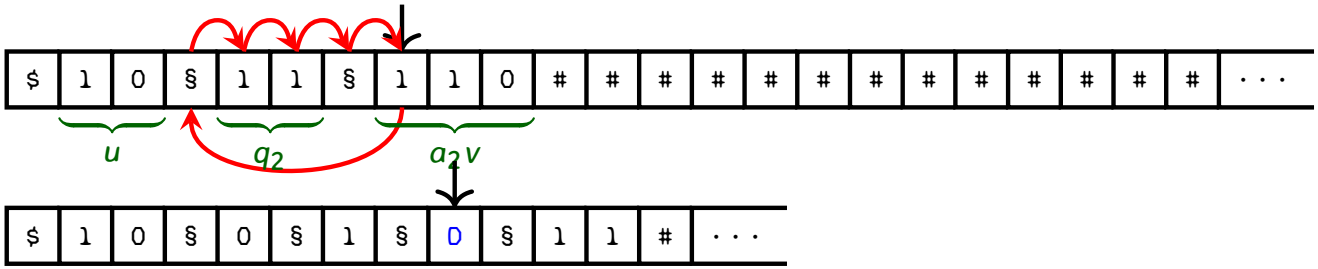
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2, et a par a_2.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



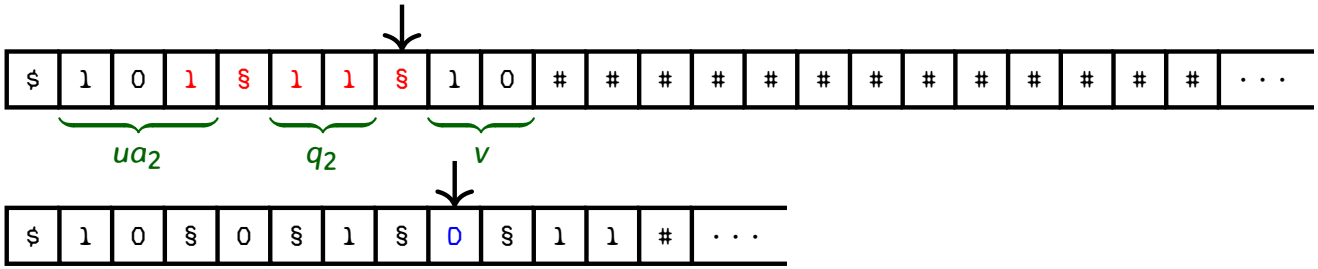
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2\rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2, et a par a_2.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



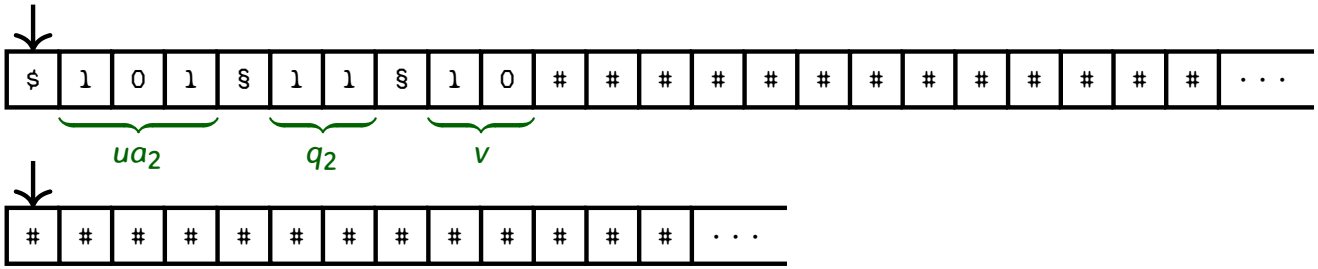
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2\rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2, et a par a_2.
- On regarde quelle est la direction de déplacement,

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



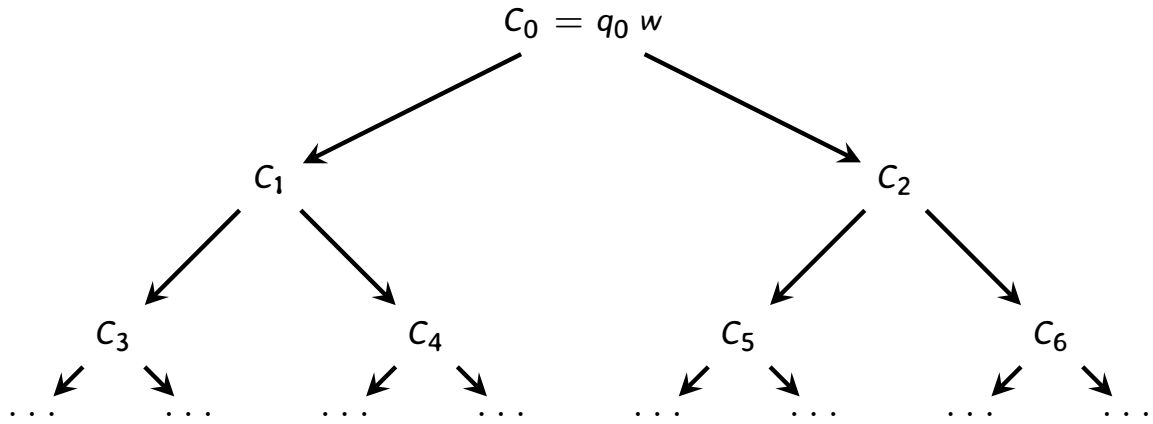
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2\rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2, et a par a_2.
- On regarde quelle est la direction de déplacement, et on déplace la tête de lecture.

Simuler l'exécution de $\mathcal{M}$
Exécuter une transition



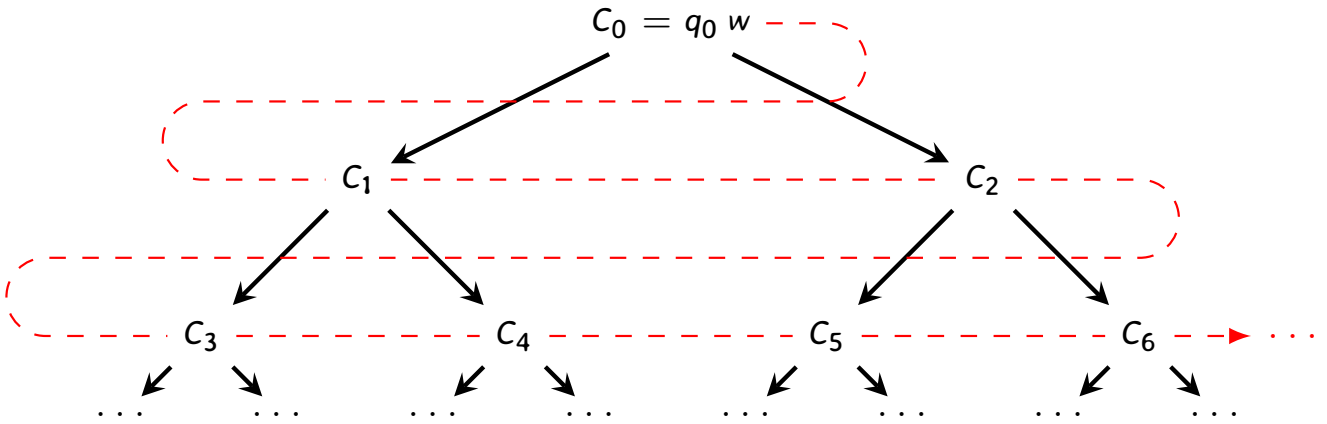
- On commence avec le codage de la paire $\langle u\,q\,av, q_1 \xrightarrow{a_1/a_2,dir} q_2 \rangle$ sur le premier ruban.
- On déplace la transition sur le deuxième ruban.
- On l'efface du premier ruban, et on replace les deux têtes au début de leurs rubans.
- On vérifie que $q = q_1$ et $a = a_1$:
  - si ça n'est pas le cas, on efface les deux rubans et on va sur l'état de rejet (q_{non}) ;
  - sinon on poursuit.
- On remplace q par q_2, et a par a_2.
- On regarde quelle est la direction de déplacement, et on déplace la tête de lecture.
- Enfin, on nettoie le second ruban, on repositionne la tête de lecture sur le premier, et on va sur l'état d'acceptation q_{oui}.

Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



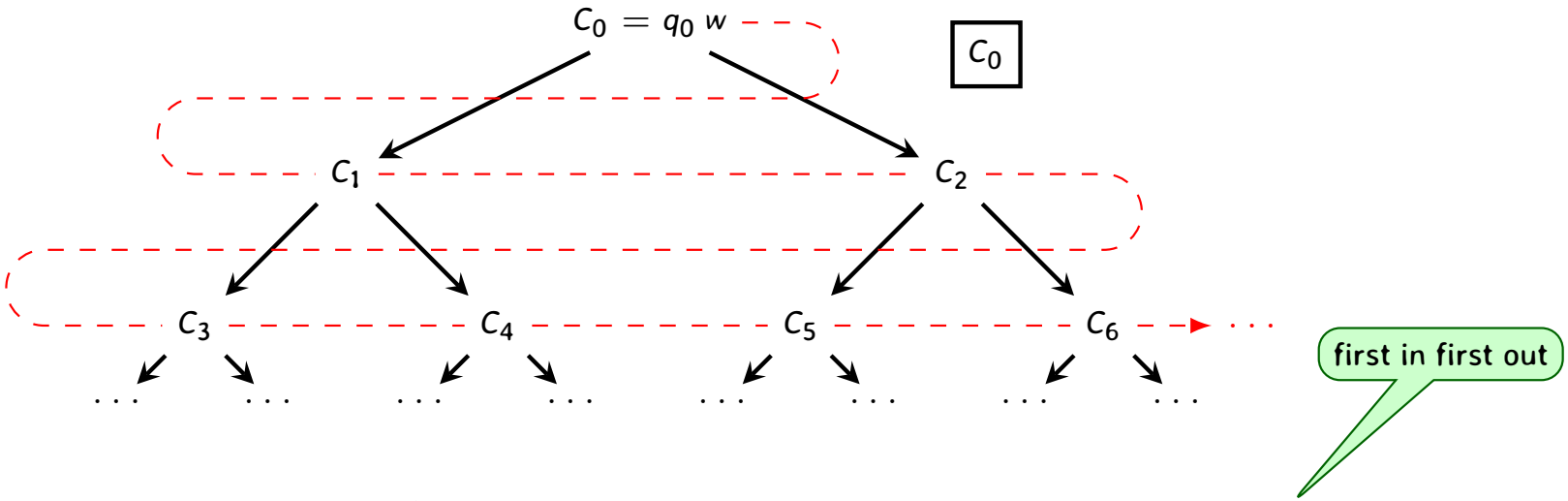
On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

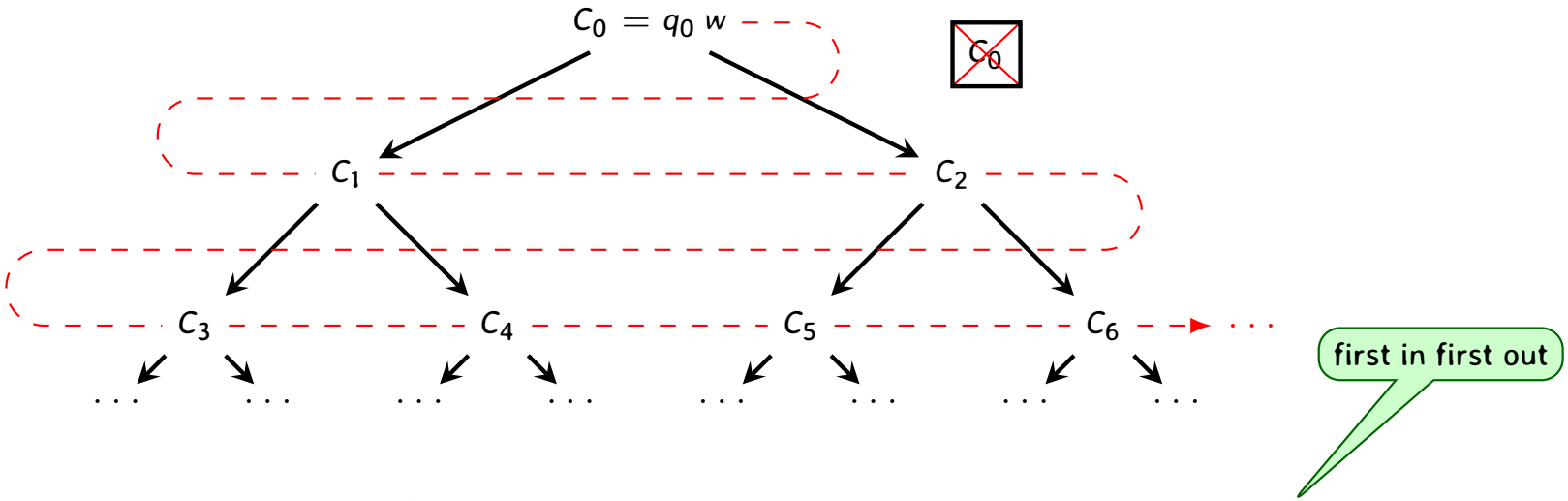
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.

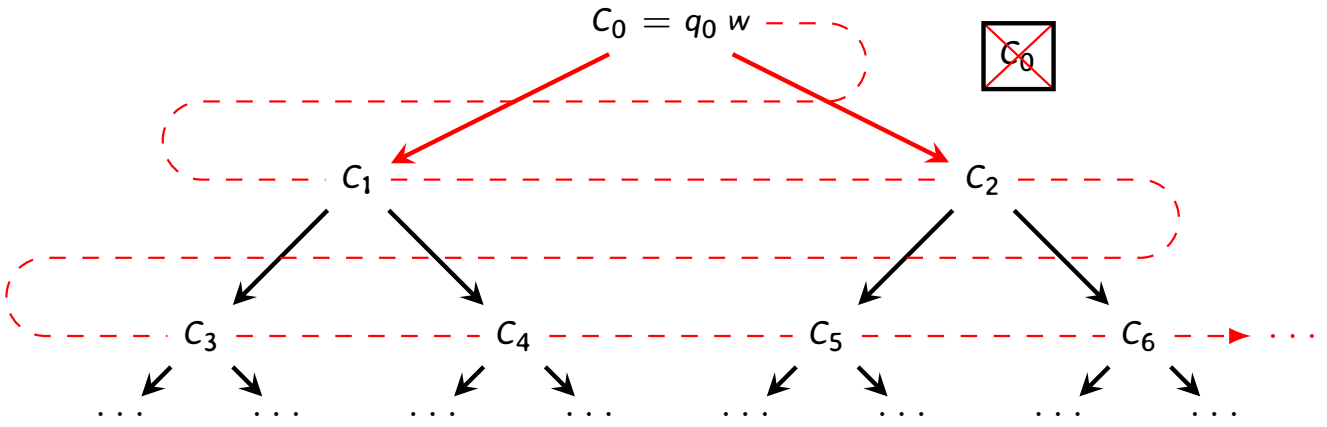
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.

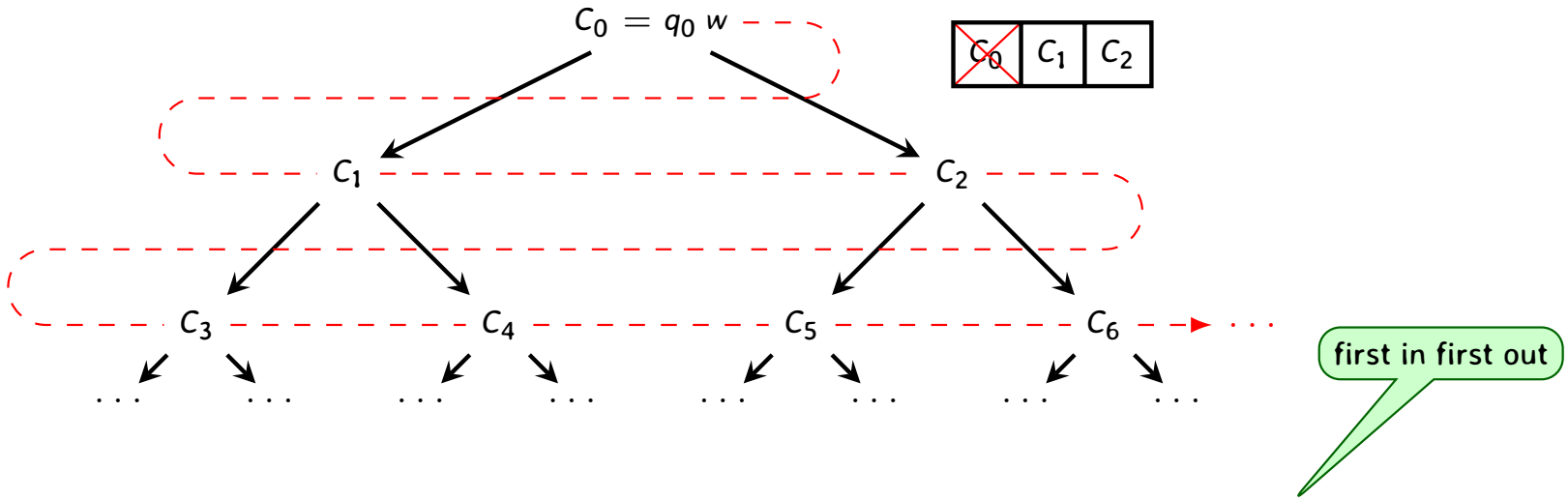
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;

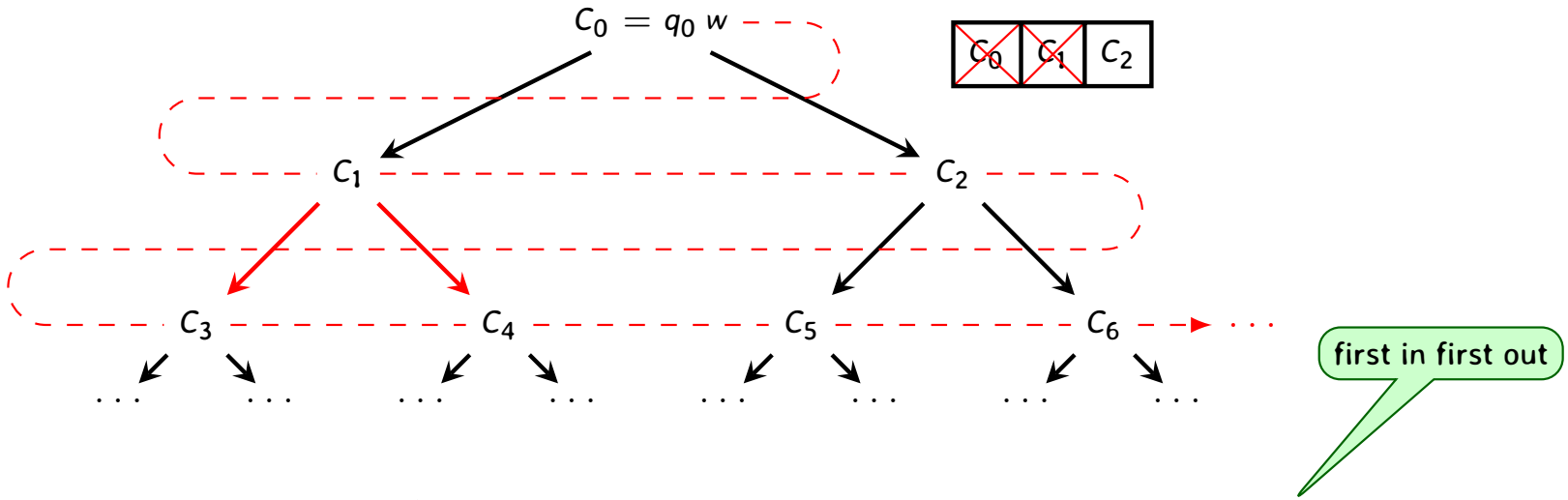
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.

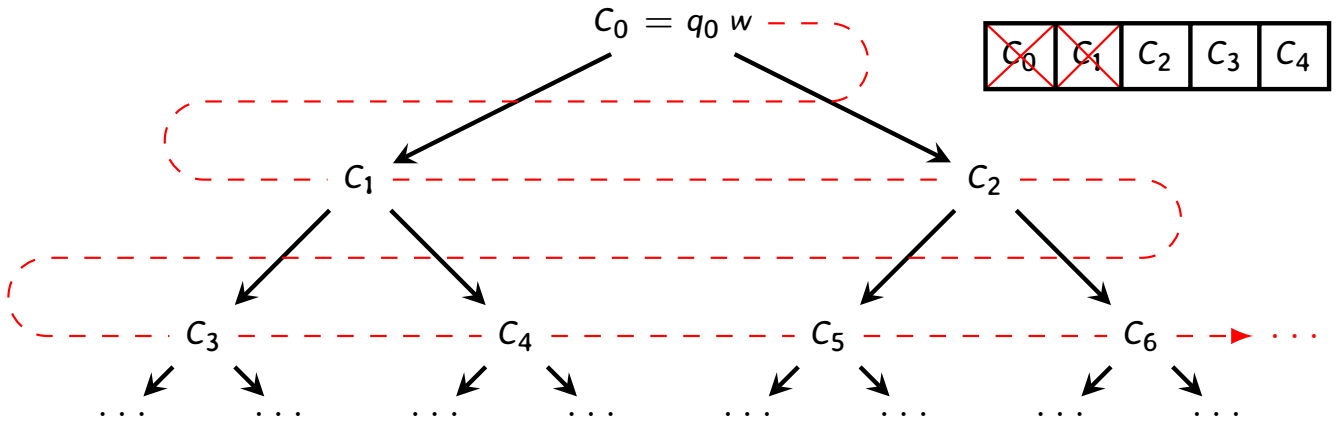
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.

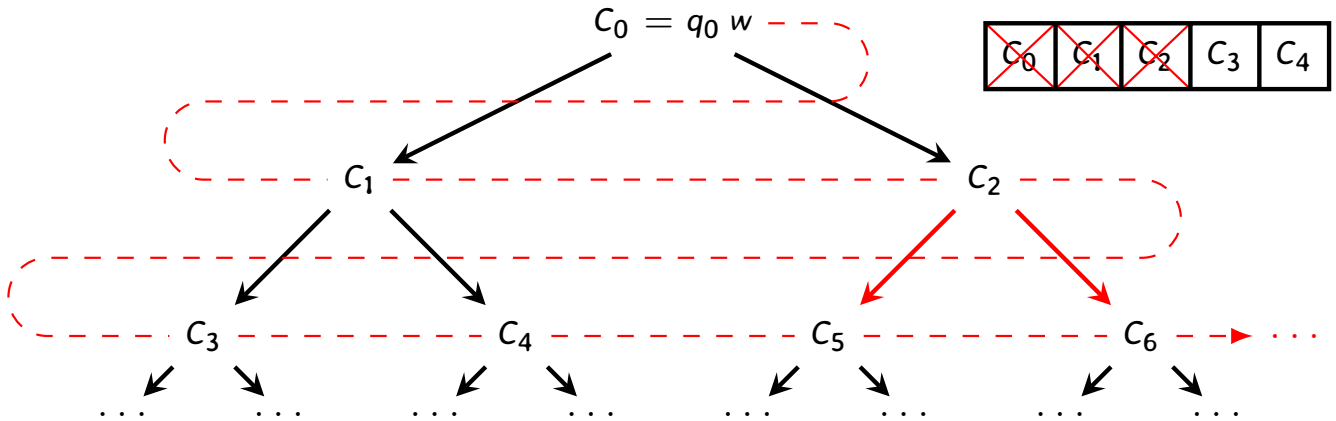
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.

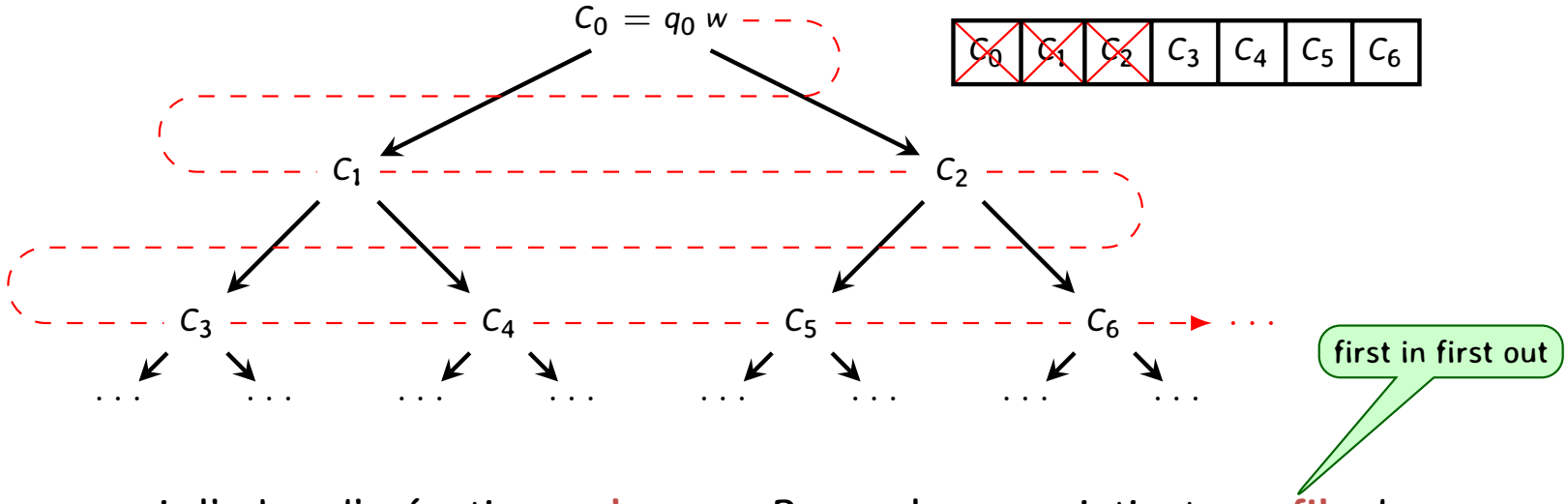
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.

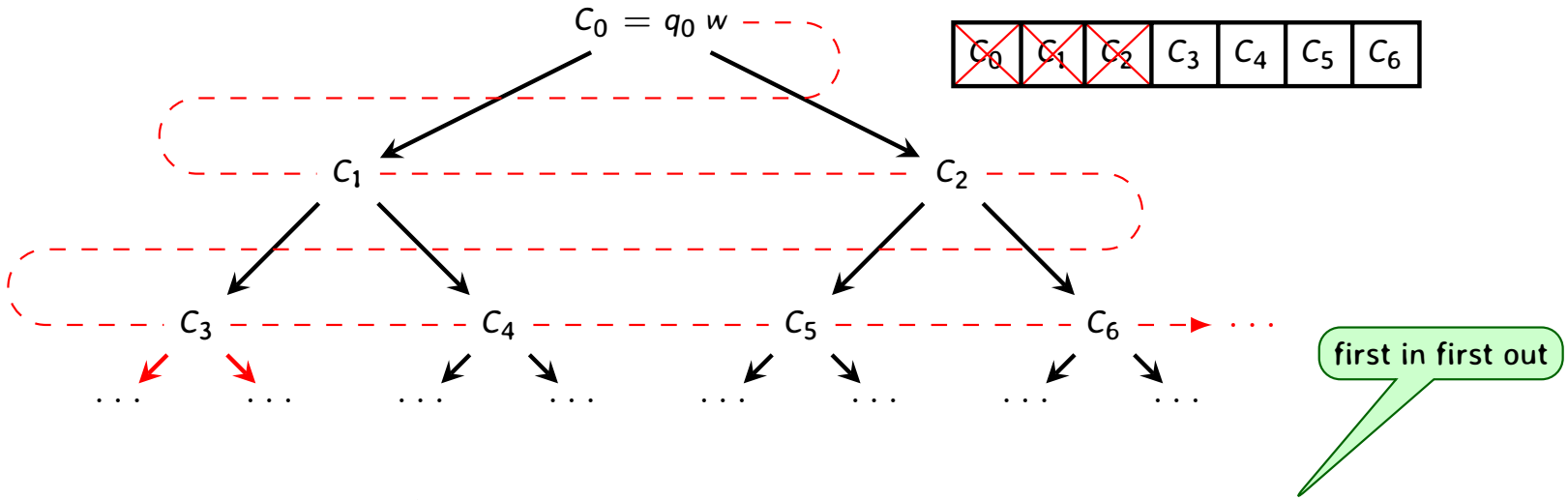
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.

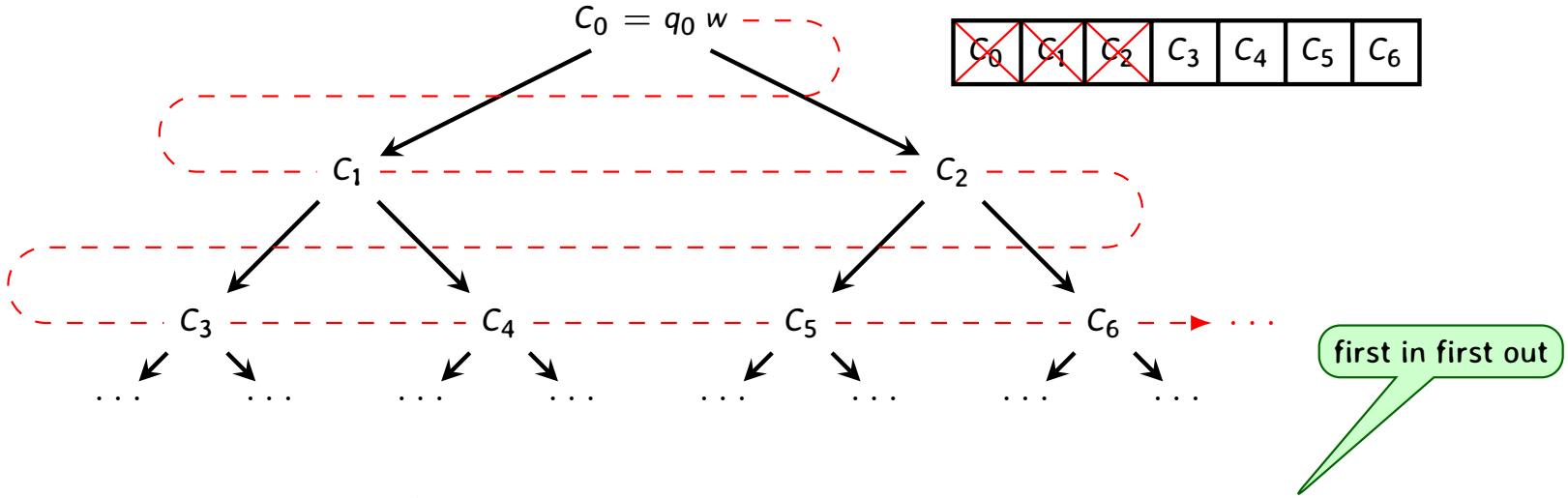
Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.

Simuler l'exécution de $\mathcal{M}$
Déterminisation : idée de l'algorithme



On va parcourir l'arbre d'exécution **en largeur**. Pour cela, on maintient une **file** de configurations.

1) On initialise la file avec $C_0 = \S\, [q_0]_{code}\, \S w$.
2) Tant que la file n'est pas vide :
   - 2.1 On enlève la première configuration C de la file ;
   - 2.2 Si C est acceptante, on **accepte** l'entrée.
   - 2.3 Sinon, pour toutes les transitions t de la machine :
     - 2.3.1 On essaie d'exécuter t à partir de C ;
     - 2.3.2 Si c'est impossible, on continue ;
     - 2.3.3 Sinon, on obtient C' telle que $C \to_{\mathcal{M}} C'$, et on l'ajoute à la fin de la file.
3) Si la file est vide et si on n'a pas trouvé de configuration acceptante, on **rejette**.

Langage accepté par la machine universelle

☞ Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.

Langage accepté par la machine universelle

☞ Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.

Langage accepté par la machine universelle

☞ Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.

- Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.

- Sinon, on passe à la simulation de $\mathcal{M}$.

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.
  - Si il existe une configuration acceptante accessible depuis la configuration initiale de $\mathcal{M}$ sur w, c'est à dire $q_0\, w$, on **accepte**.

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.
  - Si il existe une configuration acceptante accessible depuis la configuration initiale de $\mathcal{M}$ sur w, c'est à dire $q_0\, w$, on **accepte**.
  - Si l'arbre d'exécution est fini, i.e. il n'y a pas d'exécutions infinie, et si aucune configuration acceptante n'est accessible, on **rejette**.

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.
  - Si il existe une configuration acceptante accessible depuis la configuration initiale de $\mathcal{M}$ sur w, c'est à dire $q_0\, w$, on **accepte**.
  - Si l'arbre d'exécution est fini, i.e. il n'y a pas d'exécutions infinie, et si aucune configuration acceptante n'est accessible, on **rejette**.
  - Si l'arbre d'exécution est infini mais qu'il ne contient aucune configuration acceptante, $\mathcal{U}$ ne s'arrête jamais.

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.
  - Si il existe une configuration acceptante accessible depuis la configuration initiale de $\mathcal{M}$ sur w, c'est à dire $q_0\, w$, on **accepte**.
  - Si l'arbre d'exécution est fini, i.e. il n'y a pas d'exécutions infinie, et si aucune configuration acceptante n'est accessible, on **rejette**.
  - Si l'arbre d'exécution est infini mais qu'il ne contient aucune configuration acceptante, $\mathcal{U}$ ne s'arrête jamais.
- On en déduit :

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.
  - Si il existe une configuration acceptante accessible depuis la configuration initiale de $\mathcal{M}$ sur w, c'est à dire $q_0\, w$, on **accepte**.
  - Si l'arbre d'exécution est fini, i.e. il n'y a pas d'exécutions infinie, et si aucune configuration acceptante n'est accessible, on **rejette**.
  - Si l'arbre d'exécution est infini mais qu'il ne contient aucune configuration acceptante, $\mathcal{U}$ ne s'arrête jamais.
- On en déduit :
  - $\mathcal{L}(\mathcal{U}) := \{w \mid q_0\, w \to_{\mathcal{U}}^* C \text{ acceptante}\} = \{[\langle \mathcal{M}, w \rangle]_{code} \mid w \in \mathcal{L}(\mathcal{M})\}$.

Langage accepté par la machine universelle

- Les entrées de la machine $\mathcal{U}$ doivent être de la forme $[\langle \mathcal{M}, w \rangle]_{code}$.
  - Si on propose une entrée mal formattée, i.e. un mot u tel que $\forall \mathcal{M}, \forall w, u \neq [\langle \mathcal{M}, w \rangle]_{code}$, $\mathcal{U}$ rejette son entrée en temps fini.
  - Sinon, on passe à la simulation de $\mathcal{M}$.
- $\mathcal{U}$ explore l'arbre d'exécution de $\mathcal{M}$ sur l'entrée w.
  - Si il existe une configuration acceptante accessible depuis la configuration initiale de $\mathcal{M}$ sur w, c'est à dire $q_0\, w$, on **accepte**.
  - Si l'arbre d'exécution est fini, i.e. il n'y a pas d'exécutions infinie, et si aucune configuration acceptante n'est accessible, on **rejette**.
  - Si l'arbre d'exécution est infini mais qu'il ne contient aucune configuration acceptante, $\mathcal{U}$ ne s'arrête jamais.
- On en déduit :
  - $\mathcal{L}(\mathcal{U}) := \{w \mid q_0\, w \to_{\mathcal{U}}^* C \text{ acceptante}\} = \{[\langle \mathcal{M}, w \rangle]_{code} \mid w \in \mathcal{L}(\mathcal{M})\}$.
  - $\mathcal{U}$ s'arrête sur l'entrée $[\langle \mathcal{M}, w \rangle]_{code}$ si et seulement si $\mathcal{M}$ n'admet pas d'exécution infinie sur l'entrée w.

Vers l'indécidabilité
Plan

1. Ensembles dénombrables
2. Les classes **R** et **RE**
3. Machine universelle

4. Le problème de l'arrêt

Problèmes indécidables

☞ Il existe des problèmes impossibles à résoudre avec un ordinateur.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :
  - Je sais que $L_1 \subseteq \Sigma_1^\star$ est indécidable.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :
  - Je sais que $L_1 \subseteq \Sigma_1^\star$ est indécidable.
  - J'ai une fonction calculable $\varphi : \Sigma_1^\star \to \Sigma_2^\star$.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :
  - Je sais que $L_1 \subseteq \Sigma_1^\star$ est indécidable.
  - J'ai une fonction calculable $\varphi : \Sigma_1^\star \to \Sigma_2^\star$.
  - Je sais que $w \in L_1 \Leftrightarrow \varphi(w) \in L_2$.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :
  - Je sais que $L_1 \subseteq \Sigma_1^\star$ est indécidable.
  - J'ai une fonction calculable $\varphi : \Sigma_1^\star \to \Sigma_2^\star$.
  - Je sais que $w \in L_1 \Leftrightarrow \varphi(w) \in L_2$.
  - Alors, si L_2 était décidable, L_1 le serait aussi.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :
  - Je sais que $L_1 \subseteq \Sigma_1^\star$ est indécidable.
  - J'ai une fonction calculable $\varphi : \Sigma_1^\star \to \Sigma_2^\star$.
  - Je sais que $w \in L_1 \Leftrightarrow \varphi(w) \in L_2$.
  - Alors, si L_2 était décidable, L_1 le serait aussi.
  - Comme L_1 ne l'est pas, L_2 ne l'est pas non plus.

Problèmes indécidables

- Il existe des problèmes impossibles à résoudre avec un ordinateur.
- Cela correspond à l'existence de langages indécidables :

définition (Langage Indécidable)

Un langage $L \in \Sigma^\star$ est indécidable si il n'existe pas de machine à décider $\mathcal{M}$ telle que $\mathcal{L}(\mathcal{M}) = L$.

- Il est en général difficile de montrer directement qu'un langage est indécidable.
- En revanche, on peut faire des réductions :
  - Je sais que $L_1 \subseteq \Sigma_1^\star$ est indécidable.
  - J'ai une fonction calculable $\varphi : \Sigma_1^\star \to \Sigma_2^\star$.
  - Je sais que $w \in L_1 \Leftrightarrow \varphi(w) \in L_2$.
  - Alors, si L_2 était décidable, L_1 le serait aussi.
  - Comme L_1 ne l'est pas, L_2 ne l'est pas non plus.
- On va donc commencer par trouver un premier langage indécidable (non pérave).

Langage d'acceptation

☞ La machine universelle que l'on a construit précédemment reconnaît le langage suivant :

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

Langage d'acceptation

- La machine universelle que l'on a construit précédemment reconnaît le langage suivant :

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

- En revanche elle ne décide pas ce langage : si $\mathcal{M}$ diverge sur l'entrée w, alors $\mathcal{U}$ ne s'arrête jamais.

Langage d'acceptation

- La machine universelle que l'on a construit précédemment reconnaît le langage suivant :

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

- En revanche elle ne décide pas ce langage : si $\mathcal{M}$ diverge sur l'entrée w, alors $\mathcal{U}$ ne s'arrête jamais.
- Peut-on faire mieux ? Autrement dit, le langage $L_{\in}$ est-il décidable ?

Première preuve d'indécidabilité

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

On suppose $L_{\in}$ décidable.

☞ Soit $\mathcal{H}$ une machine décidant $L_{\in}$.

Première preuve d'indécidabilité

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

On suppose $L_{\in}$ décidable.

- Soit $\mathcal{H}$ une machine décidant $L_{\in}$.
- On construit la machine $\mathcal{Q}$:

```
entrée: [M]_code
simuler H sur [M, [M]_code]_code
si H accepte : rejeter
si H rejette : accepter
```

Première preuve d'indécidabilité

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

On suppose $L_{\in}$ décidable.

- Soit $\mathcal{H}$ une machine décidant $L_{\in}$.
- On construit la machine $\mathcal{Q}$:

```
entrée: [M]_code
simuler H sur [M, [M]_code]_code
si H accepte : rejeter
si H rejette : accepter
```

- On peut comprendre la machine $\mathcal{Q}$ comme suit :

$$[\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \in \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \notin \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

Première preuve d'indécidabilité

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

On suppose $L_{\in}$ décidable.

- Soit $\mathcal{H}$ une machine décidant $L_{\in}$.
- On construit la machine $\mathcal{Q}$:

```
entrée: [M]_code
simuler H sur [M, [M]_code]_code
si H accepte : rejeter
si H rejette : accepter
```

- On peut comprendre la machine $\mathcal{Q}$ comme suit :

$$[\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \in \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \notin \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

- Que se passe-t'il si on exécute $\mathcal{Q}$ sur son propre code ?

$$[\mathcal{Q}]_{code} \in \mathcal{L}(\mathcal{Q}) \Rightarrow [\mathcal{Q}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{Q}]_{code} \notin \mathcal{L}(\mathcal{Q}) \Rightarrow [\mathcal{Q}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

Première preuve d'indécidabilité

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

On suppose $L_{\in}$ décidable.

- Soit $\mathcal{H}$ une machine décidant $L_{\in}$.
- On construit la machine $\mathcal{Q}$:

```
entrée: [M]_code
simuler H sur [M, [M]_code]_code
si H accepte : rejeter
si H rejette : accepter
```

- On peut comprendre la machine $\mathcal{Q}$ comme suit :

$$[\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \in \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \notin \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

- Que se passe-t'il si on exécute $\mathcal{Q}$ sur son propre code ?

$$[\mathcal{Q}]_{code} \in \mathcal{L}(\mathcal{Q}) \Rightarrow [\mathcal{Q}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{Q}]_{code} \notin \mathcal{L}(\mathcal{Q}) \Rightarrow [\mathcal{Q}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

- C'est une contradiction !

Première preuve d'indécidabilité

$$\mathcal{L}(\mathcal{U}) = L_{\in} := \left\{ [\mathcal{M}, w]_{code} \;\middle|\; w \in \mathcal{L}(\mathcal{M}) \right\}.$$

On suppose $L_{\in}$ décidable.

- Soit $\mathcal{H}$ une machine décidant $L_{\in}$.
- On construit la machine $\mathcal{Q}$:

```
entrée: [M]_code
simuler H sur [M, [M]_code]_code
si H accepte : rejeter
si H rejette : accepter
```

- On peut comprendre la machine $\mathcal{Q}$ comme suit :

$$[\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \in \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{M}]_{code} \notin \mathcal{L}(\mathcal{M}) \Rightarrow \left[\mathcal{M}, [\mathcal{M}]_{code}\right]_{code} \notin \mathcal{L}(\mathcal{H}) \Rightarrow [\mathcal{M}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

- Que se passe-t'il si on exécute $\mathcal{Q}$ sur son propre code ?

$$[\mathcal{Q}]_{code} \in \mathcal{L}(\mathcal{Q}) \Rightarrow [\mathcal{Q}]_{code} \notin \mathcal{L}(\mathcal{Q}).$$
$$[\mathcal{Q}]_{code} \notin \mathcal{L}(\mathcal{Q}) \Rightarrow [\mathcal{Q}]_{code} \in \mathcal{L}(\mathcal{Q}).$$

- C'est une contradiction !
- Donc $L_{\in}$ est indécidable.

Remarques

- $L_{\in}$ est indécidable
- mais $L_{\in} = \mathcal{L}(\mathcal{U})$, donc $L_{\in}$ est reconnaissable !
- On a donc trouvé un langage appartenant à la classe **RE** qui n'appartient pas à la classe **R**, ce qui prouve **R** $\neq$ **RE**.
- $L_{\in}$ est indécidable et reconnaissable : donc, par le théorème de la semaine dernière, le complément de $L_{\in}$ n'est pas reconnaissable.