Calculabilité et NP-Complétude

Partie I : Machines de Turing, décidabilité

Master 2

Avant-propos

Difficultés :

- Abstrait,
- Lien difficile à faire avec les applications (nécessite du recul et un bonne assimilation des notions abstraites),
- Les résultats d'*impossibilité* reposent souvent sur des arguments complexes (comme en mathématiques),
- L'algorithmique non-déterministe.

Ca passe :

- Les idées générales,
- Le fonctionnements (version Théorie des langages) des machines de Turing,
- Le codage SAT.

Le problème est-il facilement résoluble par une machine ?

Le problème est-il facilement résoluble par une machine ?

- Qu'est-ce qu'**un problème** ? (rappels d'algorithmique, lien avec la théorie des langages)

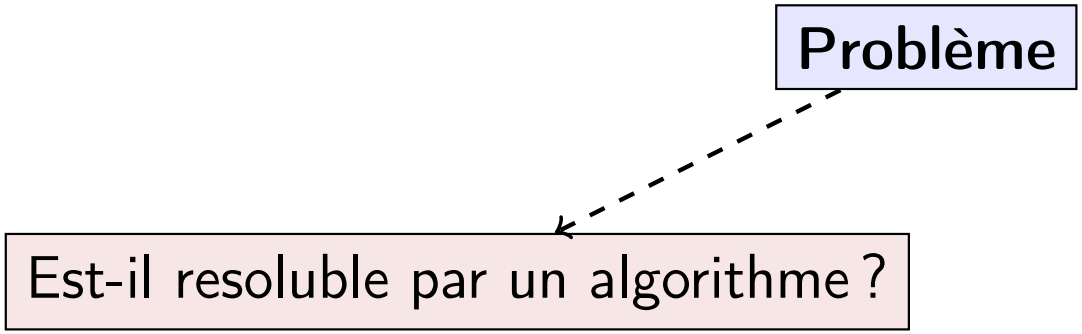
Le problème est-il facilement résoluble par une machine ?

- Qu'est-ce qu'**un problème** ? (rappels d'algorithmique, lien avec la théorie des langages)
- Que veut dire **résoluble par un machine** ? (machines de Turing, décidabilité)

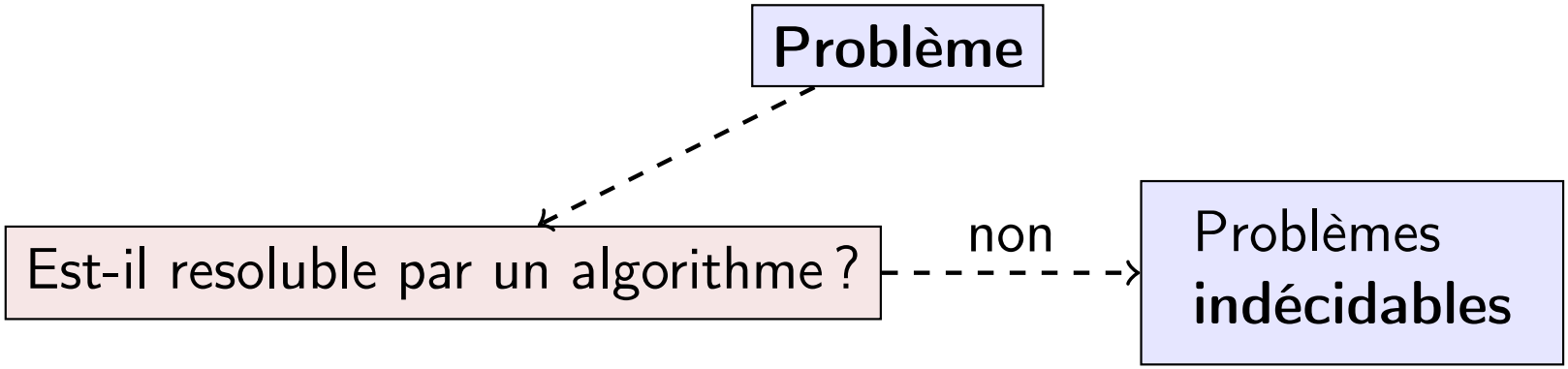
Le problème est-il facilement résoluble par une machine ?

- Qu'est-ce qu'**un problème** ? (rappels d'algorithmique, lien avec la théorie des langages)
- Que veut dire **résoluble par un machine** ? (machines de Turing, décidabilité)
- Que veut dire **facilement** ? (classes de complexité)

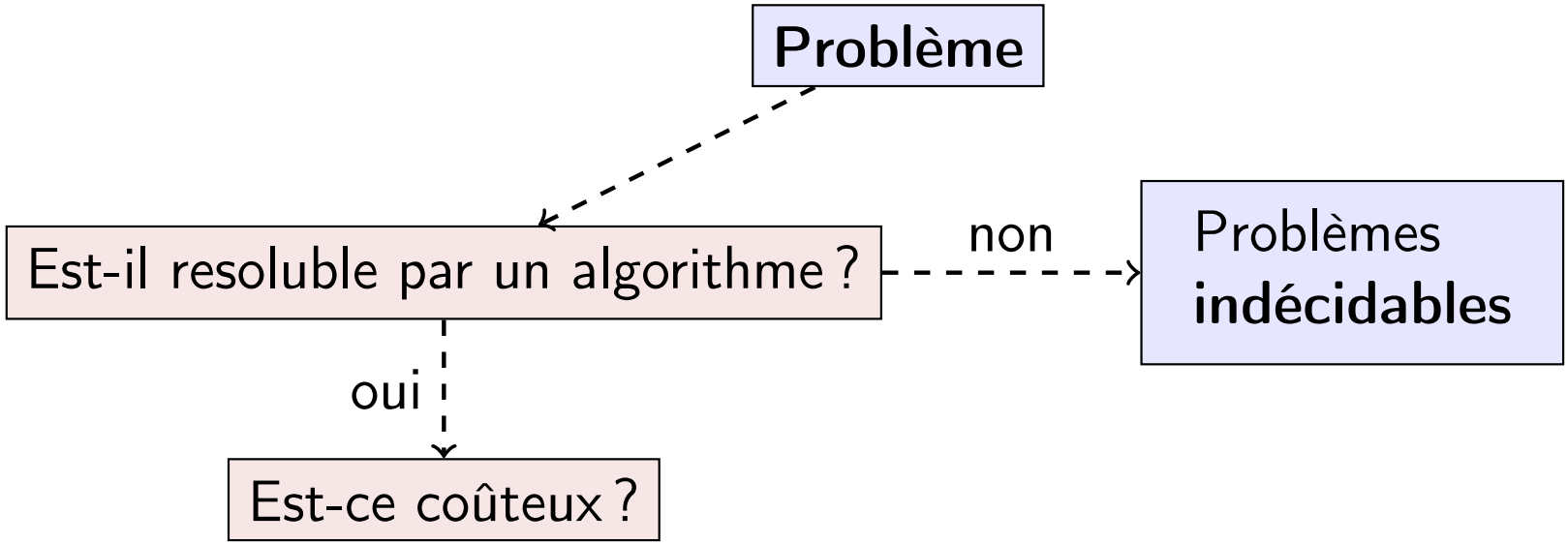
Architechture du cours (schématique)



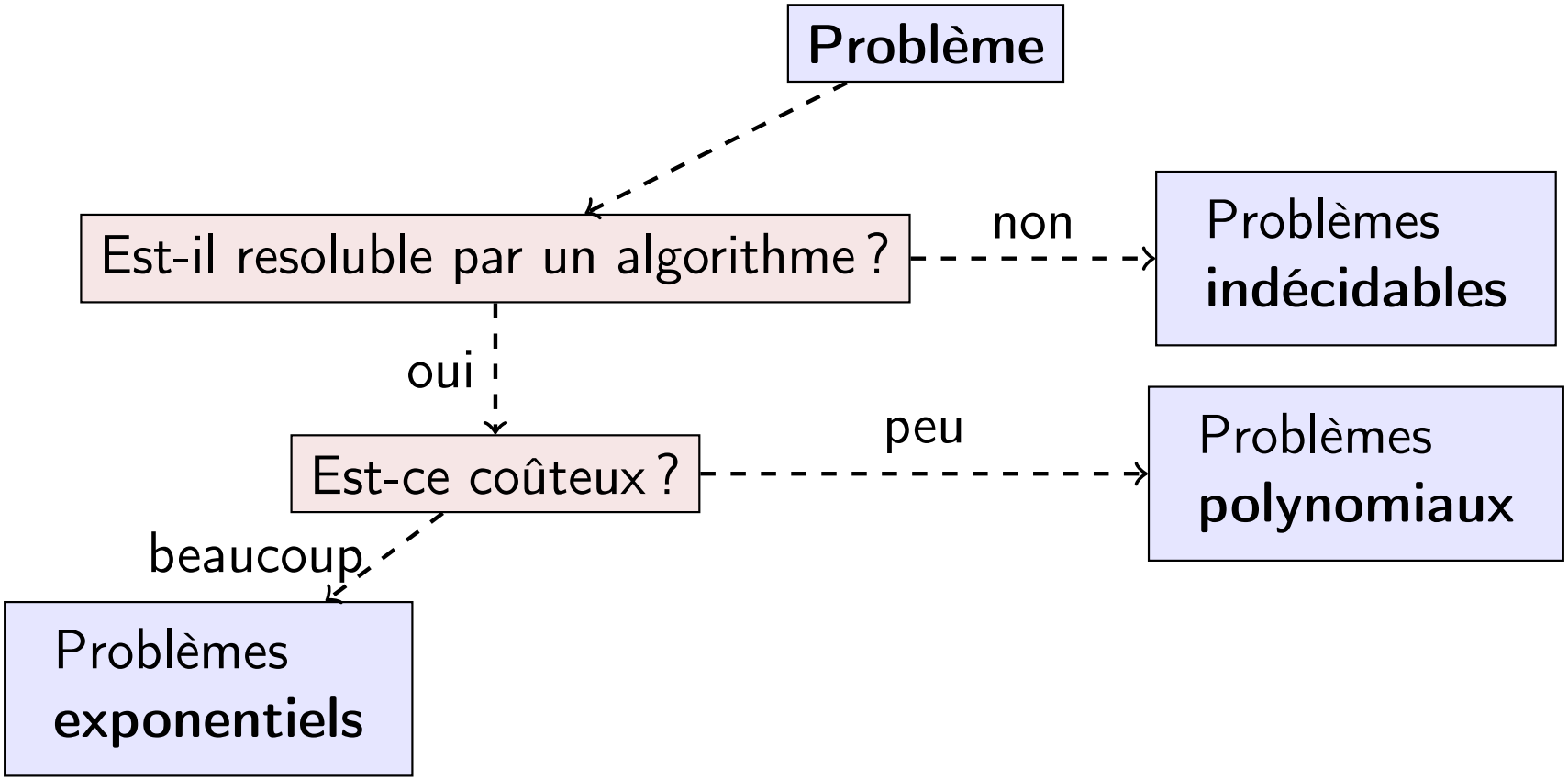
Architechture du cours (schématique)



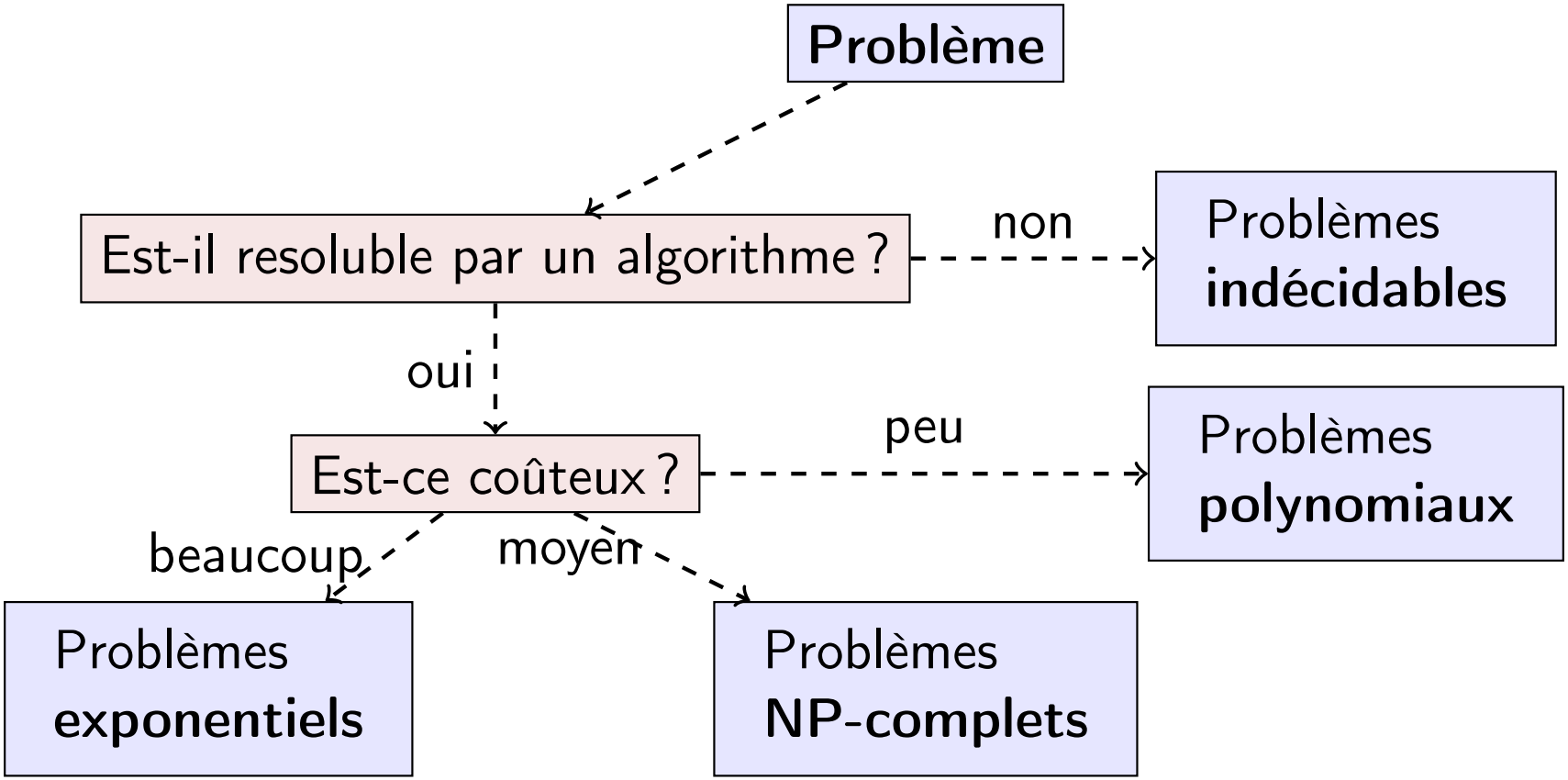
Architechture du cours (schématique)



Architechture du cours (schématique)



Architechture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)

Théorie vs Expérimentation

- Dans la pratique, on veut des langages avec de **nombreuses fonctionnalités**, des types de structures de données évolués, etc.
- Pour faire la théorie, on préfère des **objets simples** (que l'on montre équivalents), mais alors peu ou pas utilisables *en pratique*.

Pour la théorie :

- Les données sont des mots (chaînes/suites de caractères),
- Les programmes sont des machines de Turing.

Calculabilité

Les notions de calculabilité (calcul automatique) ont été abordées bien avant les premiers ordinateurs. La théorie de la calculabilité a été abordée par différentes approches :

- Logique (indirectement Gödel 1929, Tarski 1933),
- Fonctionnelle (lambda Calcul, Chuch 1930),
- Théorie des langages (Machine de Turing, 1936)

Dixième problème de Hilbert (1900) : « On donne une équation de Diophante à un nombre quelconque d'inconnues et à cœfficients entiers rationnels : on demande de trouver une méthode par laquelle, au moyen d'un nombre fini d'opérations, on pourra distinguer si l'équation est résoluble en nombres entiers rationnels. »

Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Langages formels

La théorie des mots et des langages formels :

- base de la calculabilité (via machines à états, hiérarchie de Chomsky),

Langages formels

La théorie des mots et des langages formels :

- base de la calculabilité (via machines à états, hiérarchie de Chomsky),
- algorithmique du texte (recherche de motifs, expressions régulières, codage, compilation, interpretation, etc.),

Langages formels

La théorie des mots et des langages formels :

- base de la calculabilité (via machines à états, hiérarchie de Chomsky),
- algorithmique du texte (recherche de motifs, expressions régulières, codage, compilation, interpretation, etc.),
- modélisation des systèmes,

Langages formels

La théorie des mots et des langages formels :

- base de la calculabilité (via machines à états, hiérarchie de Chomsky),
- algorithmique du texte (recherche de motifs, expressions régulières, codage, compilation, interpretation, etc.),
- modélisation des systèmes,
- mathématiques (théorie des monoïdes).

Langages formels

La théorie des mots et des langages formels :

- base de la calculabilité (via machines à états, hiérarchie de Chomsky),
- algorithmique du texte (recherche de motifs, expressions régulières, codage, compilation, interpretation, etc.),
- modélisation des systèmes,
- mathématiques (théorie des monoïdes).

On s'intéresse ici aux mots finis et aux langages de langages de mots finis. (on ne précisera pas à chaque fois *fini* pour les mots).

Alphabets et lettres

Un alphabet est un ensemble fini dont les éléments sont appelés lettres.
Un alphabet soit souvent noté A ou Σ.

Alphabets et lettres

Un alphabet est un ensemble fini dont les éléments sont appelés lettres.
Un alphabet soit souvent noté A ou Σ. Exemples :

- $\Sigma = \{0, 1\}$ utilisé pour le codage binaire,

Alphabets et lettres

Un alphabet est un ensemble fini dont les éléments sont appelés lettres.
Un alphabet soit souvent noté A ou Σ. Exemples :

- $\Sigma = \{0, 1\}$ utilisé pour le codage binaire,
- $\Sigma = \{0, 1, \ldots, A, B, C, D, E, F\}$ utilisé pour le codage hexadécimal,

Alphabets et lettres

Un alphabet est un ensemble fini dont les éléments sont appelés lettres.
Un alphabet soit souvent noté A ou Σ. Exemples :

- $\Sigma = \{0, 1\}$ utilisé pour le codage binaire,
- $\Sigma = \{0, 1, \ldots, A, B, C, D, E, F\}$ utilisé pour le codage hexadécimal,
- $\Sigma = \{a, \ldots, z\}$ utlisé pour écire en minuscule des mots (sans accents),
- ASCII, utf8, etc.

Alphabets et lettres

Un alphabet est un ensemble fini dont les éléments sont appelés lettres.
Un alphabet soit souvent noté A ou Σ. Exemples :

- $\Sigma = \{0, 1\}$ utilisé pour le codage binaire,
- $\Sigma = \{0, 1, \ldots, A, B, C, D, E, F\}$ utilisé pour le codage hexadécimal,
- $\Sigma = \{a, \ldots, z\}$ utlisé pour écire en minuscule des mots (sans accents),
- ASCII, utf8, etc.
- $\Sigma = \{a, b, c\}$.

Mots

Un mot sur un alphabet Σ est une suite finie de lettres de cet alphabet.

Mots

Un mot sur un alphabet Σ est une suite finie de lettres de cet alphabet.

On écrit de façon naturelle les différentes valeurs (qui sont des lettres) de cette suite finie en les juxtapposant dans l'ordre.

Mots

Un mot sur un alphabet Σ est une suite finie de lettres de cet alphabet.

On écrit de façon naturelle les différentes valeurs (qui sont des lettres) de cette suite finie en les juxtapposant dans l'ordre.

Le mot $u = abbabc$ sur l'alphabet $\Sigma = \{a, b, c, d\}$, est la suite dont

- La première valeur (lettre) $u(1)$ est a.

Mots

Un mot sur un alphabet Σ est une suite finie de lettres de cet alphabet.

On écrit de façon naturelle les différentes valeurs (qui sont des lettres) de cette suite finie en les juxtapposant dans l'ordre.

Le mot $u = abbabc$ sur l'alphabet $\Sigma = \{a, b, c, d\}$, est la suite dont

- La première valeur (lettre) $u(1)$ est a.
- La seconde valeur (lettre) $u(2)$ est b.

Mots

Un mot sur un alphabet Σ est une suite finie de lettres de cet alphabet.

On écrit de façon naturelle les différentes valeurs (qui sont des lettres) de cette suite finie en les juxtapposant dans l'ordre.

Le mot $u = abbabc$ sur l'alphabet $\Sigma = \{a, b, c, d\}$, est la suite dont

- La première valeur (lettre) $u(1)$ est a.
- La seconde valeur (lettre) $u(2)$ est b.
- La troisième valeur (lettre) $u(3)$ est b.
- etc.

Mots

Un mot sur un alphabet Σ est une suite finie de lettres de cet alphabet.

On écrit de façon naturelle les différentes valeurs (qui sont des lettres) de cette suite finie en les juxtapposant dans l'ordre.

Le mot $u = abbabc$ sur l'alphabet $\Sigma = \{a, b, c, d\}$, est la suite dont

- La première valeur (lettre) $u(1)$ est a.
- La seconde valeur (lettre) $u(2)$ est b.
- La troisième valeur (lettre) $u(3)$ est b.
- etc.

On peut utiliser plusieurs fois la même lettre et ne pas en utiliser certaines.

Longueur d'un mot

La longueur d'un mot est le nombre de lettres qu'il contient (comptées avec répétition).

Longueur d'un mot

La longueur d'un mot est le nombre de lettres qu'il contient (comptées avec répétition).

Le mot *abbacb* a pour longueur 6, alors que le mot *bba* a pour longueur 3. La longueur d'un mot u est noté $|u|$ (comme la valeur absolue).

Longueur d'un mot

La longueur d'un mot est le nombre de lettres qu'il contient (comptées avec répétition).

Le mot *abbacb* a pour longueur 6, alors que le mot *bba* a pour longueur 3.
La longueur d'un mot u est noté $|u|$ (comme la valeur absolue).
On a $|abbabadcb| = 9$.

Longueur d'un mot

La longueur d'un mot est le nombre de lettres qu'il contient (comptées avec répétition).

Le mot *abbacb* a pour longueur 6, alors que le mot *bba* a pour longueur 3.
La longueur d'un mot u est noté $|u|$ (comme la valeur absolue).
On a $|abbabadcb| = 9$.

Il existe un unique mot de longueur 0 (qui ne contient aucune lettre), qu'on appelle le mot vide et que l'on note ε.
On a $|\varepsilon| = 0$.

Produit de mots

Le produit de deux mots sur un alphabet Σ est le mot obtenu en juxtapposant les deux mots.

Le produit de $u = abb$ par $v = bcba$ est le mot $abbbcba$ noté uv.

Produit de mots

Le produit de deux mots sur un alphabet Σ est le mot obtenu en juxtapposant les deux mots.

Le produit de $u = abb$ par $v = bcba$ est le mot $abbbcba$ noté uv.

- **Non-commutatif** : en général $uv \neq vu$.

Produit de mots

Le produit de deux mots sur un alphabet Σ est le mot obtenu en juxtapposant les deux mots.

Le produit de $u = abb$ par $v = bcba$ est le mot $abbbcba$ noté uv.

- **Non-commutatif** : en général $uv \neq vu$.
- **Associatif** : $u(vw) = (uv)w = uvw$.

Produit de mots

Le produit de deux mots sur un alphabet Σ est le mot obtenu en juxtapposant les deux mots.

Le produit de $u = abb$ par $v = bcba$ est le mot $abbbcba$ noté uv.

- **Non-commutatif** : en général $uv \neq vu$.
- **Associatif** : $u(vw) = (uv)w = uvw$.
- **Elément neutre** : $\varepsilon u = u\varepsilon = u$.

Produit de mots

Le produit de deux mots sur un alphabet Σ est le mot obtenu en juxtapposant les deux mots.

Le produit de $u = abb$ par $v = bcba$ est le mot $abbbcba$ noté uv.

- **Non-commutatif** : en général $uv \neq vu$.
- **Associatif** : $u(vw) = (uv)w = uvw$.
- **Elément neutre** : $\varepsilon u = u\varepsilon = u$.

Pour tous mots u et v, on a

$$|uv| = |u| + |v|$$

Produit de mots

Le produit de deux mots sur un alphabet Σ est le mot obtenu en juxtapposant les deux mots.

Le produit de $u = abb$ par $v = bcba$ est le mot $abbbcba$ noté uv.

- **Non-commutatif** : en général $uv \neq vu$.
- **Associatif** : $u(vw) = (uv)w = uvw$.
- **Elément neutre** : $\varepsilon u = u\varepsilon = u$.

Pour tous mots u et v, on a

$$|uv| = |u| + |v|$$

Le produit de mots est aussi appelé concaténation ou produit de concaténation.

Langages (formels)

Un langage sur un alphabet Σ est un ensemble de mots sur Σ.

Langages (formels)

Un langage sur un alphabet Σ est un ensemble de mots sur Σ.
Par exemple sur $\Sigma = \{a, b, c\}$:

- $\{ab, aac, ba\}$.

Langages (formels)

Un langage sur un alphabet Σ est un ensemble de mots sur Σ.
Par exemple sur $\Sigma = \{a, b, c\}$:

- $\{ab, aac, ba\}$.
- L'ensemble des mots de longueur paire.

Langages (formels)

Un langage sur un alphabet Σ est un ensemble de mots sur Σ.
Par exemple sur $\Sigma = \{a, b, c\}$:

- $\{ab, aac, ba\}$.
- L'ensemble des mots de longueur paire.
- L'ensemble des mots ayant autant de a que de b.

Langages (formels)

Un langage sur un alphabet Σ est un ensemble de mots sur Σ.
Par exemple sur $\Sigma = \{a, b, c\}$:

- $\{ab, aac, ba\}$.
- L'ensemble des mots de longueur paire.
- L'ensemble des mots ayant autant de a que de b.
- $\varnothing$.
- etc.

Langages (formels)

Un langage sur un alphabet Σ est un ensemble de mots sur Σ.
Par exemple sur $\Sigma = \{a, b, c\}$:

- $\{ab, aac, ba\}$.
- L'ensemble des mots de longueur paire.
- L'ensemble des mots ayant autant de a que de b.
- $\varnothing$.
- etc.

L'ensemble de tous les mots sur un alphabet Σ est noté Σ^*.

Ici $*$ ne veut pas dire *privé de zéro* comme usuellement lorsqu'on écrit $\mathbb{N}^*$ ou $\mathbb{R}^*$.

Opérations sur les langages

Les langages sont des sous-ensembles, on peut donc faire les opérations classique union, intersection, complément, etc.

Produit

$$LK = \{uv \mid u \in L \text{ et } v \in K\}$$

Etoile

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup \dots$$

$\{ab, ba, aa, bb\}^*$: ensemble des mots de mots de longueur paire sur $\{a, b\}$.

Exercices

Exercice 1

Calculer LK où $L = \{aaba, bb\}$ et $K = \{ab, ba\}$.

Exercice 2

- Donner deux langages L et K finis tels que $|LK| < |L|.|K|$.
- Donner deux langages L, K tels que $(L \cap K)^* \neq L^* \cap K^*$.

Exercices

Exercice 1

Calculer LK où $L = \{aaba, bb\}$ et $K = \{ab, ba\}$.

$\{aabaab, aababa, bbab, bbba\}$

Exercice 2

- Donner deux langages L et K finis tels que $|LK| < |L|.|K|$.
- Donner deux langages L, K tels que $(L \cap K)^* \neq L^* \cap K^*$.

Exercices

Exercice 1

Calculer LK où $L = \{aaba, bb\}$ et $K = \{ab, ba\}$.

Exercice 2

- Donner deux langages L et K finis tels que $|LK| < |L|.|K|$.
- Donner deux langages L, K tels que $(L \cap K)^* \neq L^* \cap K^*$.

- $L = \{a, aa\}$, $K = \{b, ab\}$, on a $LK = \{ab, aab, aaab\}$.
- $L = \{a\}$, $K = \{aa\}$; $(L \cap K)^* = \{\varepsilon\}$ alors que $aa \in L^* \cap K^*$.

Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Rappels sur les problèmes

Un problème est constitué de données (sous une certaines forme) et d'une question ou d'une sortie dépendant ces données.

Rappels sur les problèmes

Un problème est constitué de données (sous une certaines forme) et d'une question ou d'une sortie dépendant ces données.

Donnée(s) : Un entier n en binaire
Sortie : `Vrai` si n est premier, `Faux` sinon ?

Rappels sur les problèmes

Un problème est constitué de données (sous une certaines forme) et d'une question ou d'une sortie dépendant ces données.

Donnée(s) : Un entier n en binaire
Sortie : `Vrai` si n est premier, `Faux` sinon ?

Donnée(s) : Un entier n en binaire strictement supérieur à 1
Question : Quel est le plus petit diviseur de n strictement supérieur à 1 ?

Rappels sur les problèmes

Un problème est constitué de données (sous une certaines forme) et d'une question ou d'une sortie dépendant ces données.

Donnée(s) : Un entier n en binaire
Sortie : `Vrai` si n est premier, `Faux` sinon ?

Donnée(s) : Un entier n en binaire strictement supérieur à 1
Question : Quel est le plus petit diviseur de n strictement supérieur à 1 ?

Donnée(s) : Un entier n supérieur à 1 par sa décomposition en facteurs premiers
Question : Quel est le plus petit diviseur de n strictement supérieur à 1 ?

Rappels sur les problèmes

Un problème est constitué de données (sous une certaines forme) et d'une question ou d'une sortie dépendant ces données.

Donnée(s) : Un entier n en binaire
Sortie : `Vrai` si n est premier, `Faux` sinon ?

Donnée(s) : Un entier n en binaire strictement supérieur à 1
Question : Quel est le plus petit diviseur de n strictement supérieur à 1 ?

Donnée(s) : Un entier n supérieur à 1 par sa décomposition en facteurs premiers
Question : Quel est le plus petit diviseur de n strictement supérieur à 1 ?

Un problème de décision est un problème dont la réponse/sortie est oui-non ou vrai-faux. On n'étudiera la calculabilité que pour les problèmes de décision.

Problèmes de décision et codage des entrées

On se donne un alphabet A. On dit qu'un problème est codé sur A si toute donnée est un mot sur A.

Problèmes de décision et codage des entrées

On se donne un alphabet A. On dit qu'un problème est codé sur A si toute donnée est un mot sur A.

Donnée(s) : Un entier n en binaire
Question : n est-il premier ?
est un problème codé sur $\{0, 1\}$.

Problèmes de décision et codage des entrées

On se donne un alphabet A. On dit qu'un problème est codé sur A si toute donnée est un mot sur A.

Donnée(s) : Un entier n en binaire
Question : n est-il premier ?
est un problème codé sur $\{0, 1\}$.

Donnée(s) : Deux entiers n_1 et n_2 en binaire
Question : n_1 est-il plus grand que n_2
est un problème codé sur $\{0, 1, \#\}$, avec la convention que le couple d'entiers est donné sous la forme $n_1 \# n_2$.

Problèmes de décision et codage des entrées

On se donne un alphabet A. On dit qu'un problème est codé sur A si toute donnée est un mot sur A.

Donnée(s) : Un entier n en binaire
Question : n est-il premier ?
est un problème codé sur $\{0, 1\}$.

Donnée(s) : Deux entiers n_1 et n_2 en binaire
Question : n_1 est-il plus grand que n_2
est un problème codé sur $\{0, 1, \#\}$, avec la convention que le couple d'entiers est donné sous la forme $n_1 \# n_2$.

Coder linéairement (par un mot) les données n'est pas restrictif dans le cadre général de la calculabilité (explications plus tard).

Problèmes et langages

Soit un alphabet A.

A chaque langage L sur A, on peut associer le problème de décision sur A suivant :
Donnée : w un mot sur A
Question : est-ce $w \in L$?

Problèmes et langages

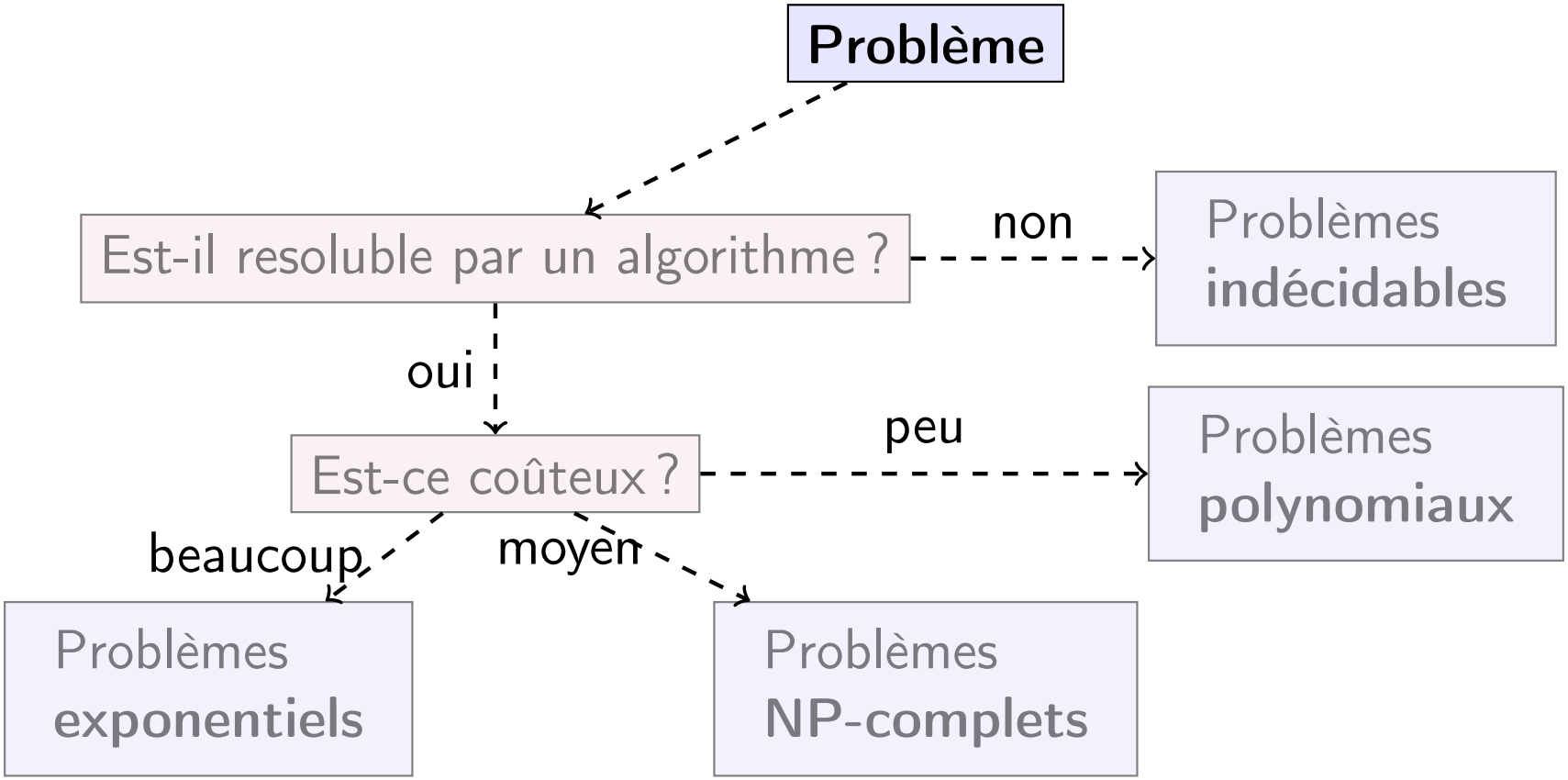
Soit un alphabet A.

A chaque langage L sur A, on peut associer le problème de décision sur A suivant :
Donnée : w un mot sur A
Question : est-ce $w \in L$?

Reciproquement, à chaque problème de décision P sur A, on peut associer le langage des mots sur A pour lesquels la réponse à la question est oui/vrai.

Etudier un problème (de décision sur A) est identique à étudier un langage sur A. La classification des problèmes revient à une classification des langages formels.

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

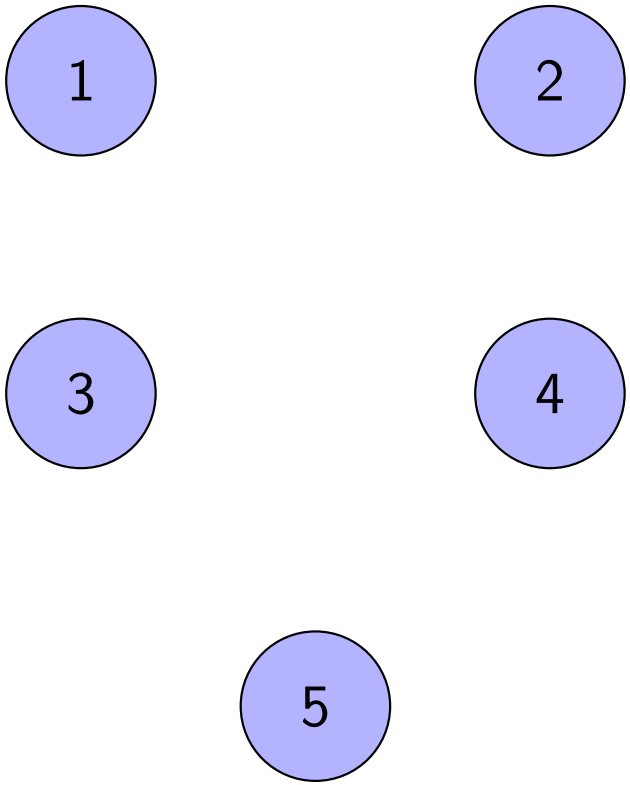
Solveurs IP (VS)

Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

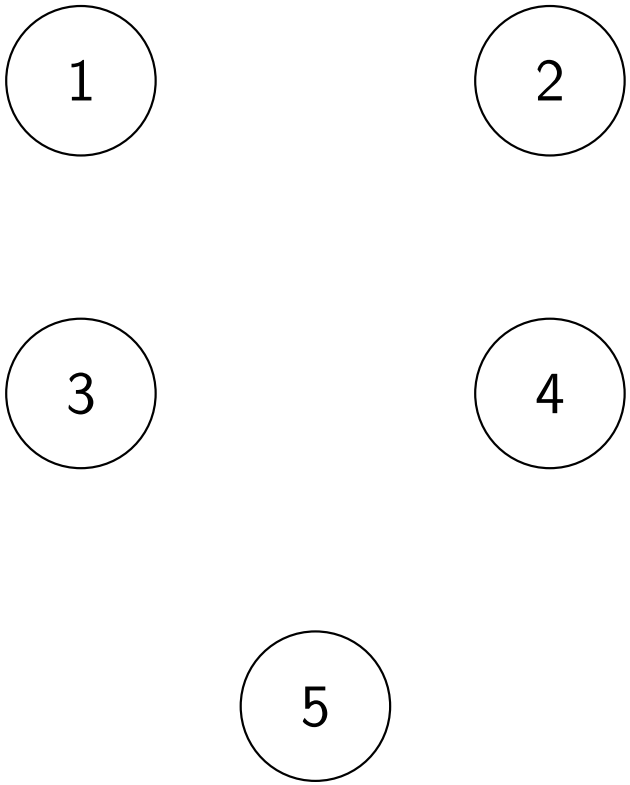
Automates finis

Automates finis



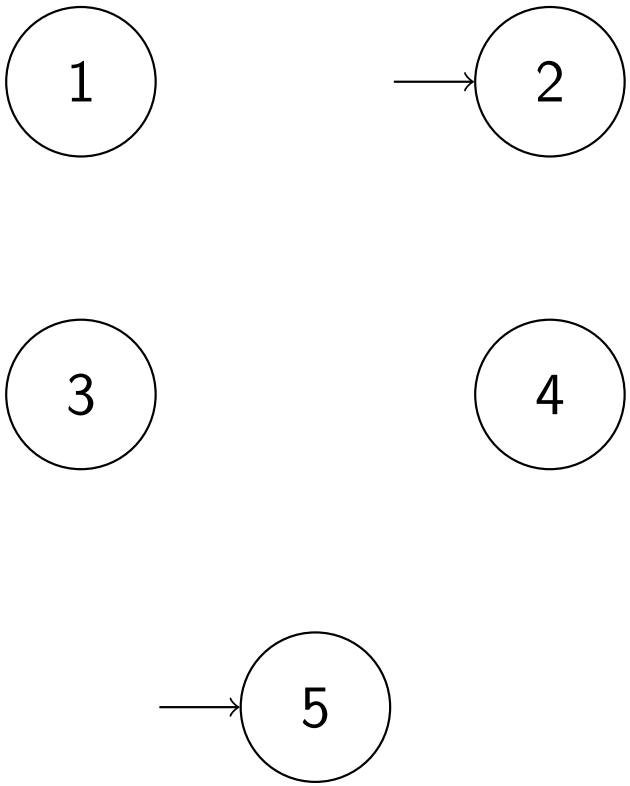
Ens. des états
$Q = \{1, 2, 3, 4, 5\}$

Automates finis



Ens. des états
$Q = \{1, 2, 3, 4, 5\}$

Automates finis



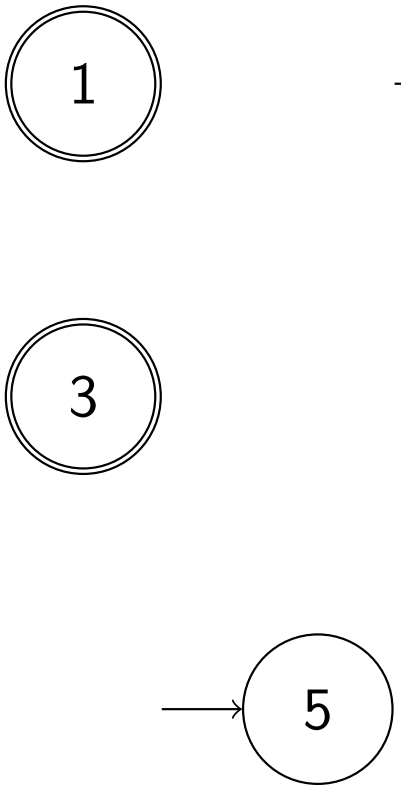
Ens. des états

$Q = \{1, 2, 3, 4, 5\}$

Etats initiaux

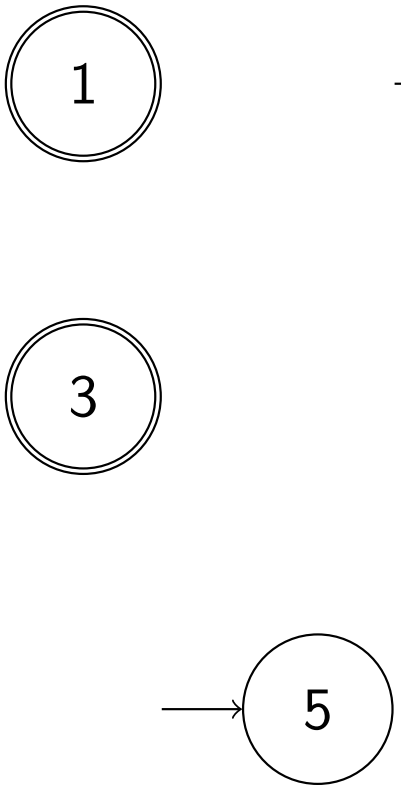
$I = \{2, 5\}$

Automates finis



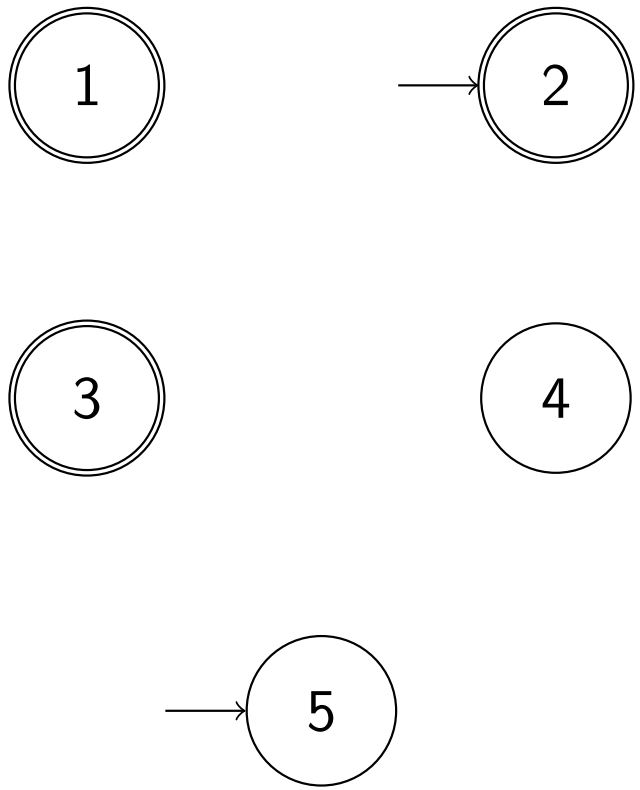
Ens. des états
$Q = \{1, 2, 3, 4, 5\}$
Etats initiaux
$I = \{2, 5\}$
Etats finaux
$F = \{1, 2, 3\}$

Automates finis



Ens. des états
$Q = \{1, 2, 3, 4, 5\}$
Etats initiaux
$I = \{2, 5\}$
Etats finaux
$F = \{1, 2, 3\}$

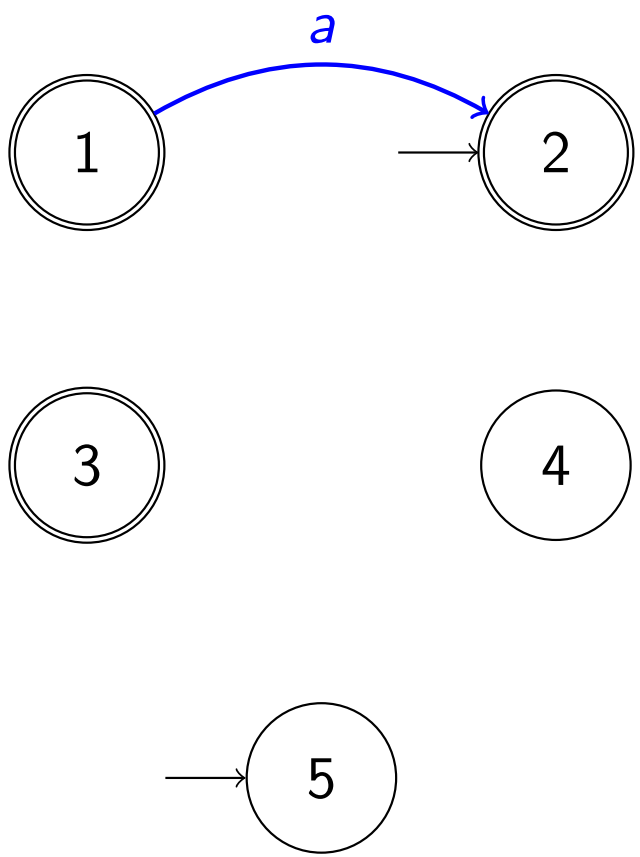
Automates finis



Ens. des états
$Q = \{1, 2, 3, 4, 5\}$
Etats initiaux
$I = \{2, 5\}$
Etats finaux
$F = \{1, 2, 3\}$

Ens. des transitions
$\Delta = \{$
$(1, a, 2), (1, a, 3), (1, b, 2),$
$(2, b, 4), (4, a, 4),$
$(5, a, 4), (5, b, 3)\}$

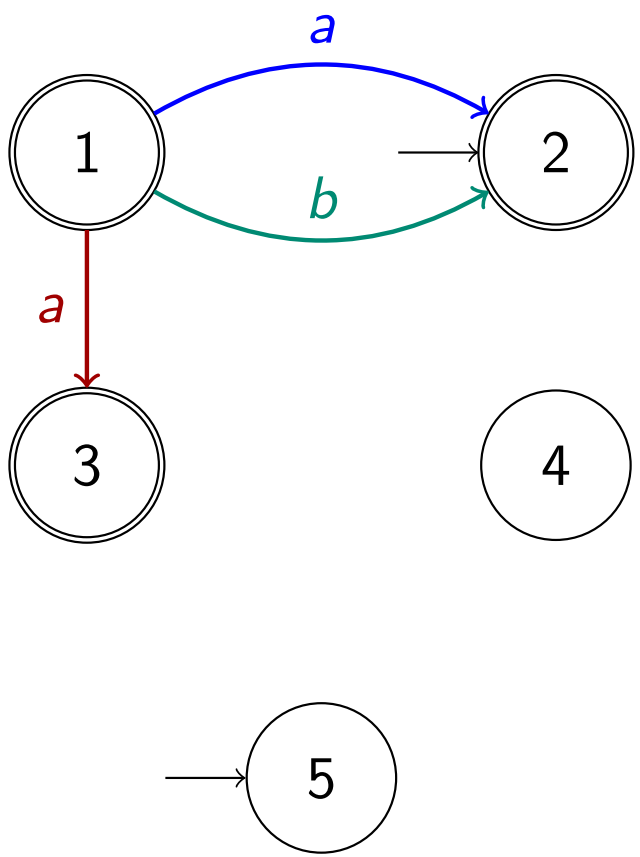
Automates finis



Ens. des états
$Q = \{1, 2, 3, 4, 5\}$
Etats initiaux
$I = \{2, 5\}$
Etats finaux
$F = \{1, 2, 3\}$

Ens. des transitions
$\Delta = \{$
$(1, a, 2), (1, a, 3), (1, b, 2),$
$(2, b, 4), (4, a, 4),$
$(5, a, 4), (5, b, 3)\}$

Automates finis



Ens. des états

$Q = \{1, 2, 3, 4, 5\}$

Etats initiaux

$I = \{2, 5\}$

Etats finaux

$F = \{1, 2, 3\}$

Ens. des transitions

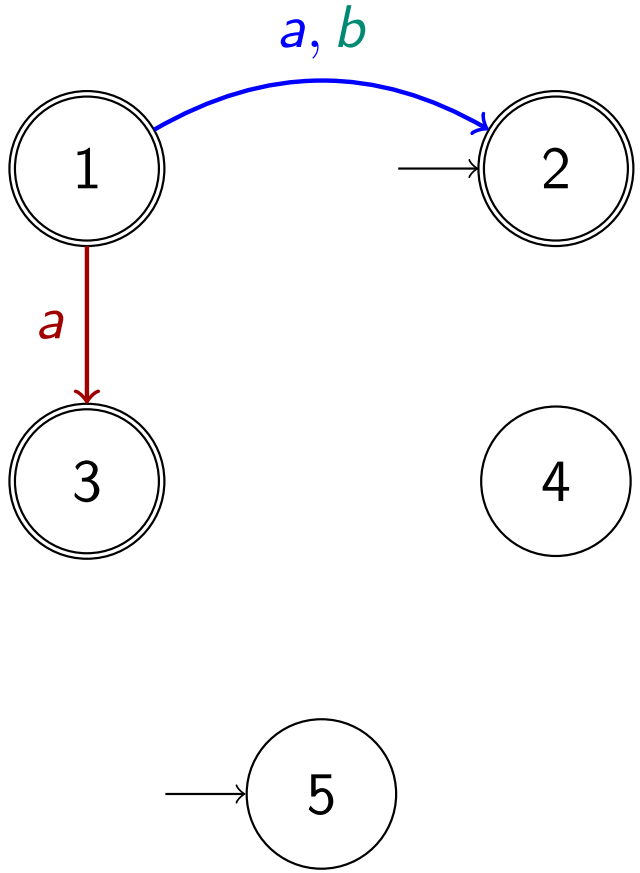
$\Delta = \{$

$(1, a, 2), (1, a, 3), (1, b, 2),$

$(2, b, 4), (4, a, 4),$

$(5, a, 4), (5, b, 3)\}$

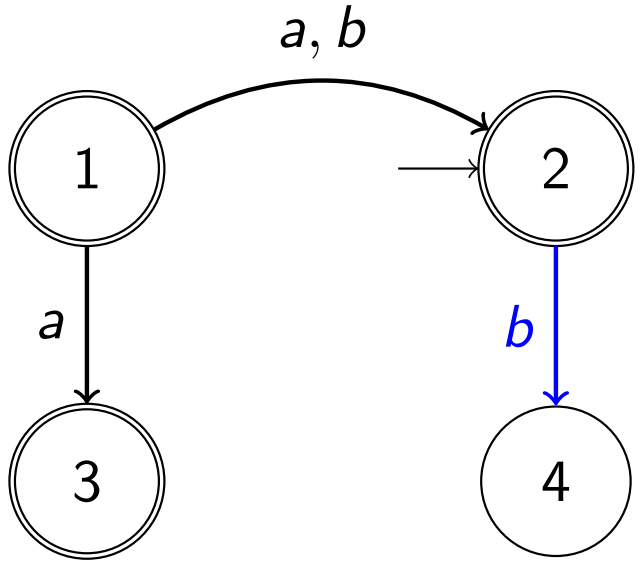
Automates finis

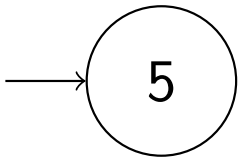


Ens. des états
$Q = \{1, 2, 3, 4, 5\}$
Etats initiaux
$I = \{2, 5\}$
Etats finaux
$F = \{1, 2, 3\}$

Ens. des transitions
$\Delta = \{$
$(1, a, 2), (1, a, 3), (1, b, 2),$
$(2, b, 4), (4, a, 4),$
$(5, a, 4), (5, b, 3)\}$

Automates finis





Ens. des états

$Q = \{1, 2, 3, 4, 5\}$

Etats initiaux

$I = \{2, 5\}$

Etats finaux

$F = \{1, 2, 3\}$

Ens. des transitions

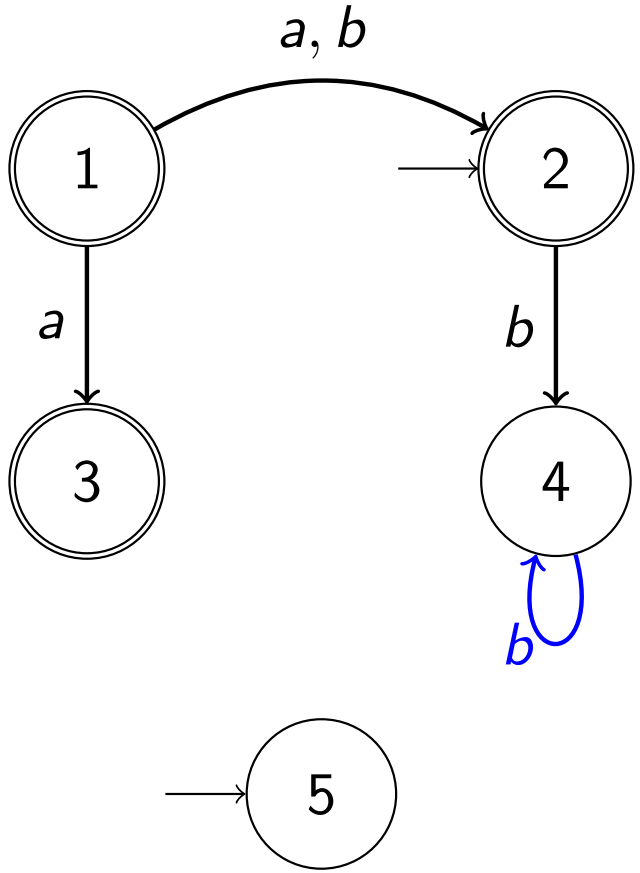
$\Delta = \{$

$(1, a, 2), (1, a, 3), (1, b, 2),$

$(2, b, 4), (4, b, 4),$

$(5, a, 4), (5, b, 3)\}$

Automates finis



Ens. des états

$Q = \{1, 2, 3, 4, 5\}$

Etats initiaux

$I = \{2, 5\}$

Etats finaux

$F = \{1, 2, 3\}$

Ens. des transitions

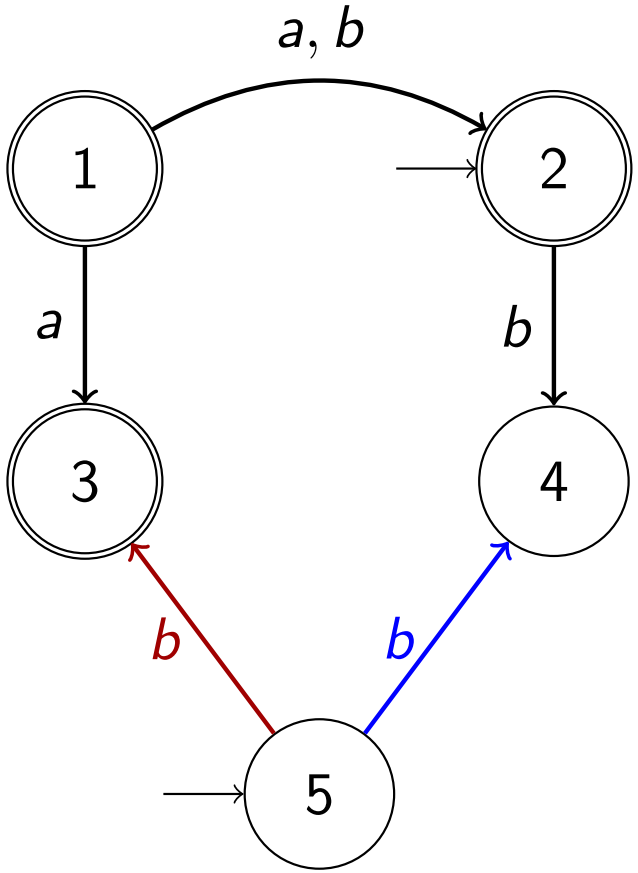
$\Delta = \{$

$(1, a, 2), (1, a, 3), (1, b, 2),$

$(2, b, 4), (4, b, 4),$

$(5, a, 4), (5, b, 3)\}$

Automates finis



Ens. des états

$Q = \{1, 2, 3, 4, 5\}$

Etats initiaux

$I = \{2, 5\}$

Etats finaux

$F = \{1, 2, 3\}$

Ens. des transitions

$\Delta = \{$

$(1, a, 2), (1, a, 3), (1, b, 2),$

$(2, b, 4), (4, b, 4),$

$(5, a, 4), (5, b, 3)\}$

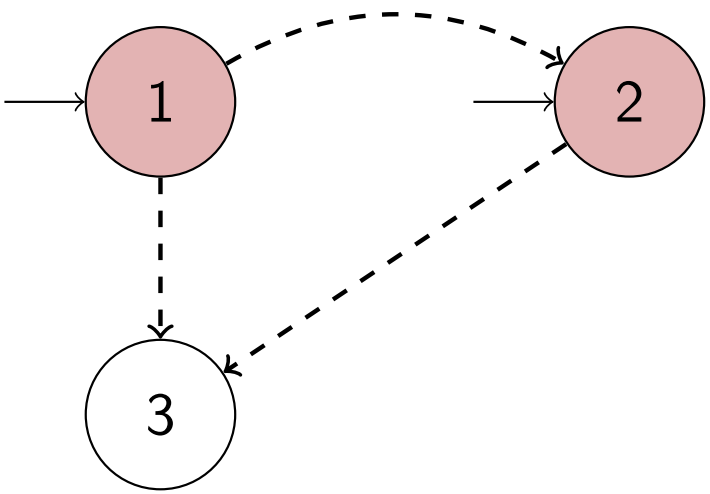
Automates finis déterministes

Un automate fini est déterministe s'il vérifie les deux conditions suivantes :

Automates finis déterministes

Un automate fini est déterministe s'il vérifie les deux conditions suivantes :

- Il possède exactement un état initial,

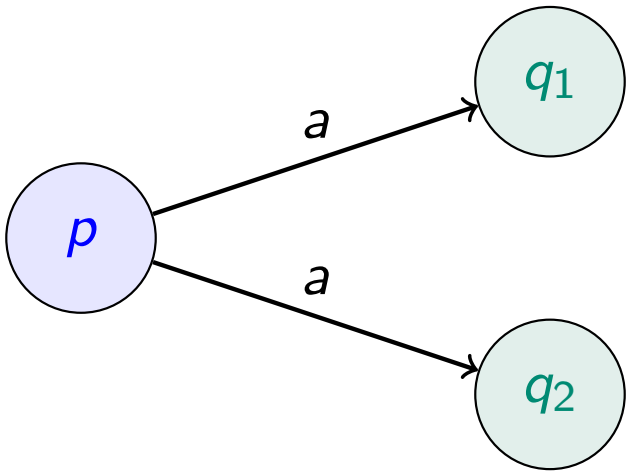


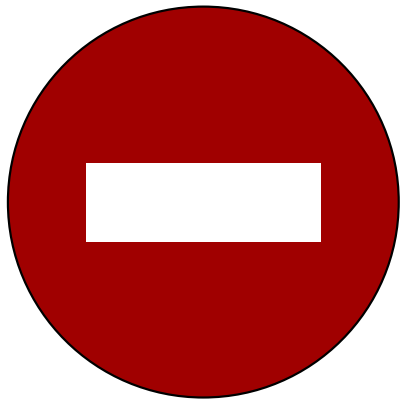
Il y a deux états initiaux 1 et 2, l'automate **ne** peut **pas** être déterministe.

Automates finis déterministes

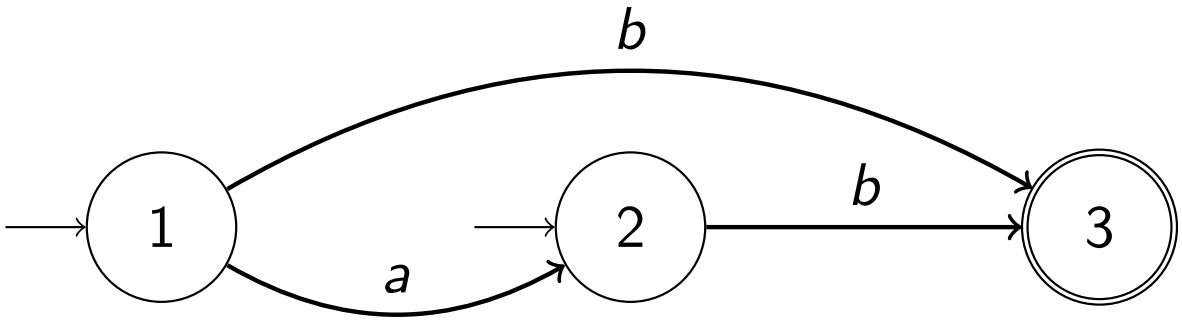
Un automate fini est déterministe s'il vérifie les deux conditions suivantes :

- Il possède exactement un état initial,
- Pour tout état p, toute lettre a, il existe au plus un état q tel que (p, a, q) soit une transition.



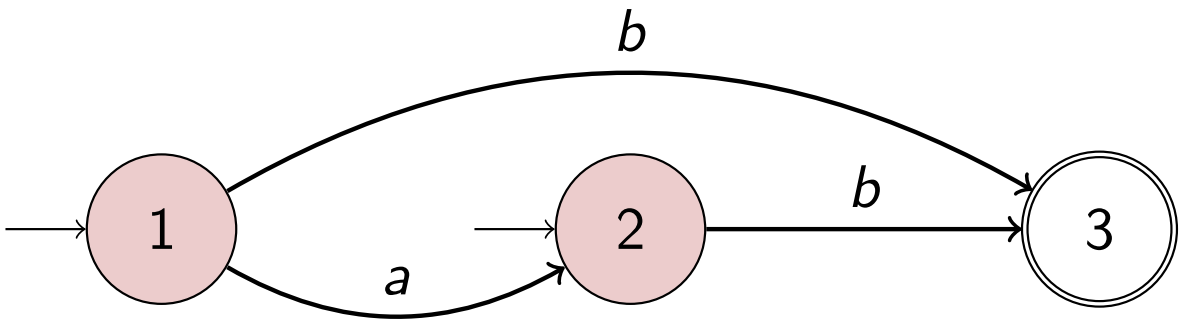


Automates finis déterministes, exemple 1



Cet automate est-il déterministe ?

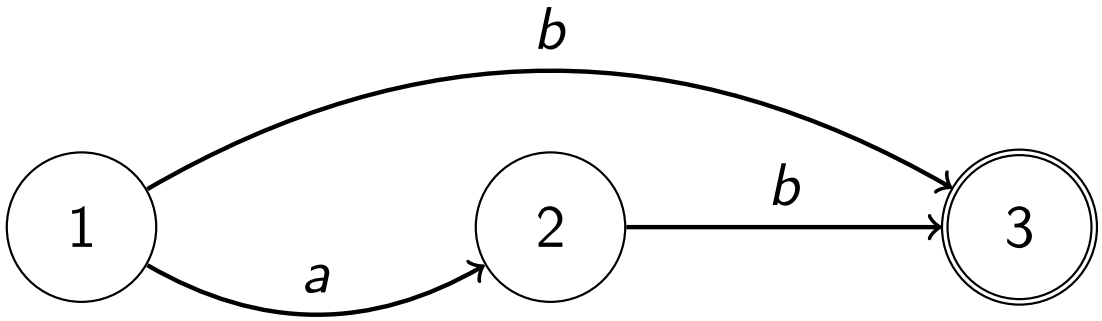
Automates finis déterministes, exemple 1



Cet automate est-il déterministe ?

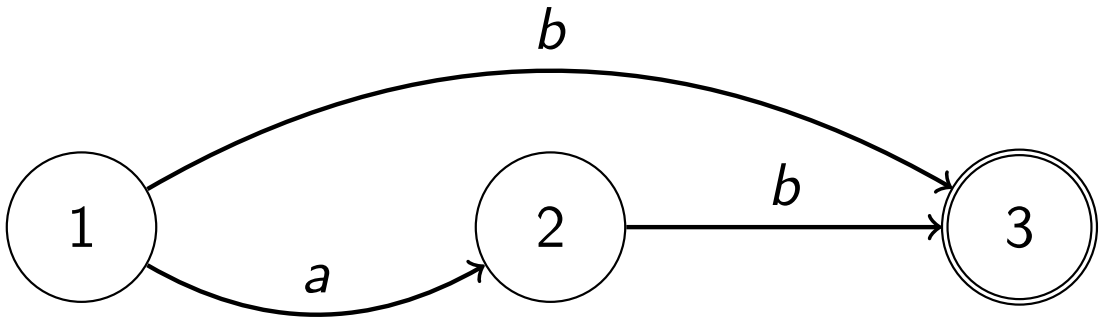
Non, car il ne possède pas exactement un état initial (il en a deux).

Automates finis déterministes, exemple 2



Cet automate est-il déterministe ?

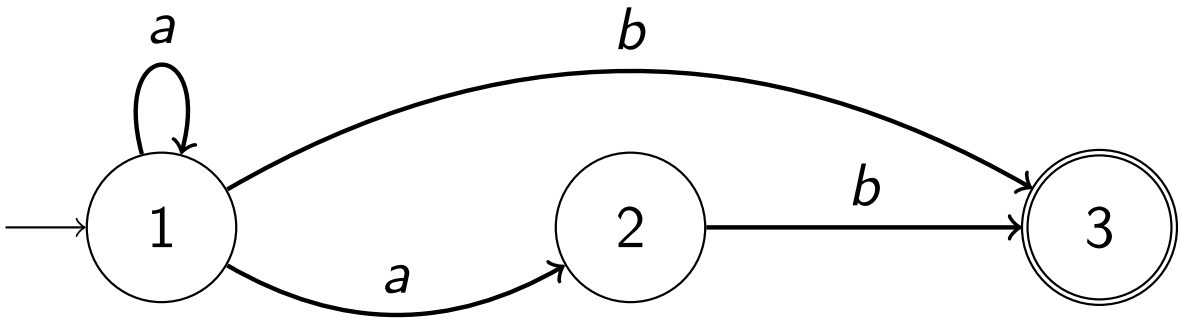
Automates finis déterministes, exemple 2



Cet automate est-il déterministe ?

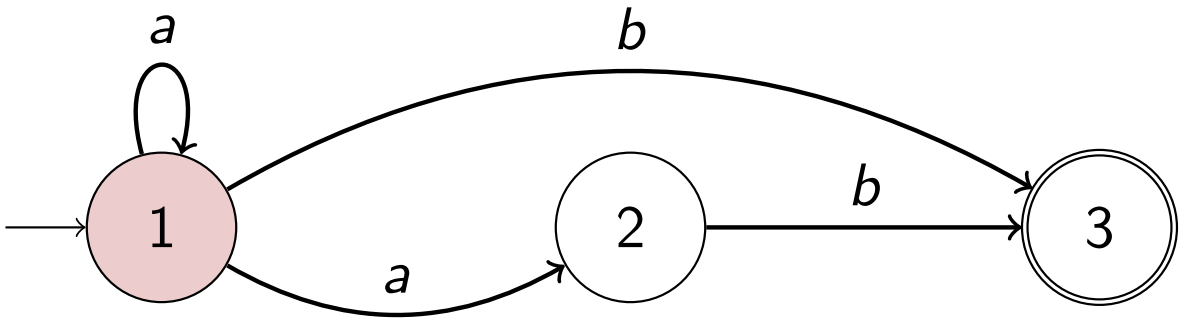
Non, car il ne possède pas exactement un état initial (il n'en a aucun).

Automates finis déterministes, exemple 3



Cet automate est-il déterministe ?

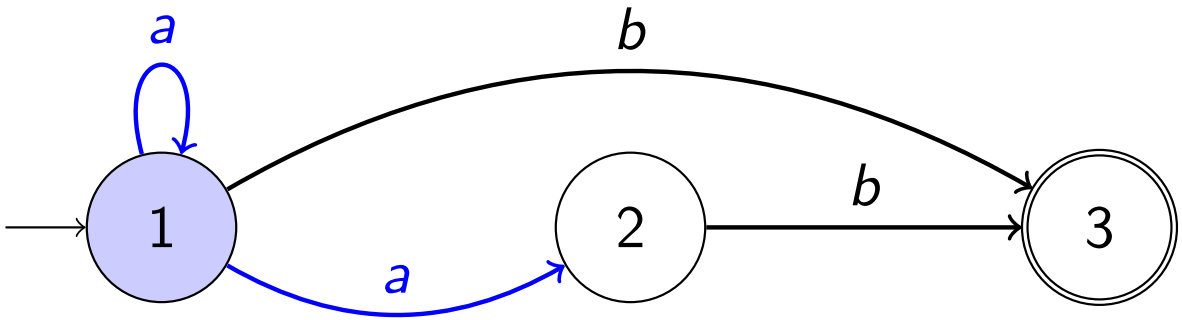
Automates finis déterministes, exemple 3



Cet automate est-il déterministe ?

Il a un unique état initial : 1.

Automates finis déterministes, exemple 3

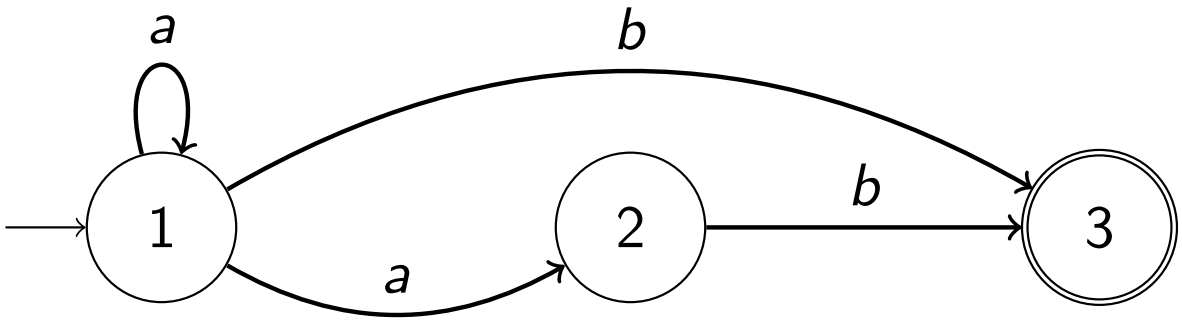


Cet automate est-il déterministe ?

Il a un unique état initial : 1.

Mais il a deux transitions etiquetées par *a* partant de 1.

Automates finis déterministes, exemple 3



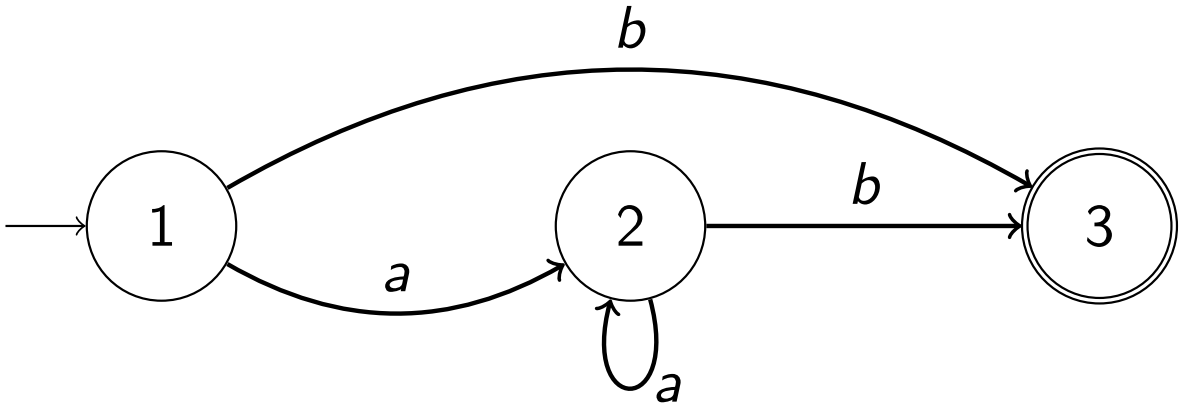
Cet automate est-il déterministe ?

Il a un unique état initial : 1.

Mais il a deux transitions etiquetées par *a* partant de 1.

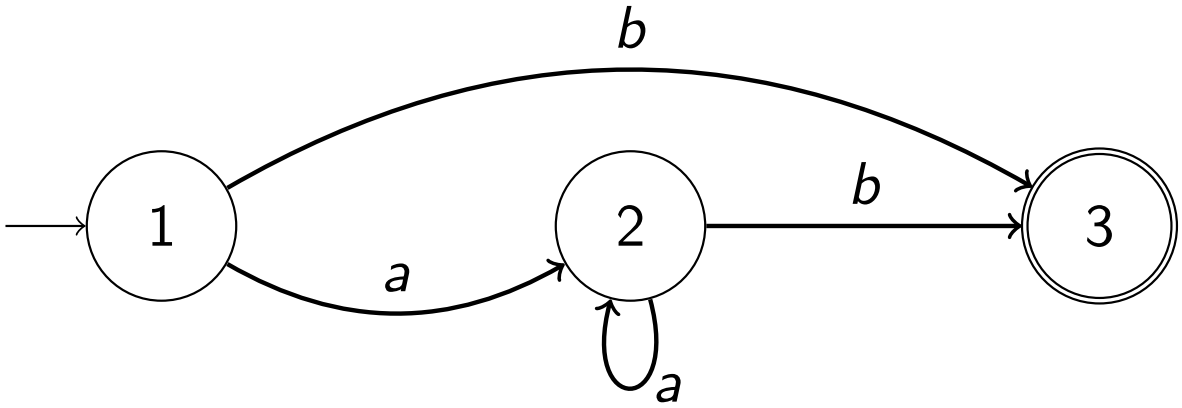
L'automate n'est pas déterminsite.

Automates finis déterministes, exemple 4



Cet automate est-il déterministe ?

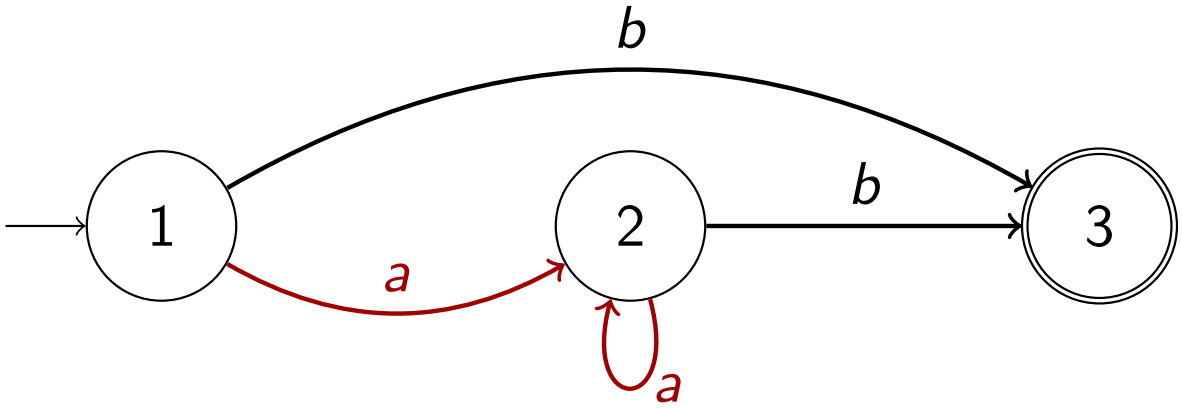
Automates finis déterministes, exemple 4



Cet automate est-il déterministe ?

Il n'a qu'un seul état initial.

Automates finis déterministes, exemple 4

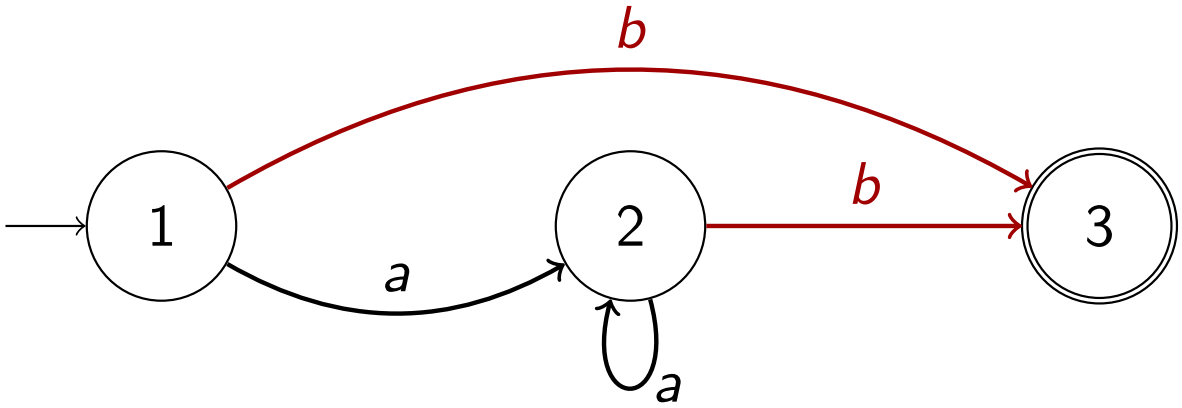


Cet automate est-il déterministe ?

Il n'a qu'un seul état initial.

Il n'a jamais plusieurs transitions partant d'un même état et toutes étiquetées par a.

Automates finis déterministes, exemple 4



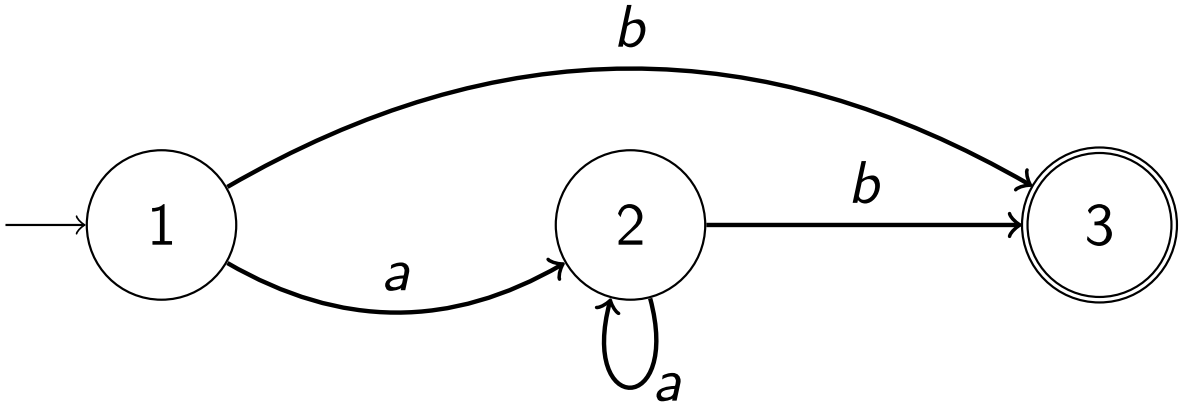
Cet automate est-il déterministe ?

Il n'a qu'un seul état initial.

Il n'a jamais plusieurs transitions partant d'un même état et toutes étiquetées par *a*.

Il n'a jamais plusieurs transitions partant d'un même état et toutes étiquetées par *b*.

Automates finis déterministes, exemple 4



Cet automate est-il déterministe ?

Il n'a qu'un seul état initial.

Il n'a jamais plusieurs transitions partant d'un même état et toutes étiquetées par *a*.

Il n'a jamais plusieurs transitions partant d'un même état et toutes étiquetées par *b*.

L'automate est déterministe.

Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Chemin (acceptant/réussi)

Dans un automate, un chemin (suite de transitions consécutives) est acceptant si

- Il part d'un état initial,
- Il termine dans un état final,

Un mot est accepté par un automate fini s'il existe un chemin réussi l'ayant pour étiquette.

Chemin (acceptant/réussi)

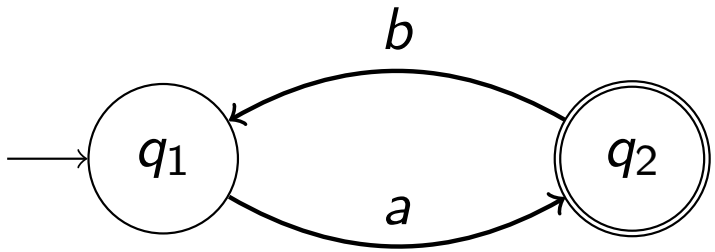
Dans un automate, un chemin (suite de transitions consécutives) est acceptant si

- Il part d'un état initial,
- Il termine dans un état final,

Un mot est accepté par un automate fini s'il existe un chemin réussi l'ayant pour étiquette.

Le langage reconnu par un automate $\mathcal{A}$, noté $L(\mathcal{A})$, est l'ensemble des mots reconnus.

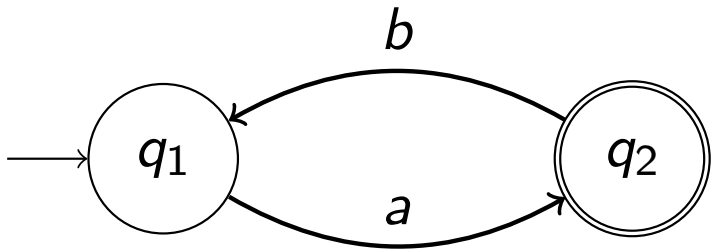
Automates déterministes



Le mot *ababa* est reconnu par l'automate :

$$(q_1, a, q_2)(q_2, b, q_1)(q_1, a, q_2)(q_2, b, q_1)(q_1, a, q_2)$$

Automates déterministes

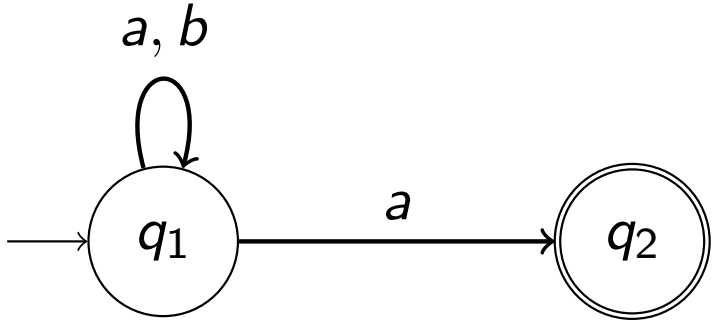


Le mot *ababa* est reconnu par l'automate :

$$(q_1, a, q_2)(q_2, b, q_1)(q_1, a, q_2)(q_2, b, q_1)(q_1, a, q_2)$$

Cet automate reconnaît $\{ab\}^*a$.

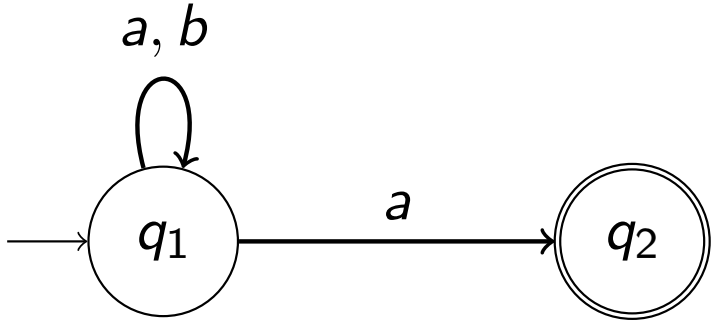
Automates non déterministes



Le mot *aaa* est reconnu par l'automate :

$$(q_1, a, q_1)(q_1, a, q_1)(q_1, a, q_2)$$

Automates non déterministes

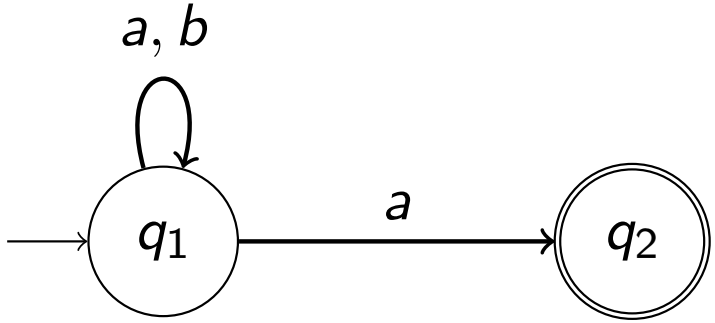


Le mot *aaa* est reconnu par l'automate :

$$(q_1, a, q_1)(q_1, a, q_1)(q_1, a, q_2)$$

Cet automate reconnaît l'ensemble des mots terminant par *a*.

Automates non déterministes



Le mot *aaa* est reconnu par l'automate :

$$(q_1, a, q_1)(q_1, a, q_1)(q_1, a, q_2)$$

Cet automate reconnaît l'ensemble des mots terminant par *a*.

Attention à la définition existentielle de reconnaissance.

Théorème de Kleen

Un langage est reconnu par un automate fini ssi il est exprimable par une expression régulière.

Exercices

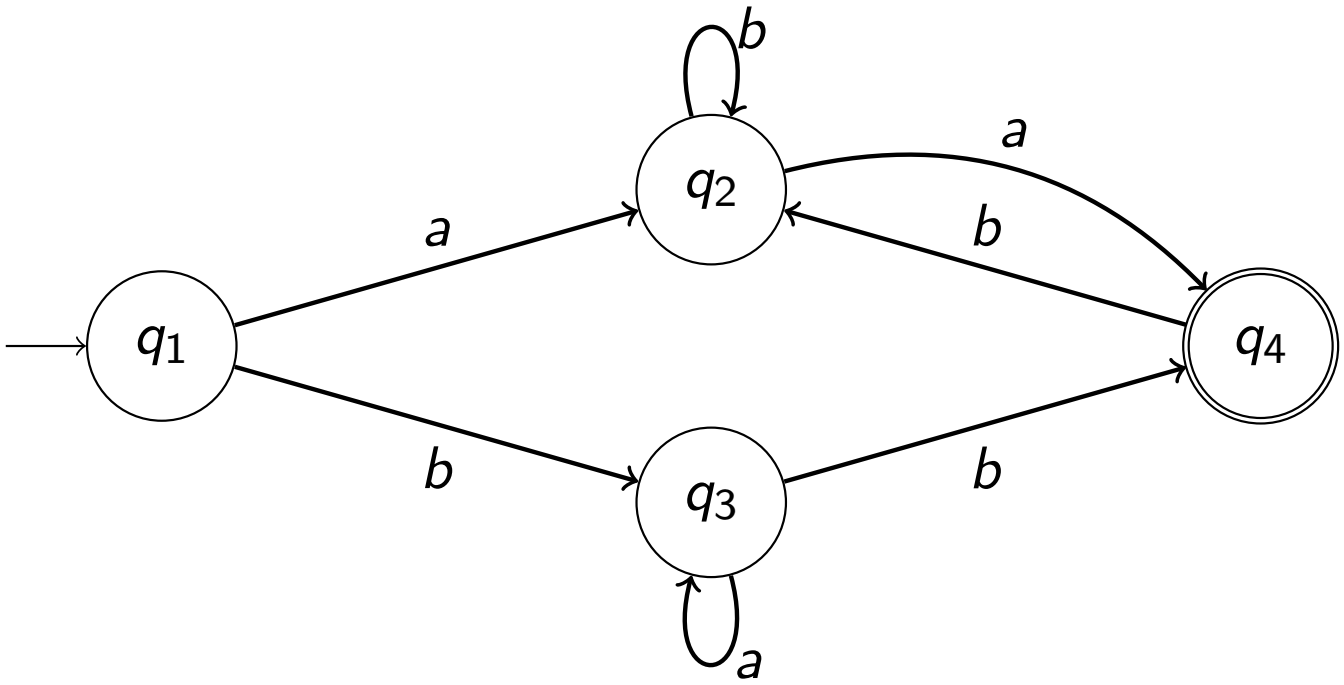
Pour chacun des langages suivant sur l'alphabet $\{a, b, c\}$, donner une expression régulière de ce langage ainsi qu'un automate fini le reconnaissant.

1. Mots finissant par a,
2. Mots commençant par a,
3. Mots finissant et commençant par a,
4. Mots de longeur paire,
5. Mots où tout a est suivi immédiatement d'un b.

Plan

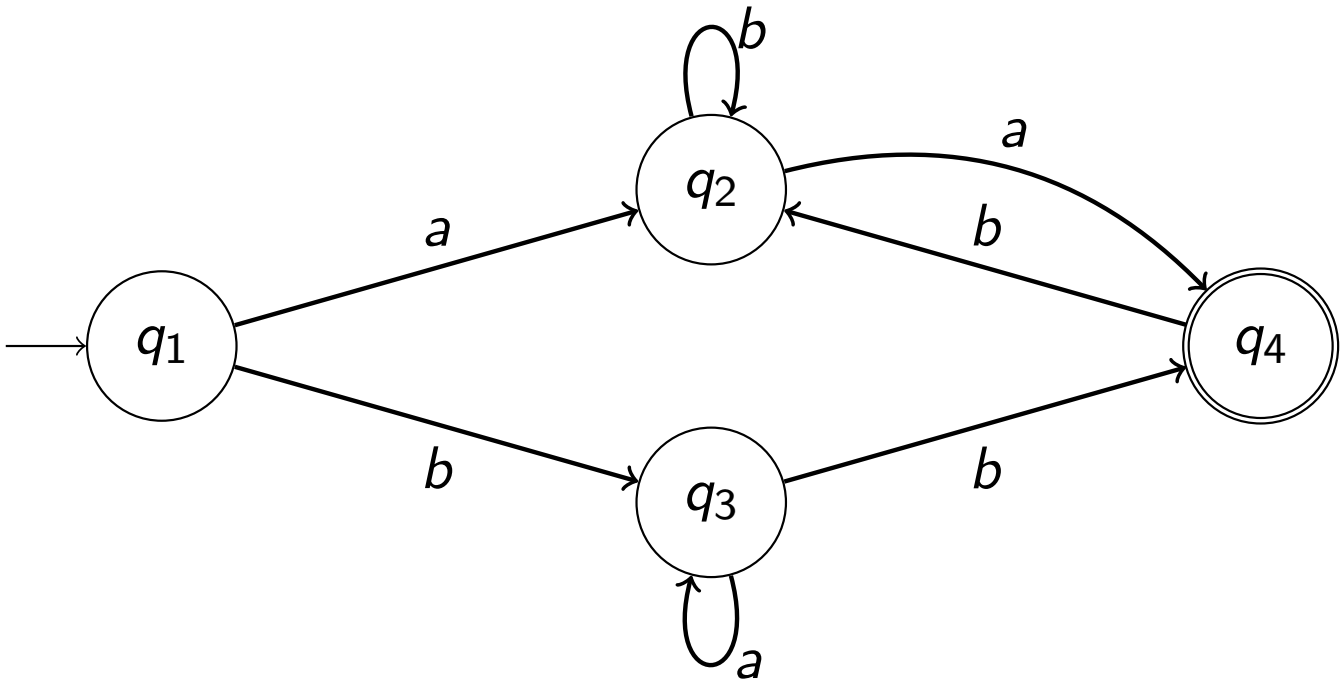
1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Rappel sur les automates déterministes (1)

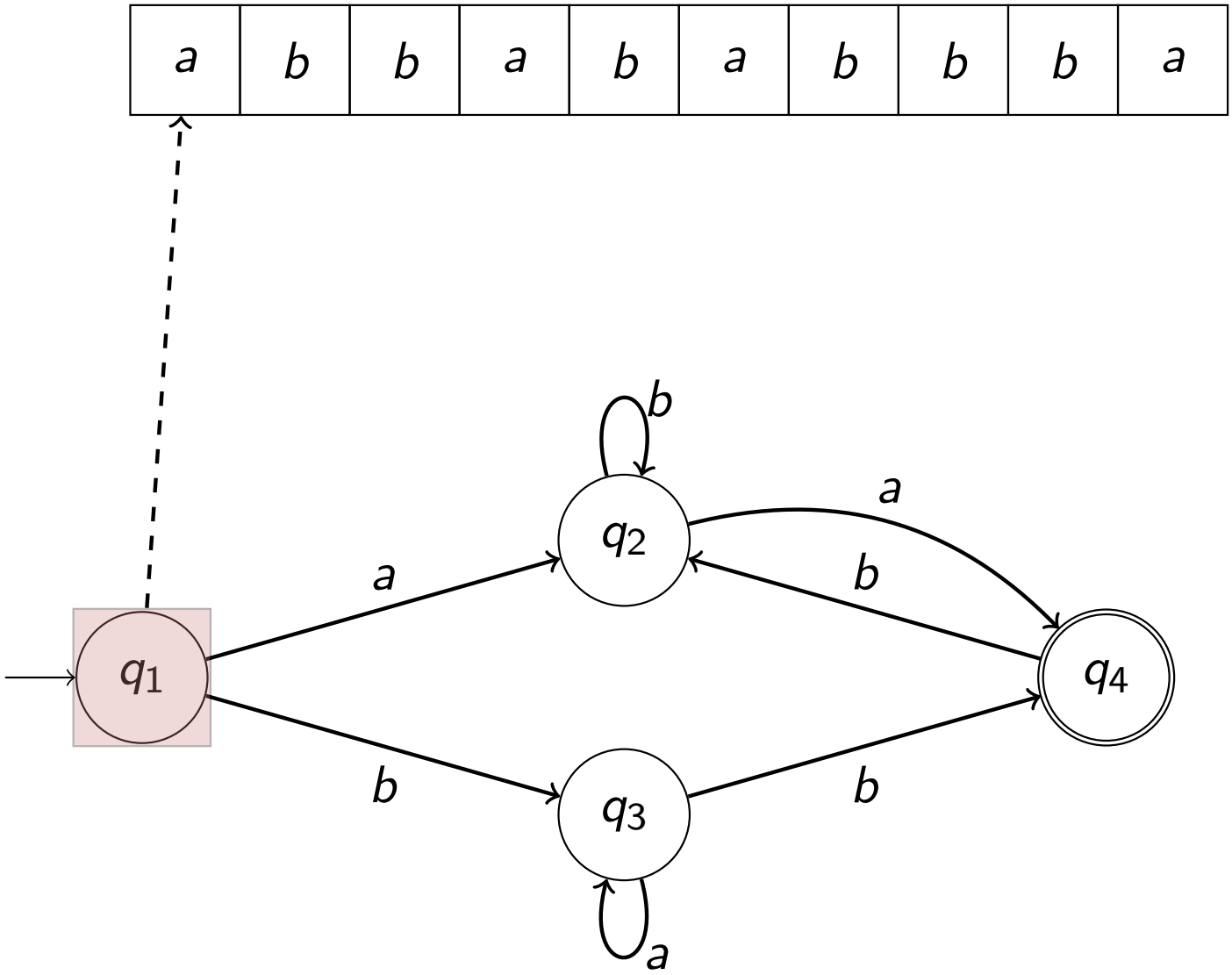


Rappel sur les automates déterministes (1)

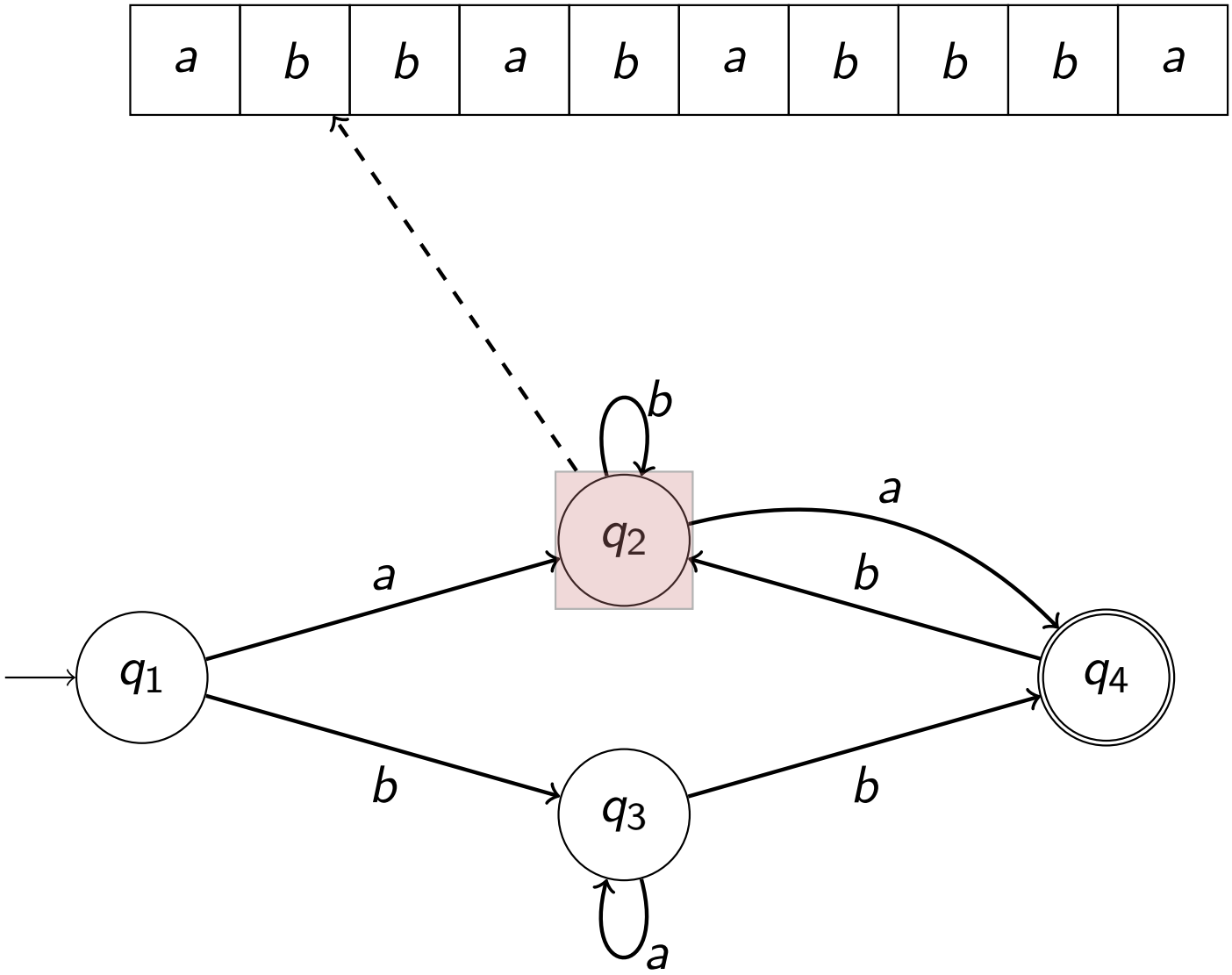
a	b	b	a	b	a	b	b	b	a



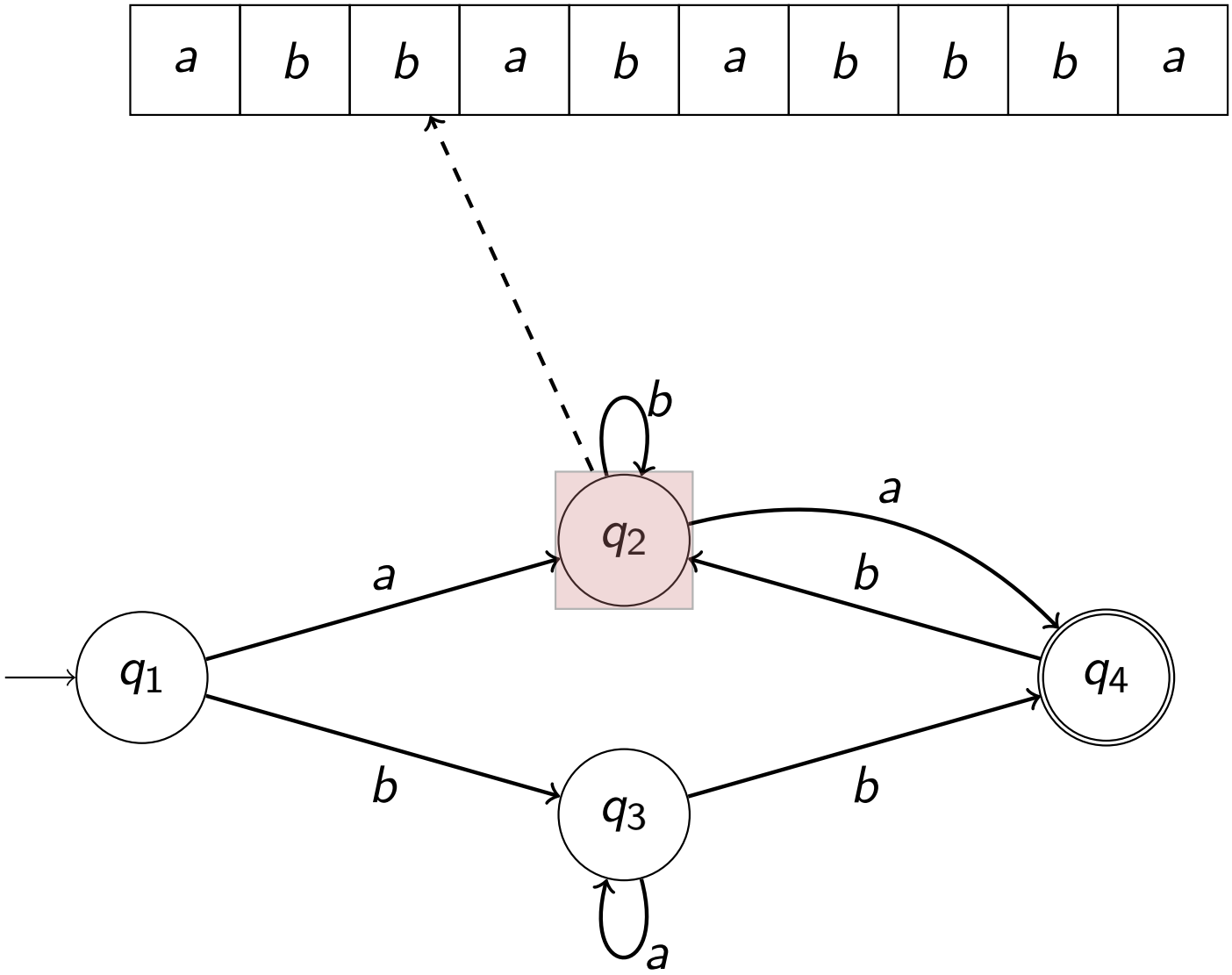
Rappel sur les automates déterministes (1)



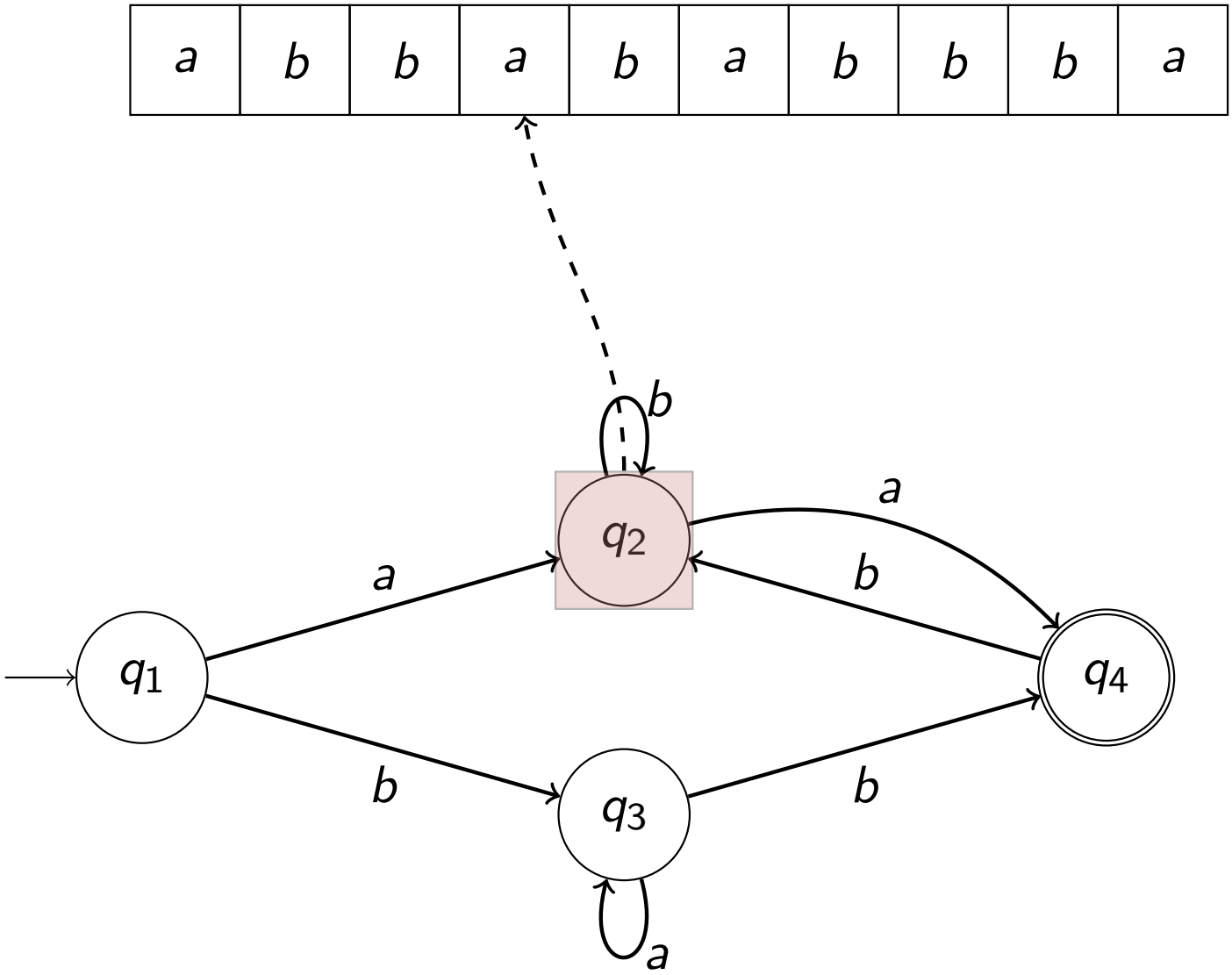
Rappel sur les automates déterministes (1)



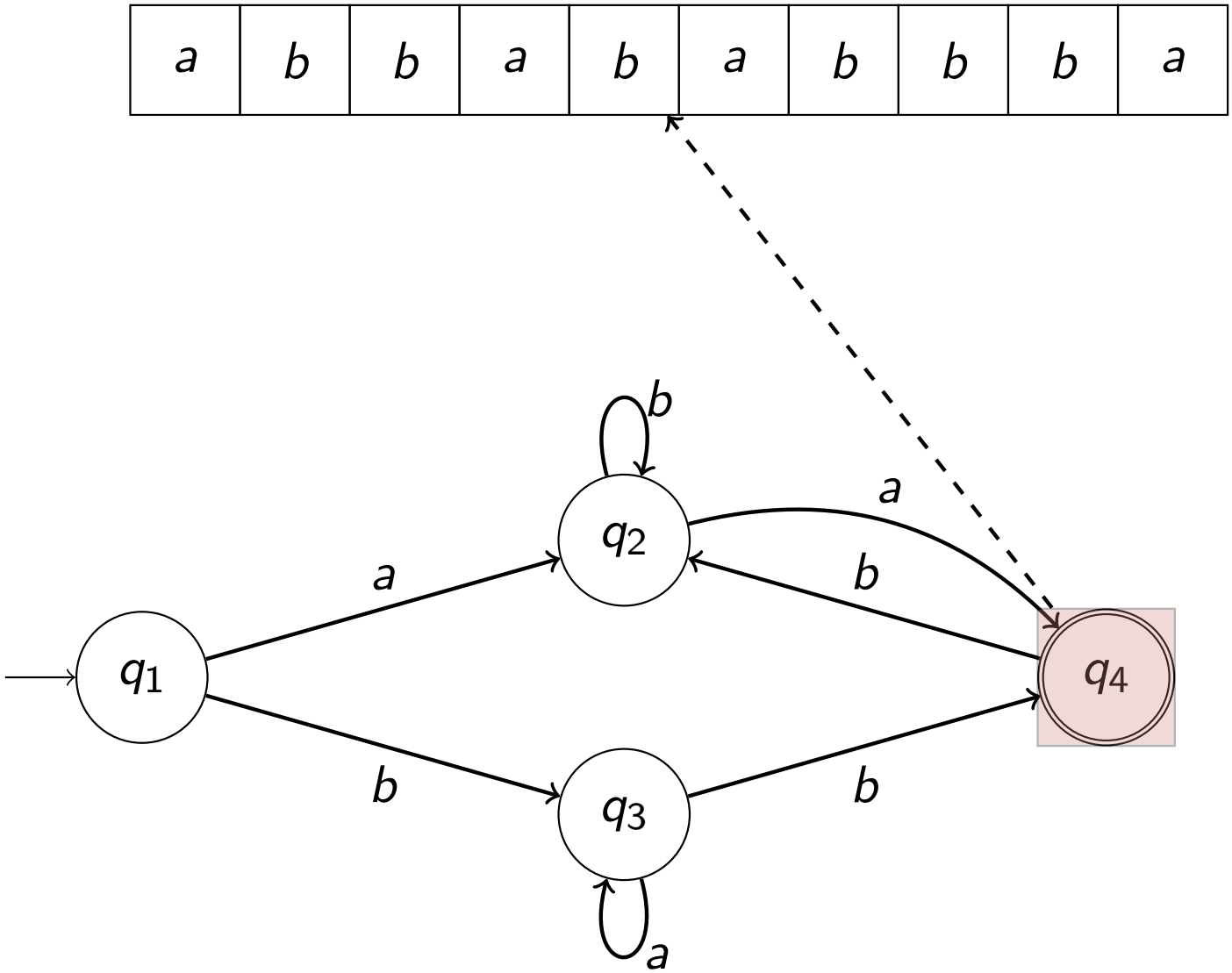
Rappel sur les automates déterministes (1)



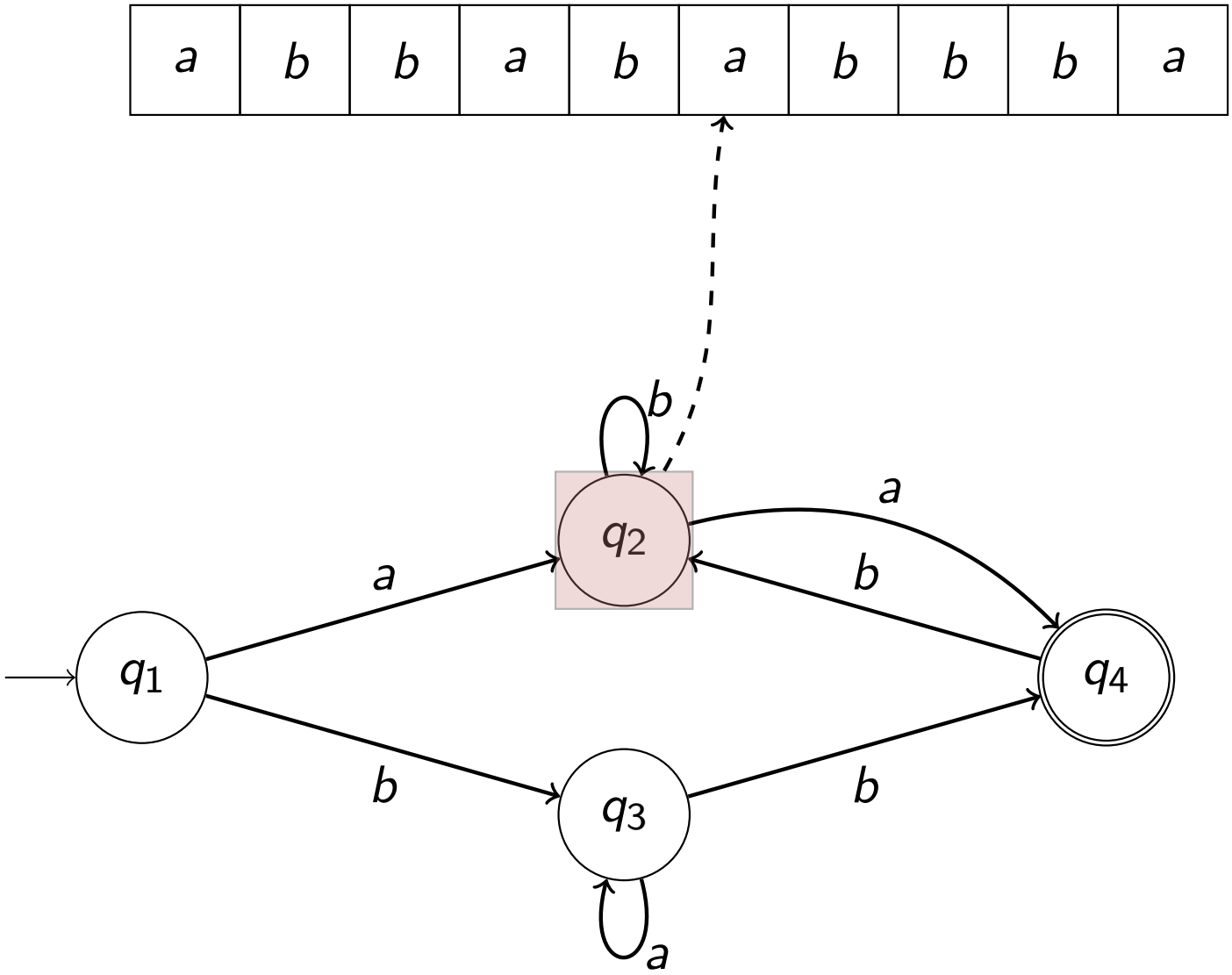
Rappel sur les automates déterministes (1)



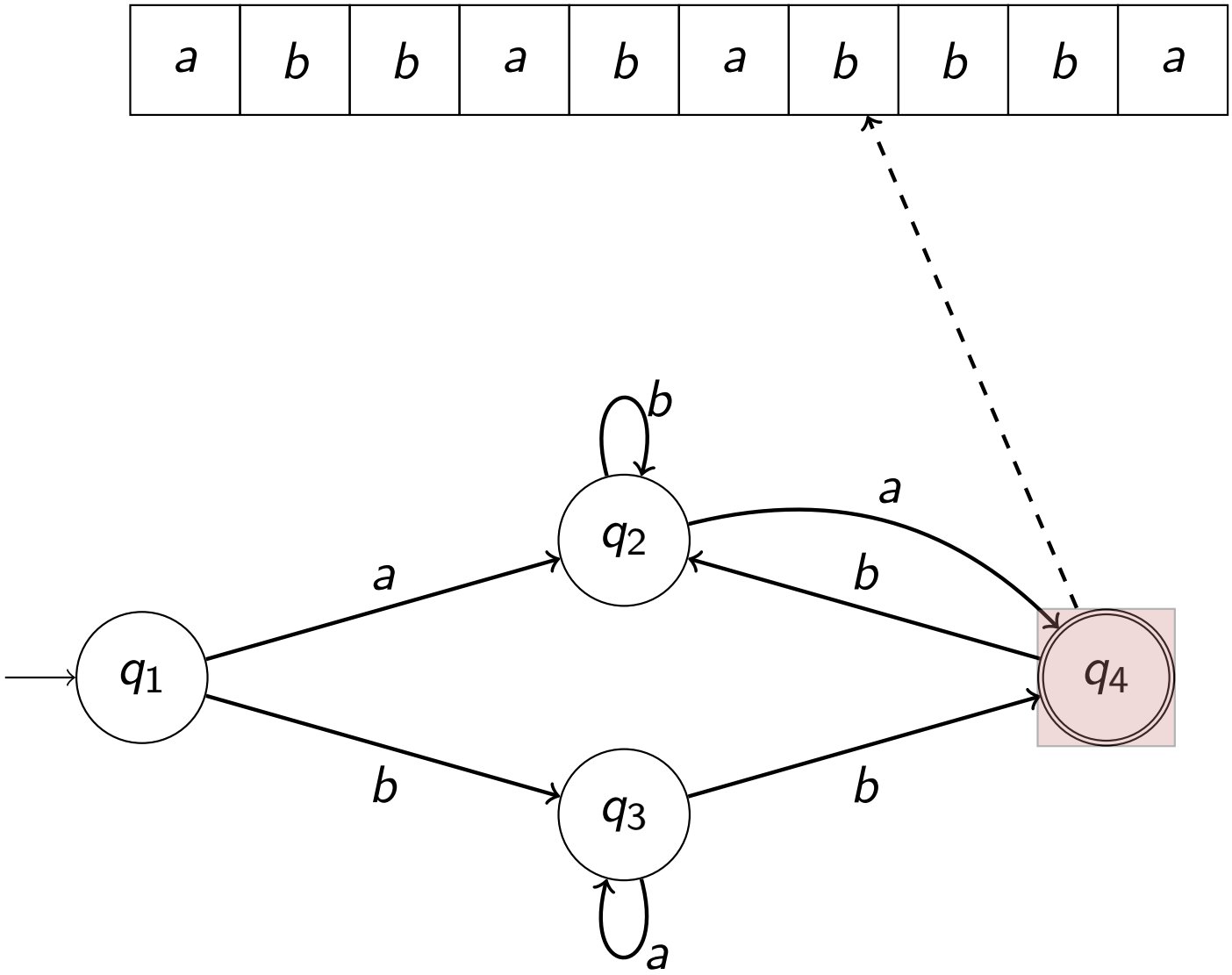
Rappel sur les automates déterministes (1)



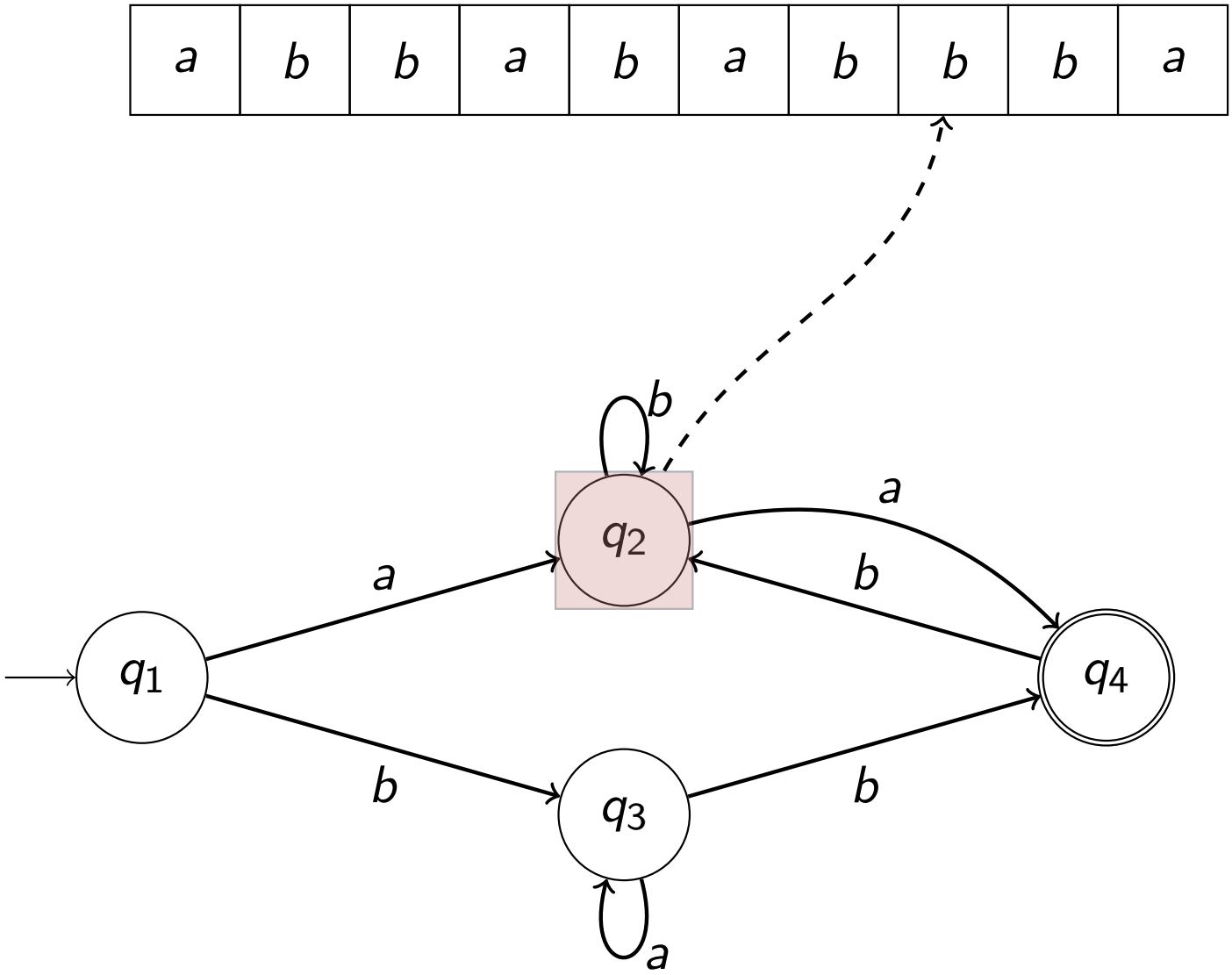
Rappel sur les automates déterministes (1)



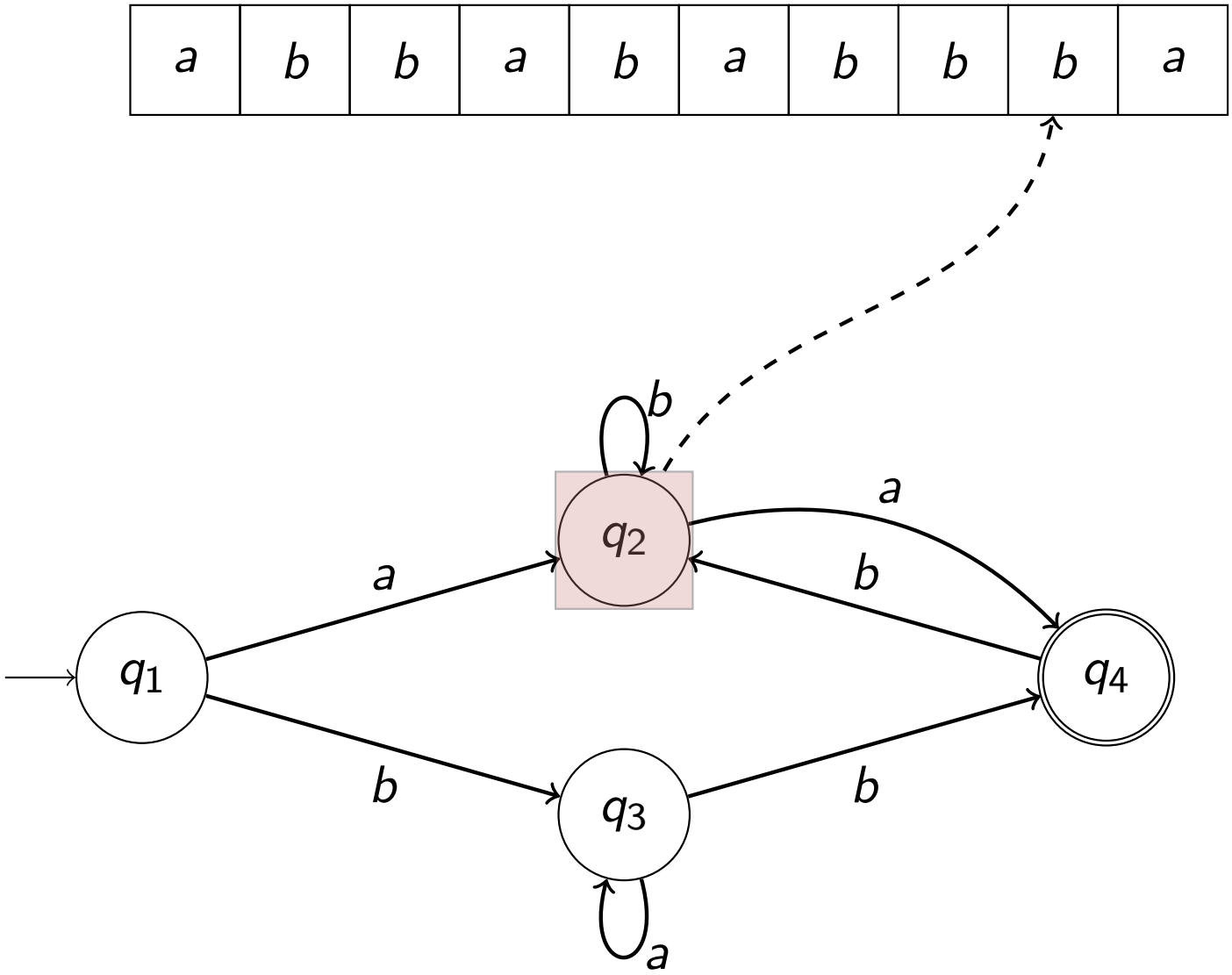
Rappel sur les automates déterministes (1)



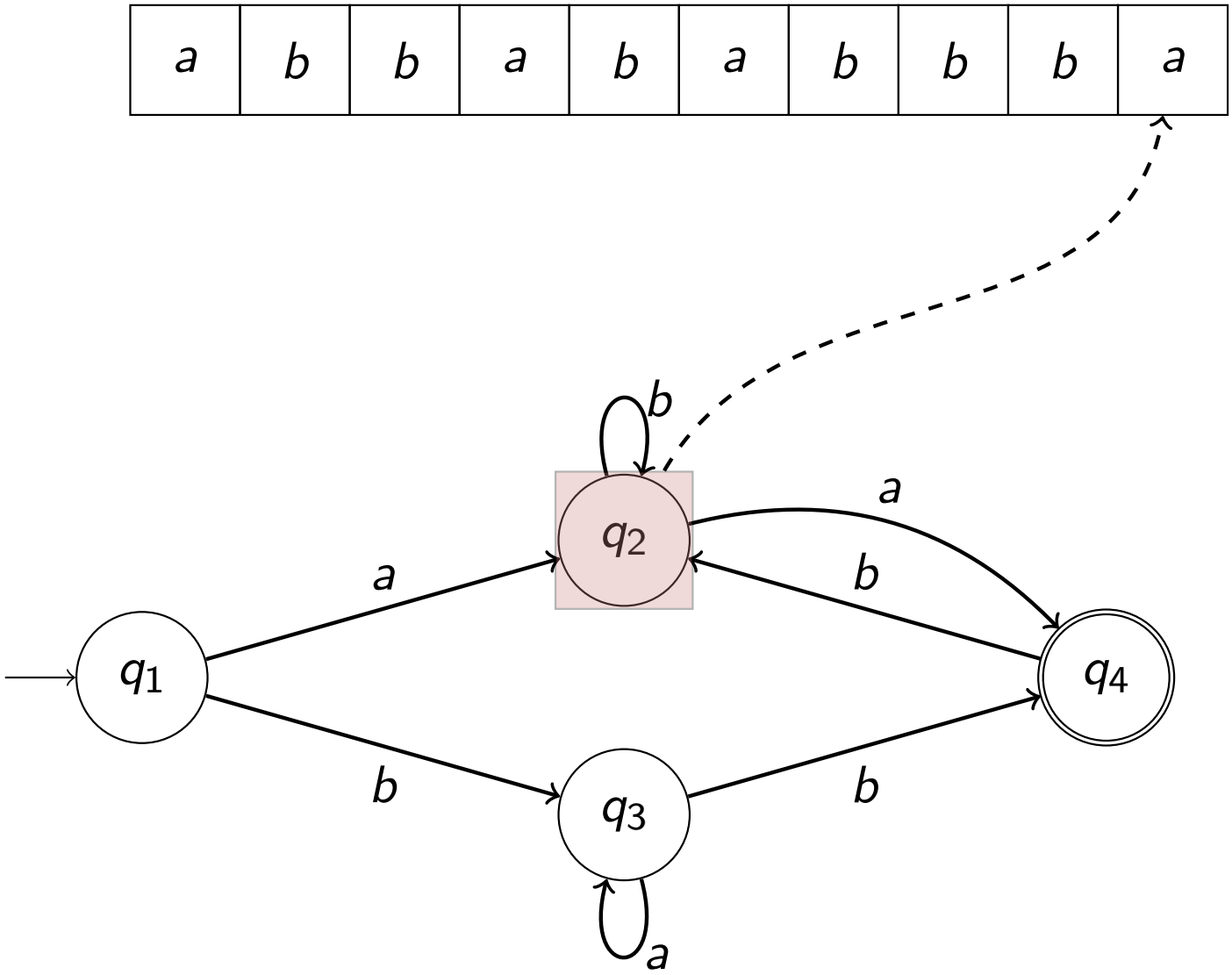
Rappel sur les automates déterministes (1)



Rappel sur les automates déterministes (1)



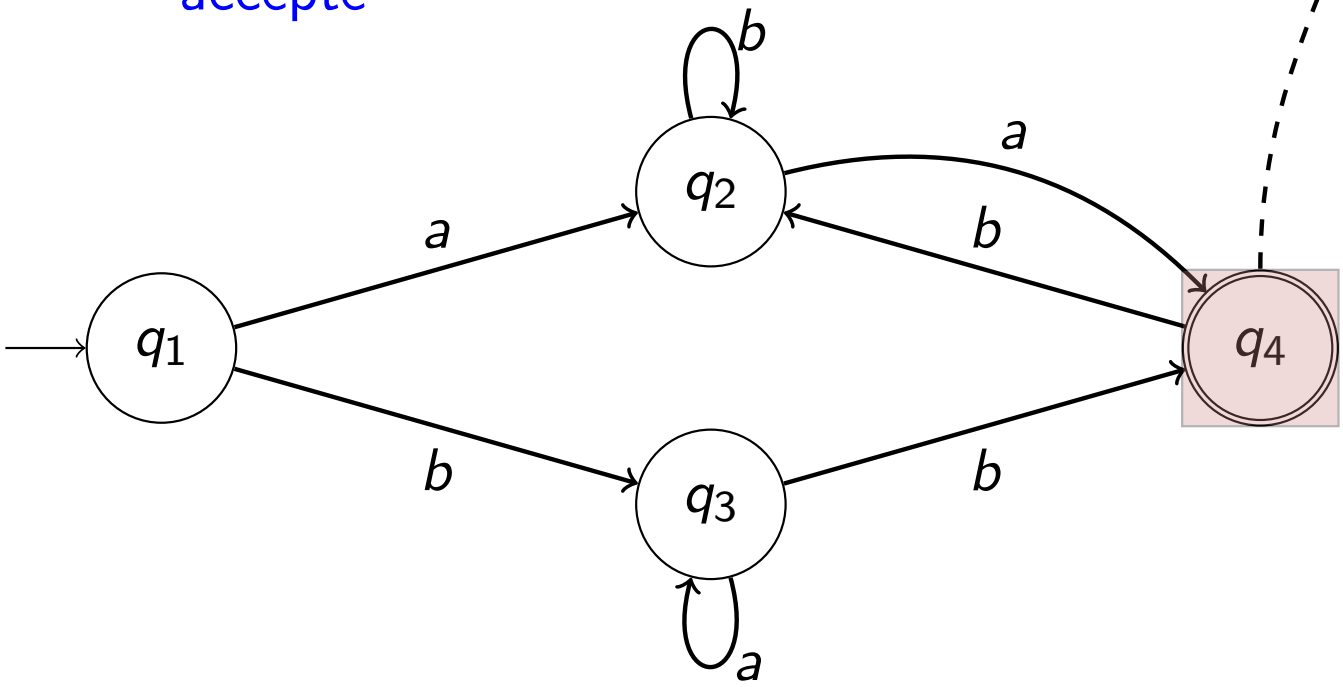
Rappel sur les automates déterministes (1)



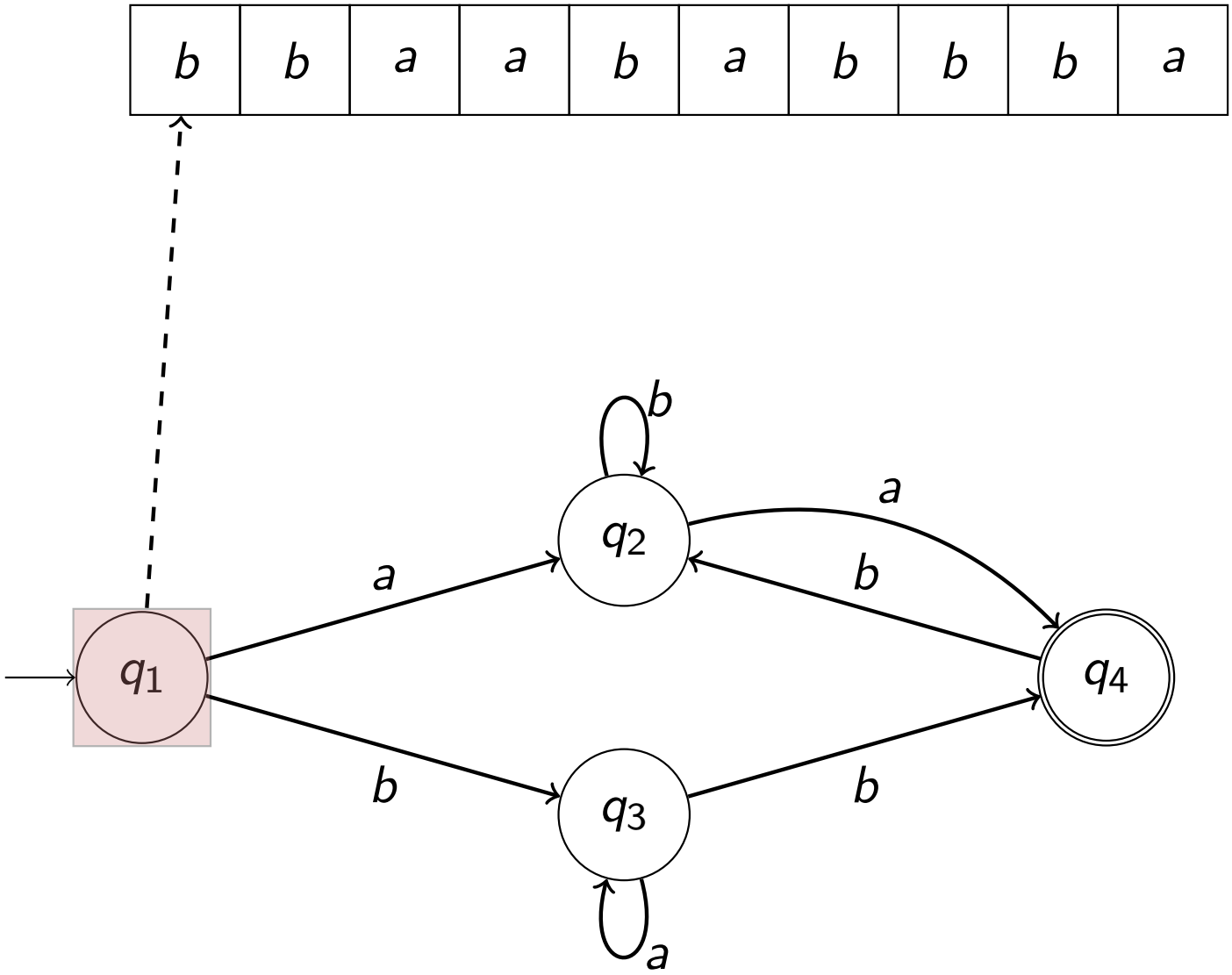
Rappel sur les automates déterministes (1)

a	b	b	a	b	a	b	b	b	a

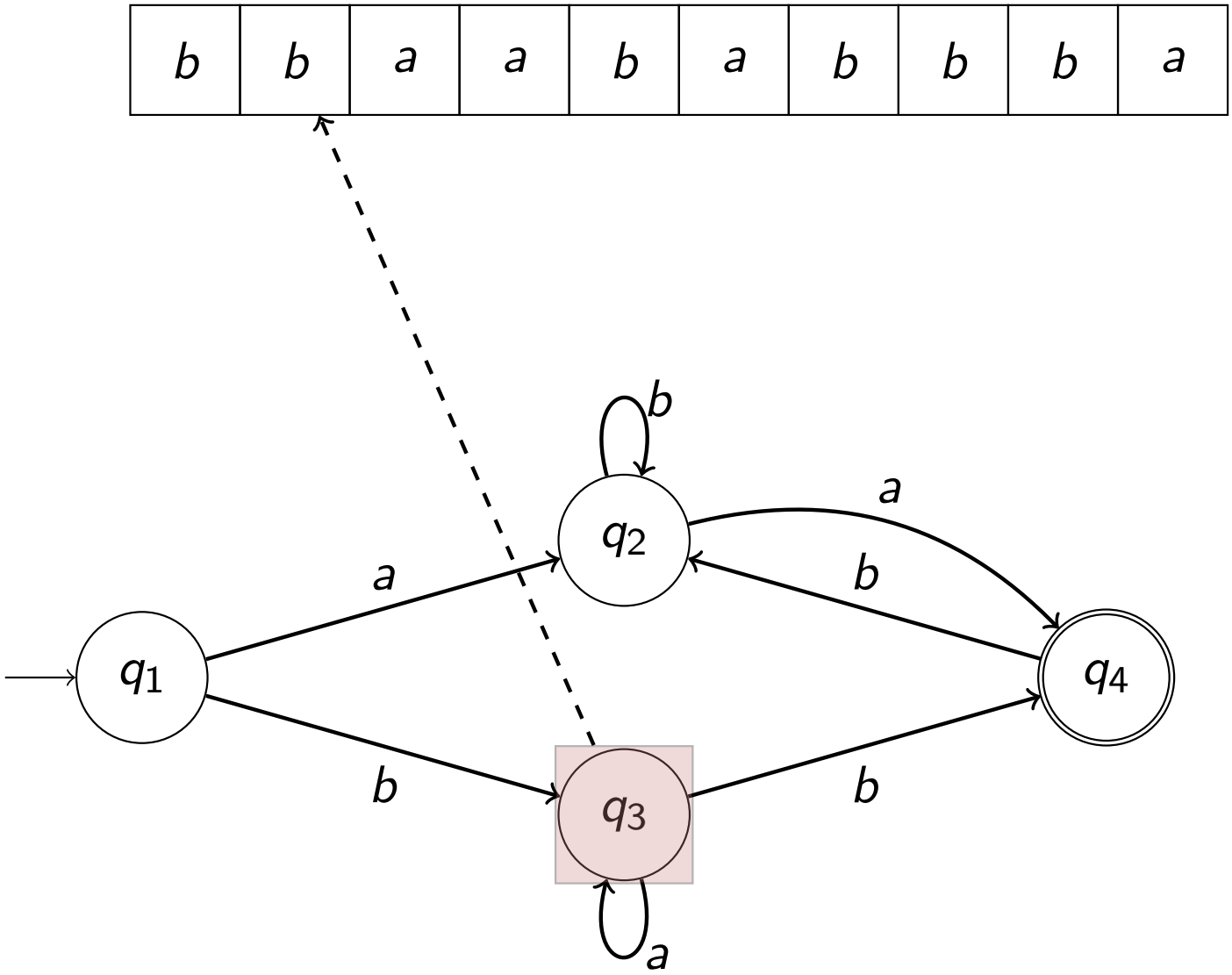
Le mot est accepté



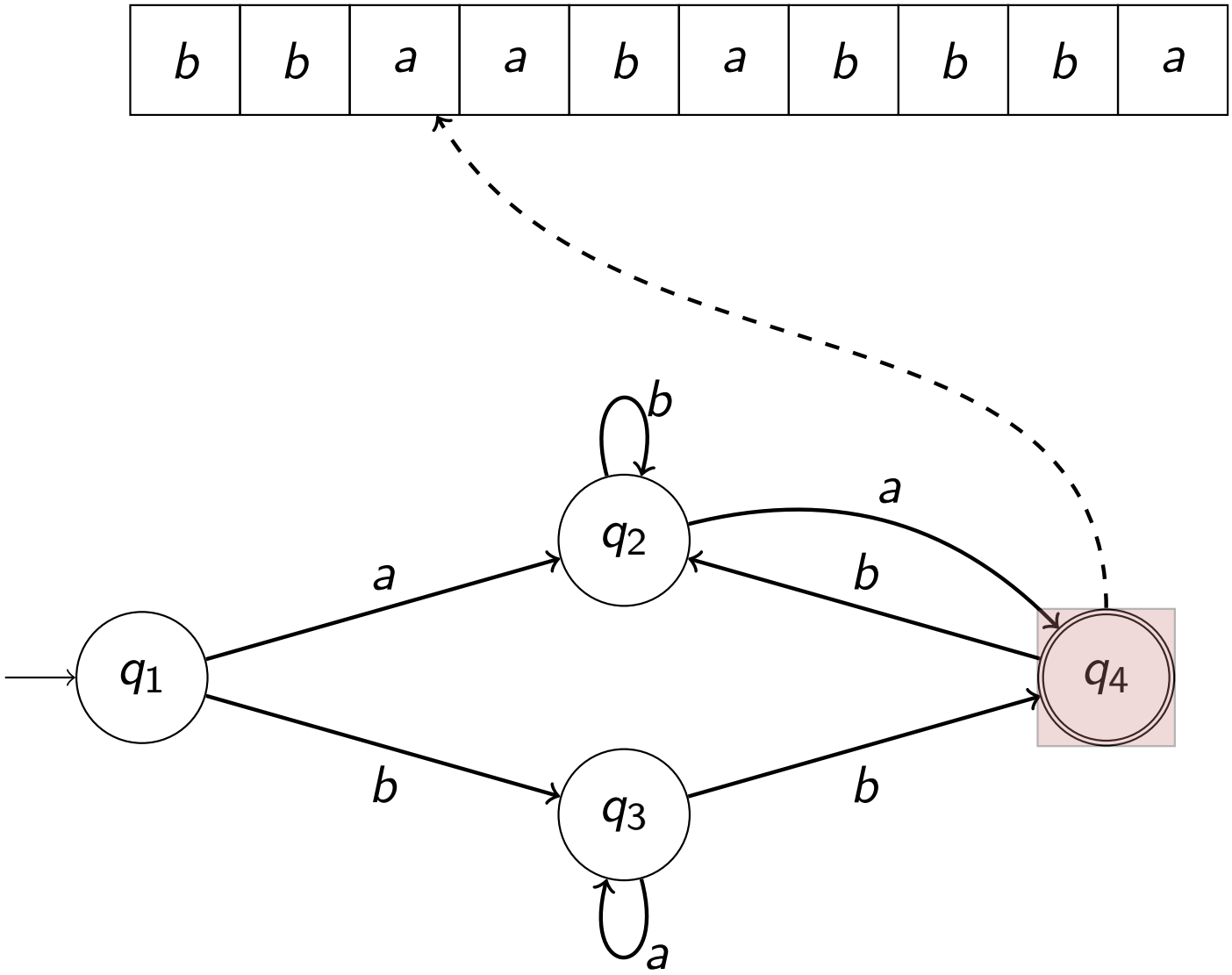
Rappel sur les automates déterministes (2)



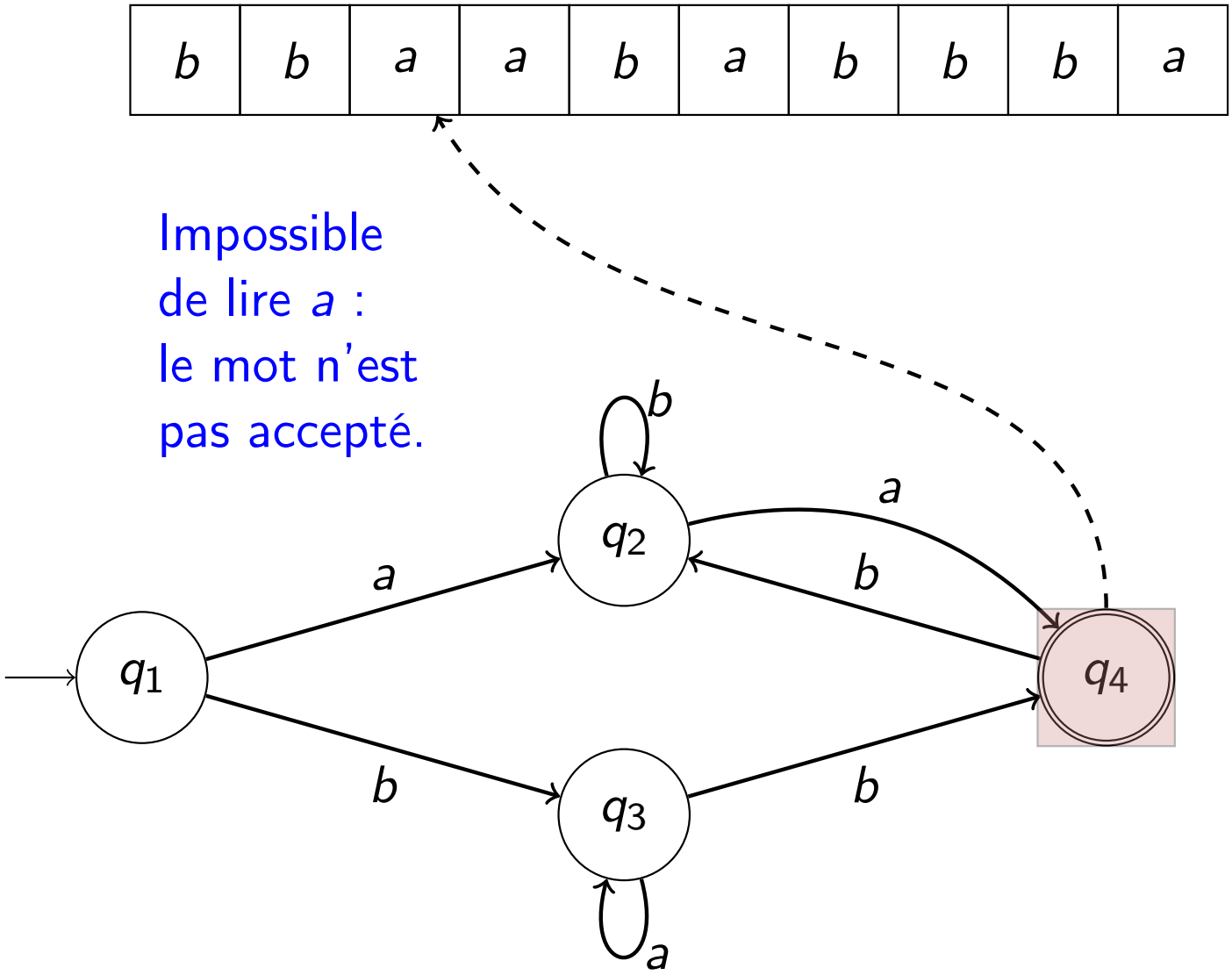
Rappel sur les automates déterministes (2)



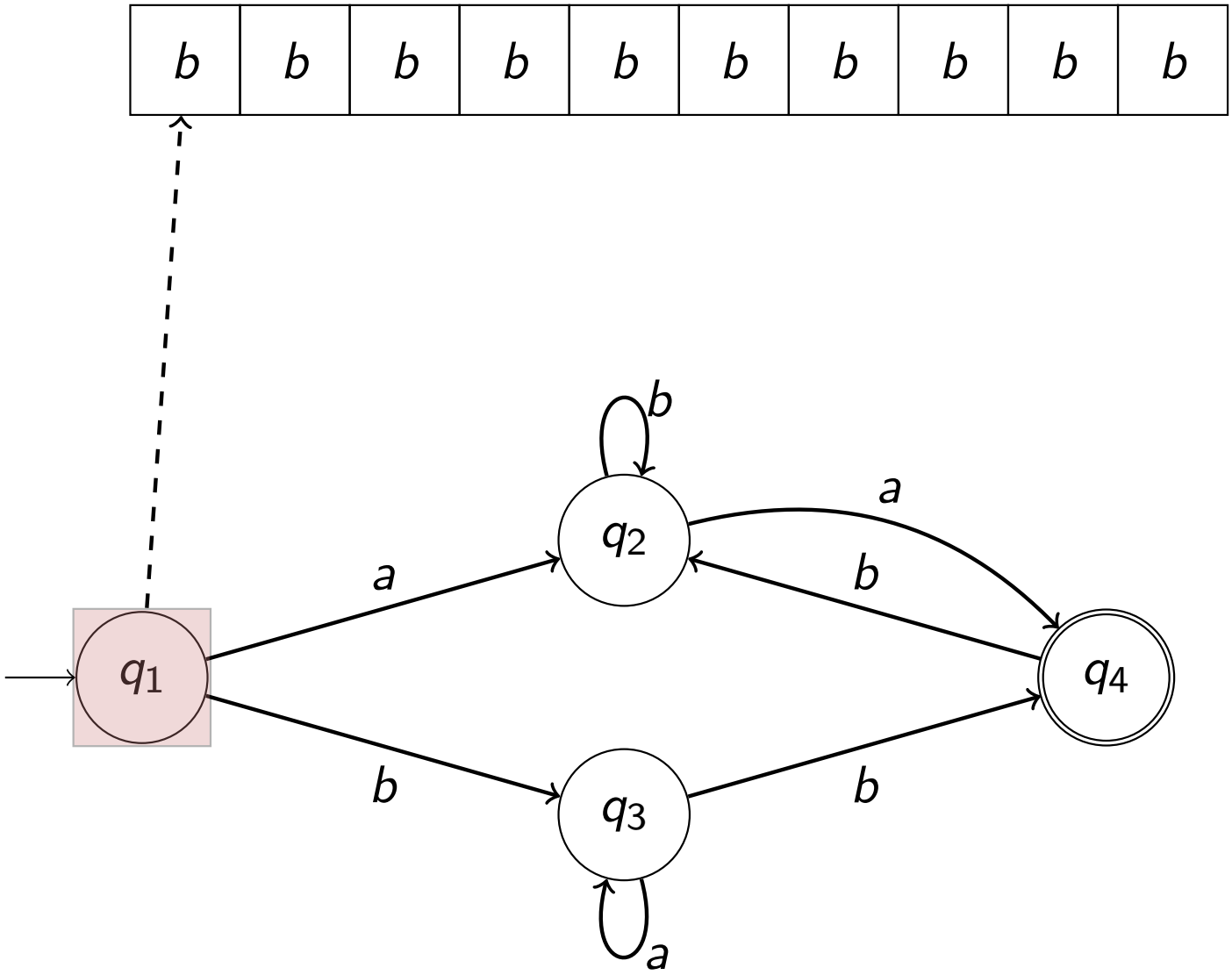
Rappel sur les automates déterministes (2)



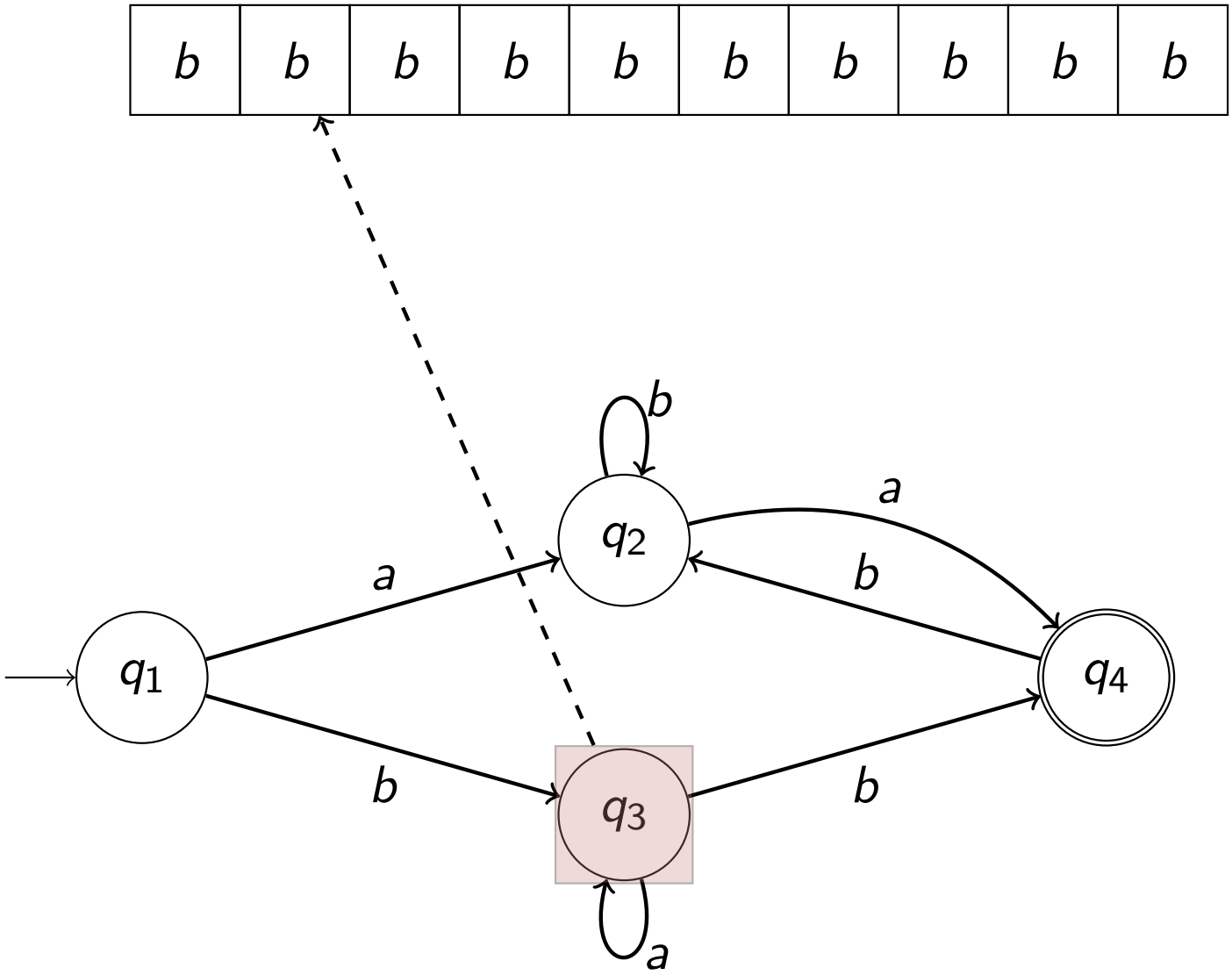
Rappel sur les automates déterministes (2)



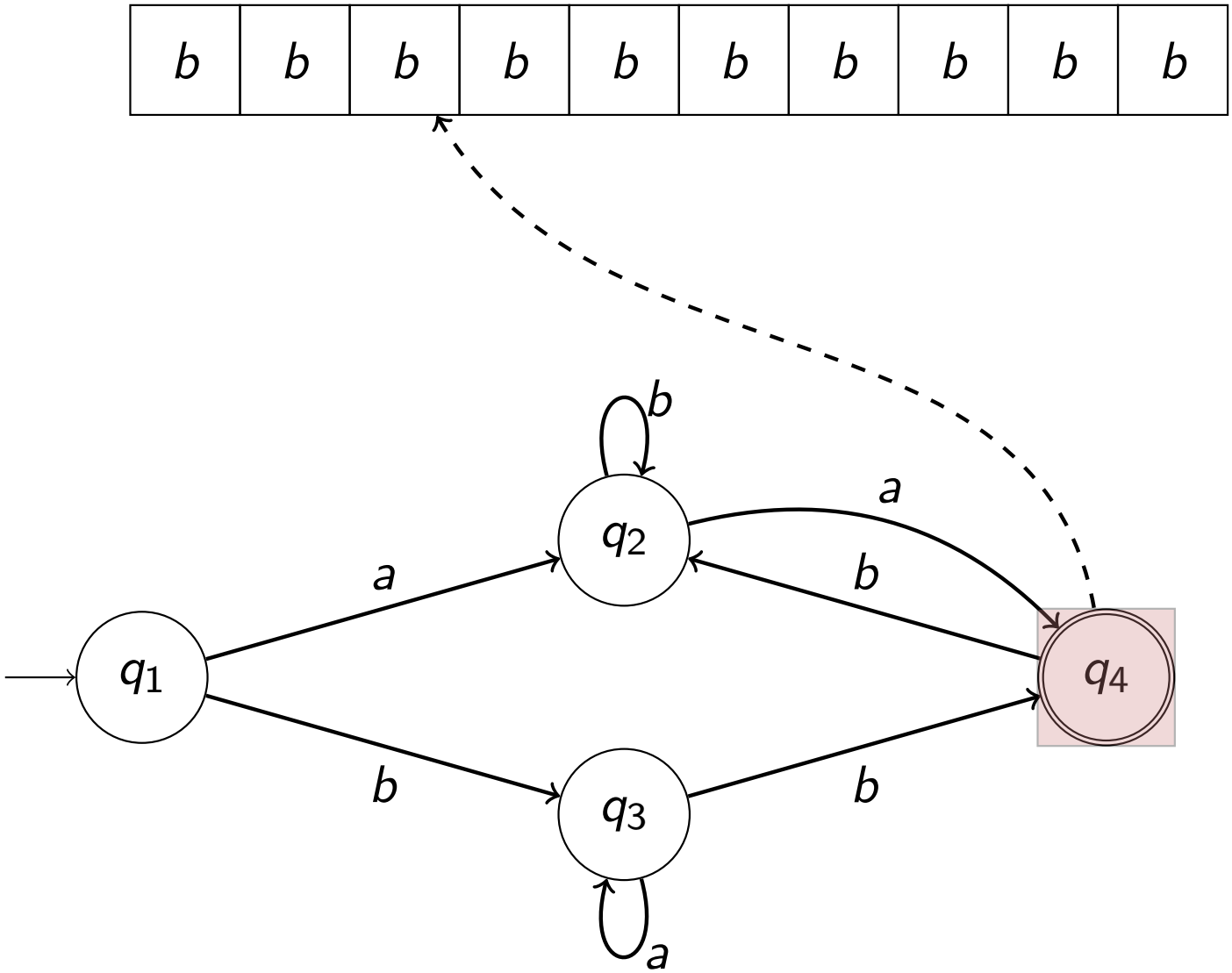
Rappel sur les automates déterministes (3)



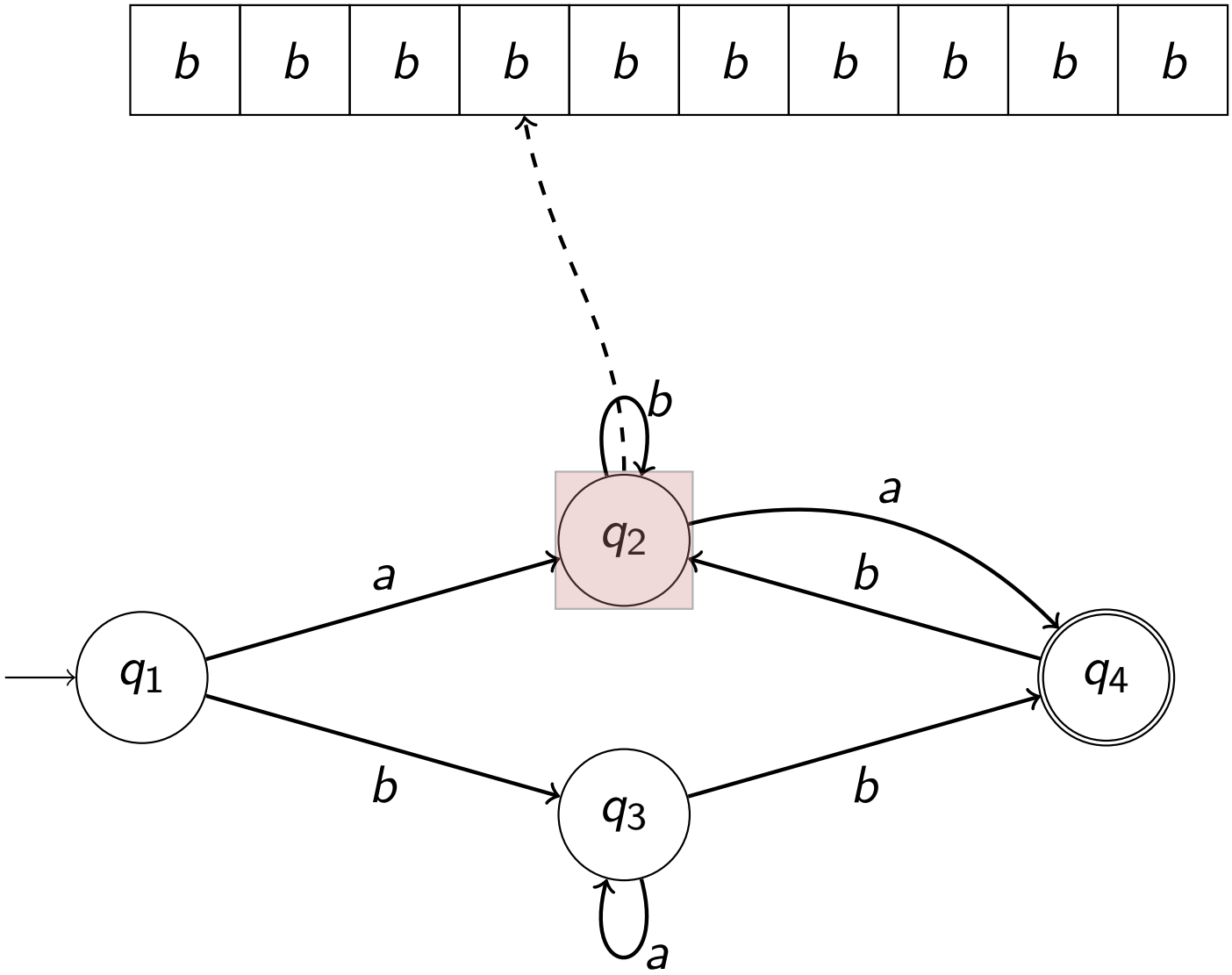
Rappel sur les automates déterministes (3)



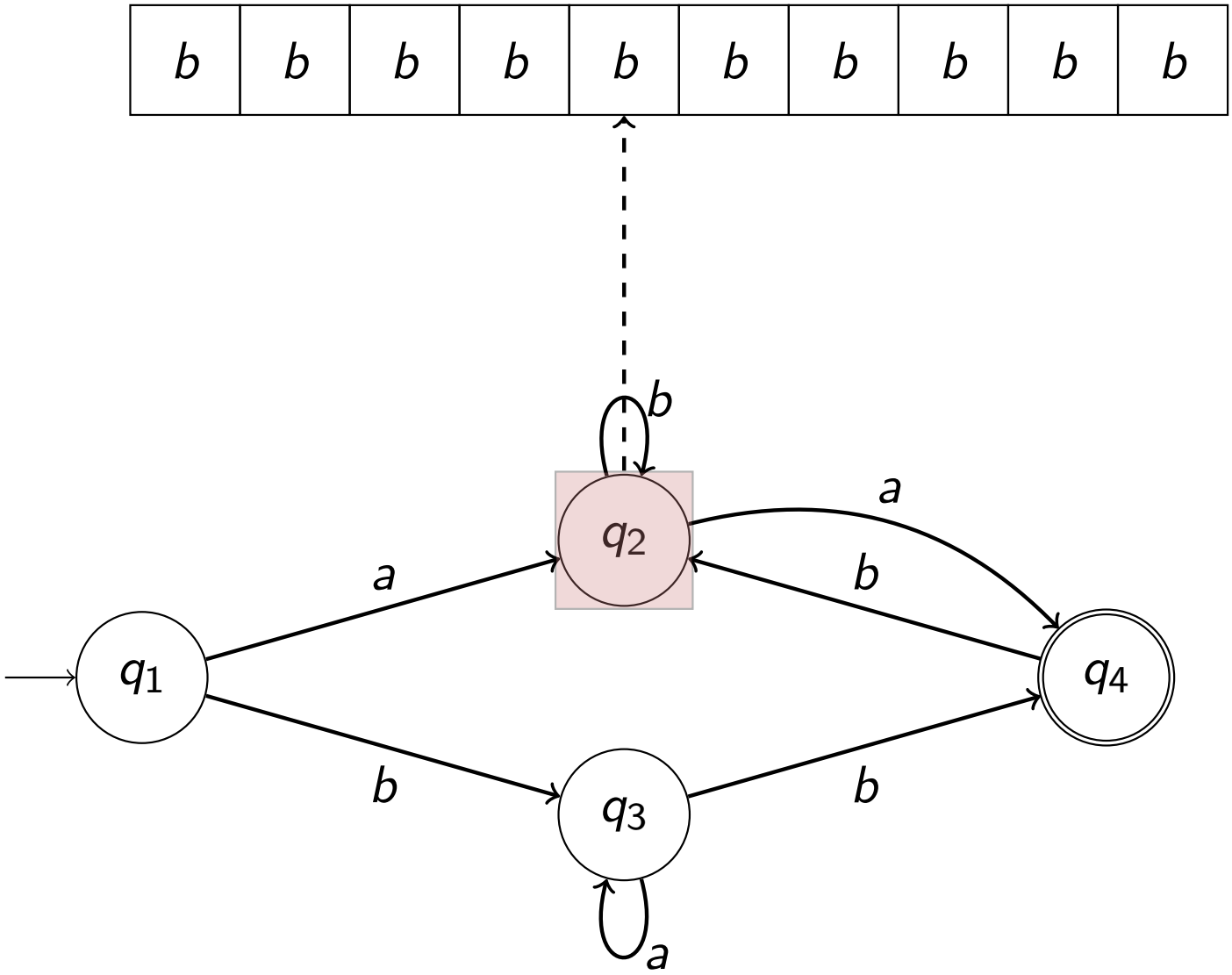
Rappel sur les automates déterministes (3)



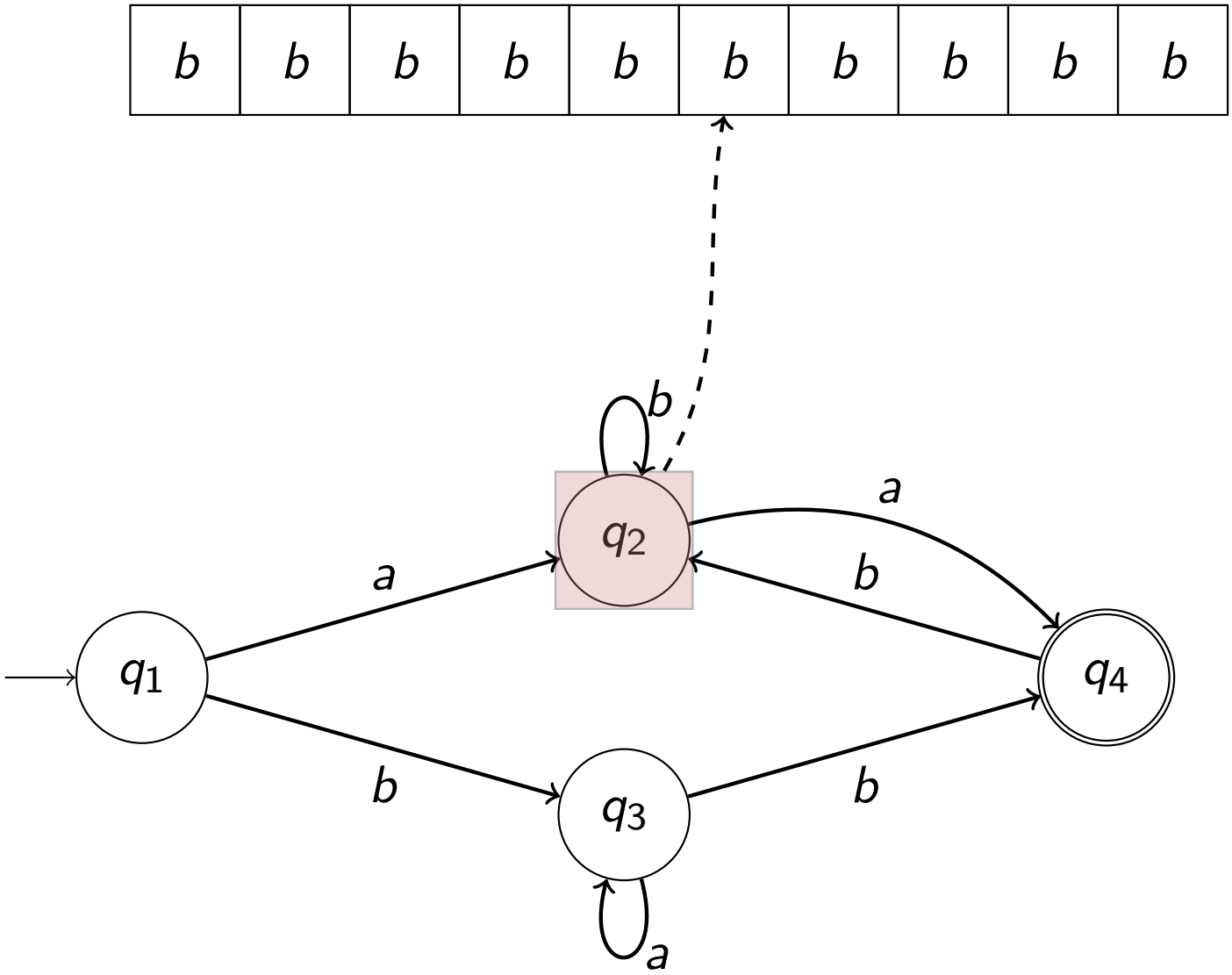
Rappel sur les automates déterministes (3)



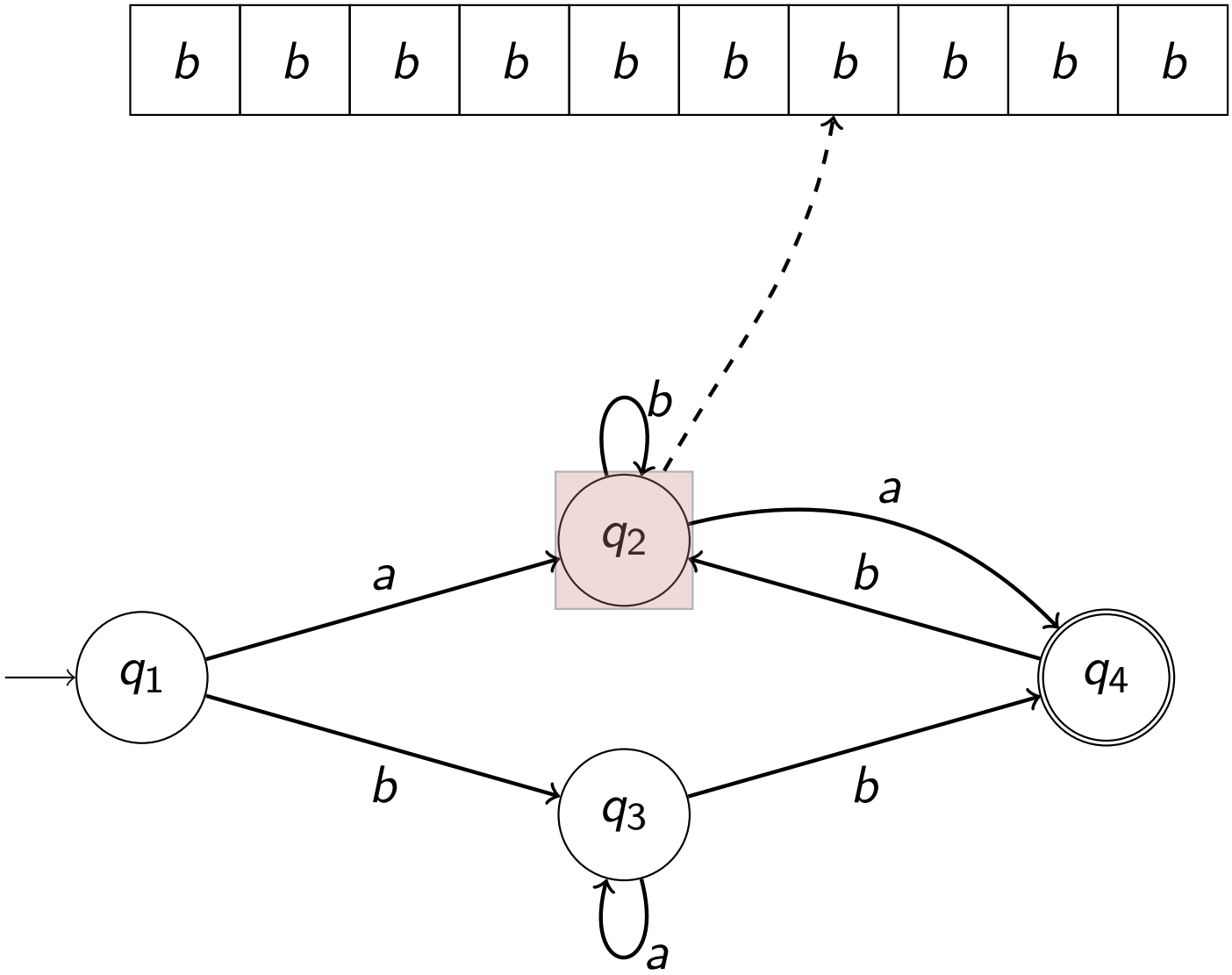
Rappel sur les automates déterministes (3)



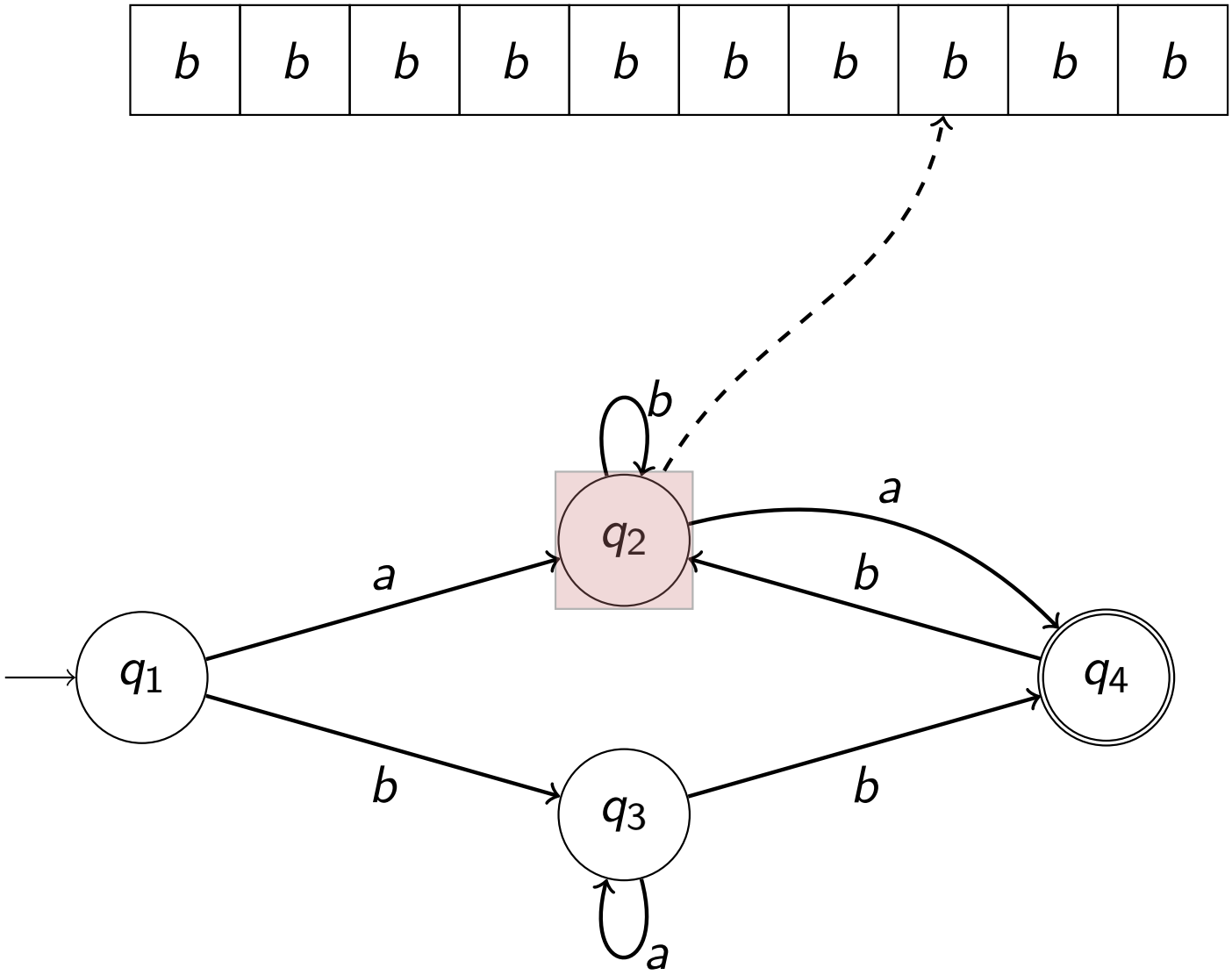
Rappel sur les automates déterministes (3)



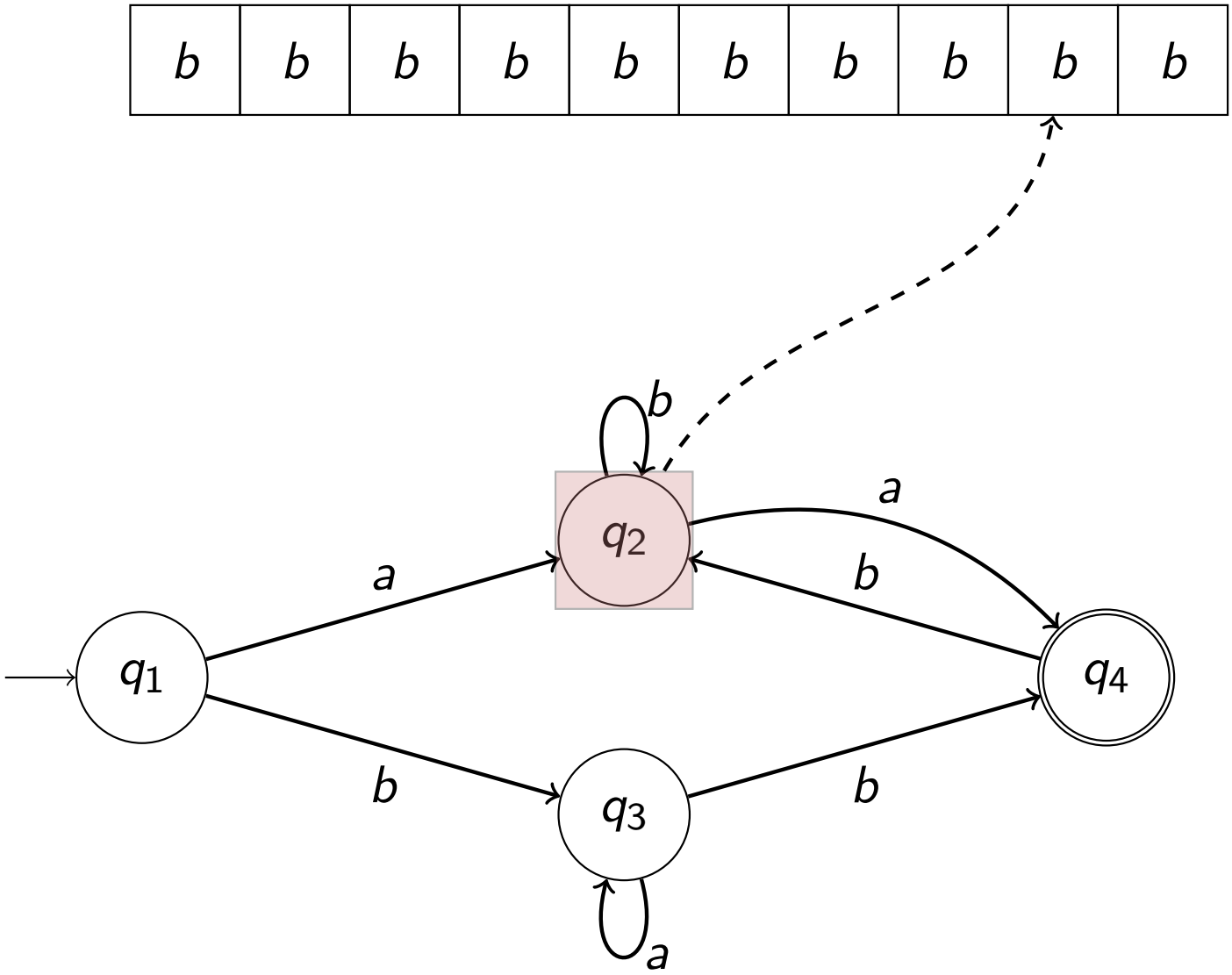
Rappel sur les automates déterministes (3)



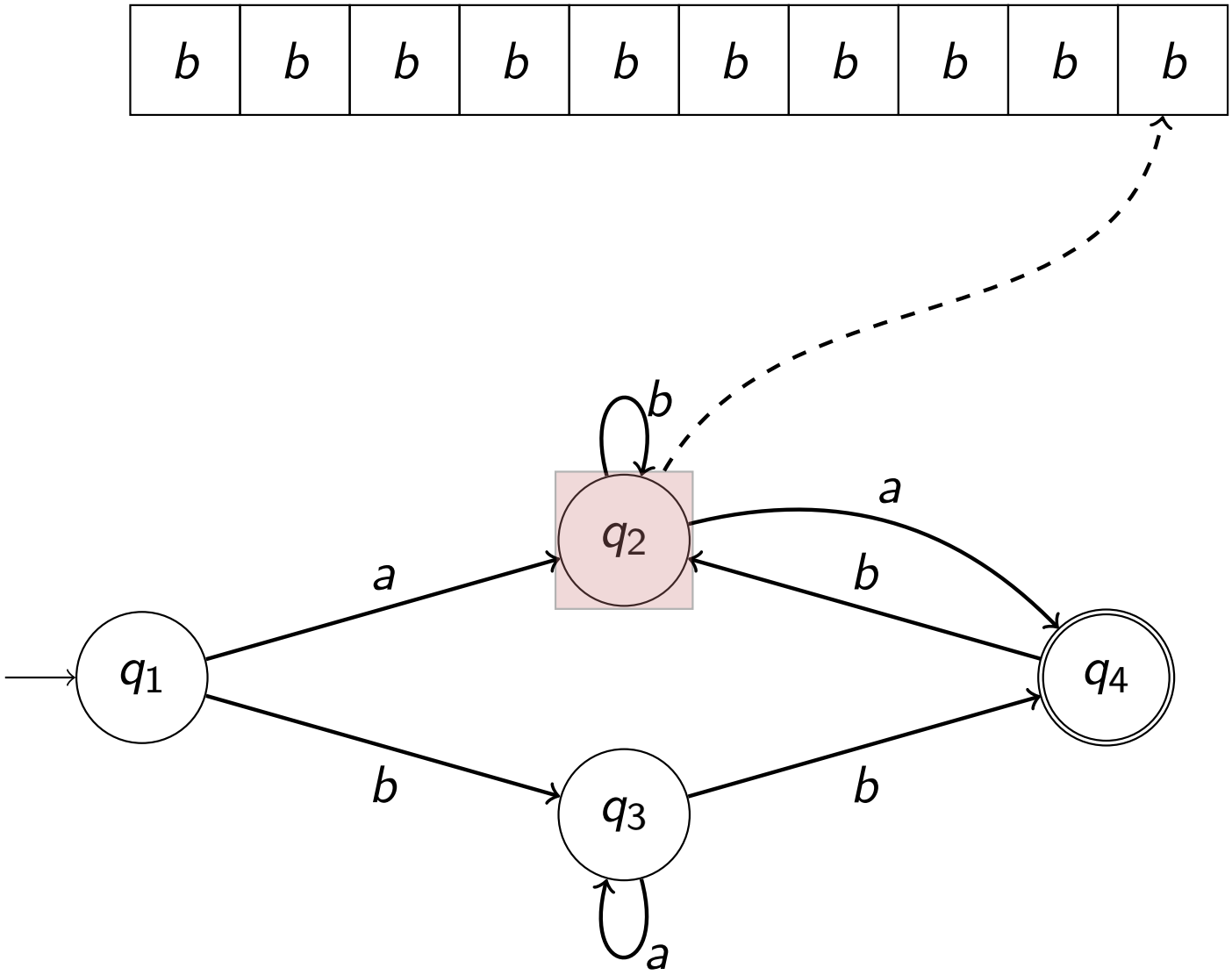
Rappel sur les automates déterministes (3)



Rappel sur les automates déterministes (3)



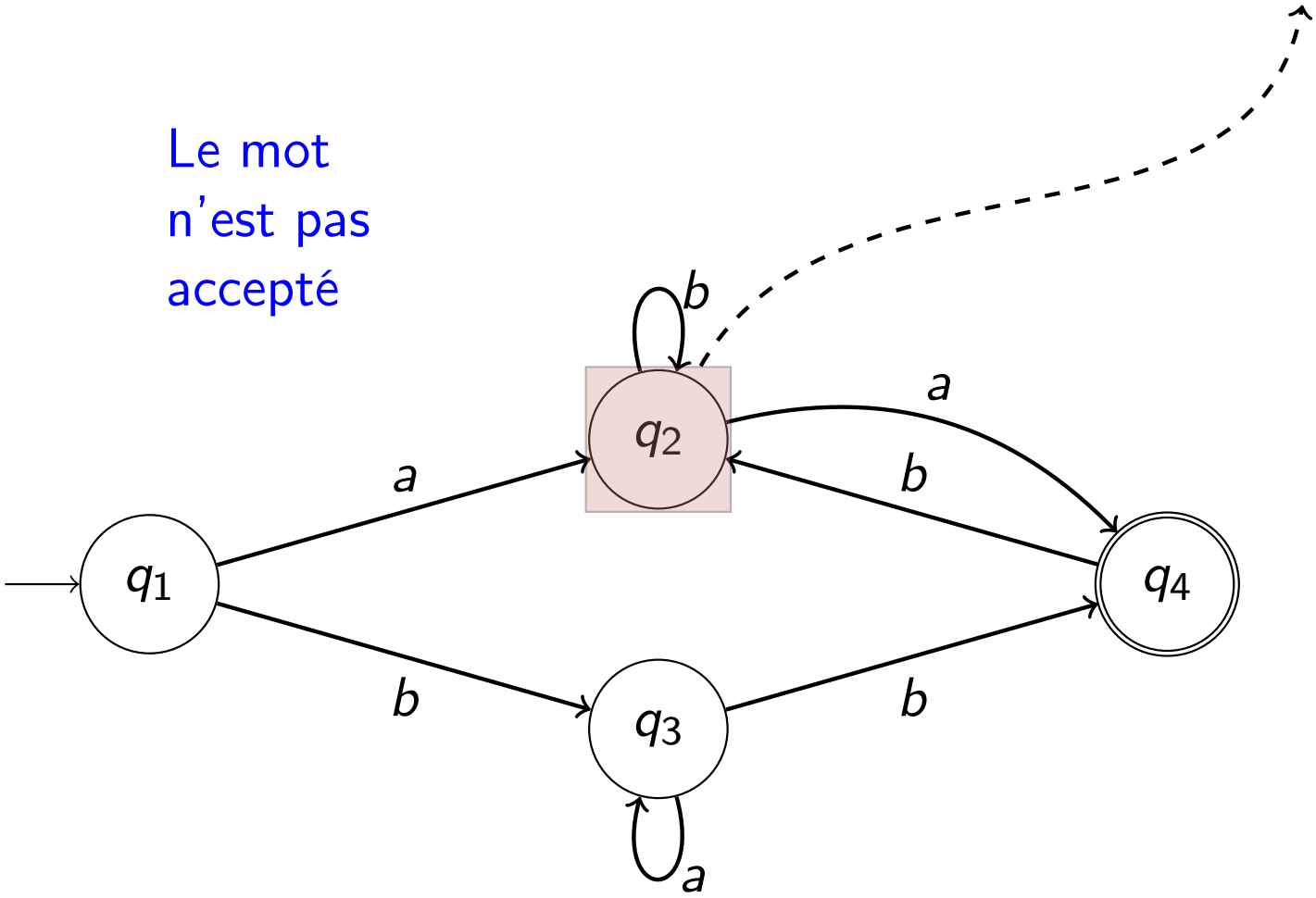
Rappel sur les automates déterministes (3)



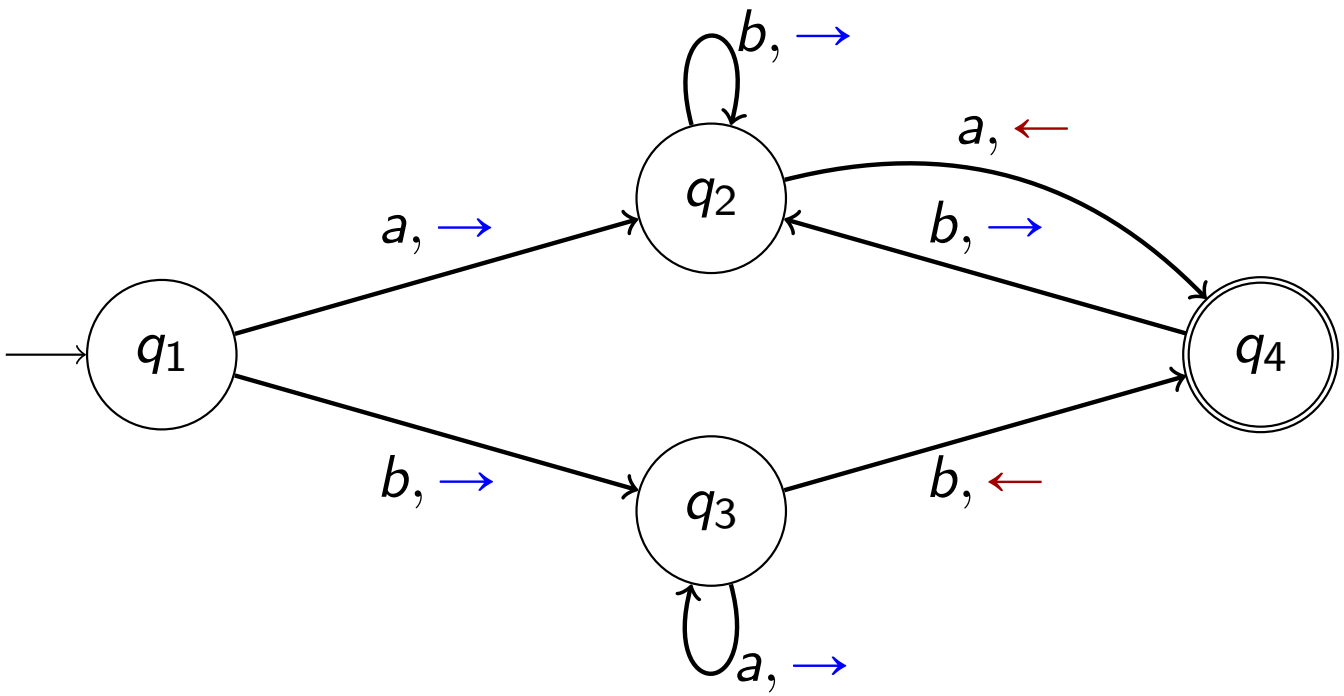
Rappel sur les automates déterministes (3)

b	b	b	b	b	b	b	b	b	b

Le mot
n'est pas
accepté

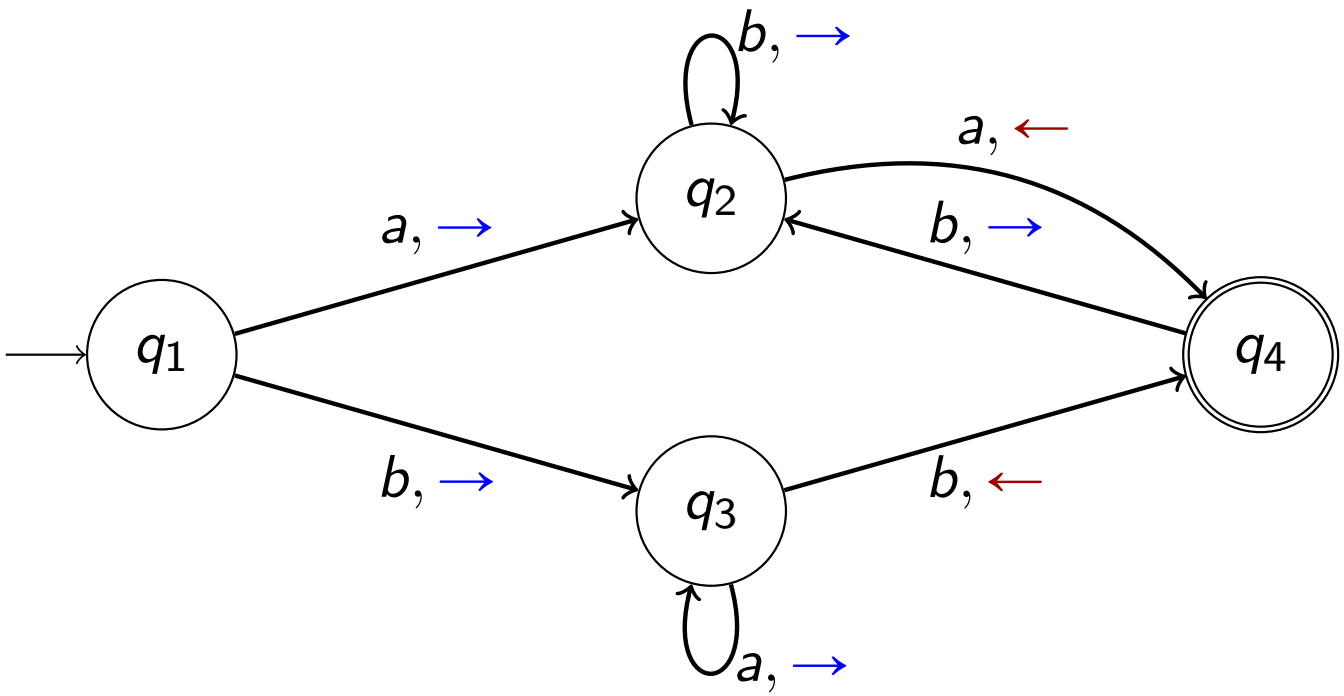


Automates *two ways*

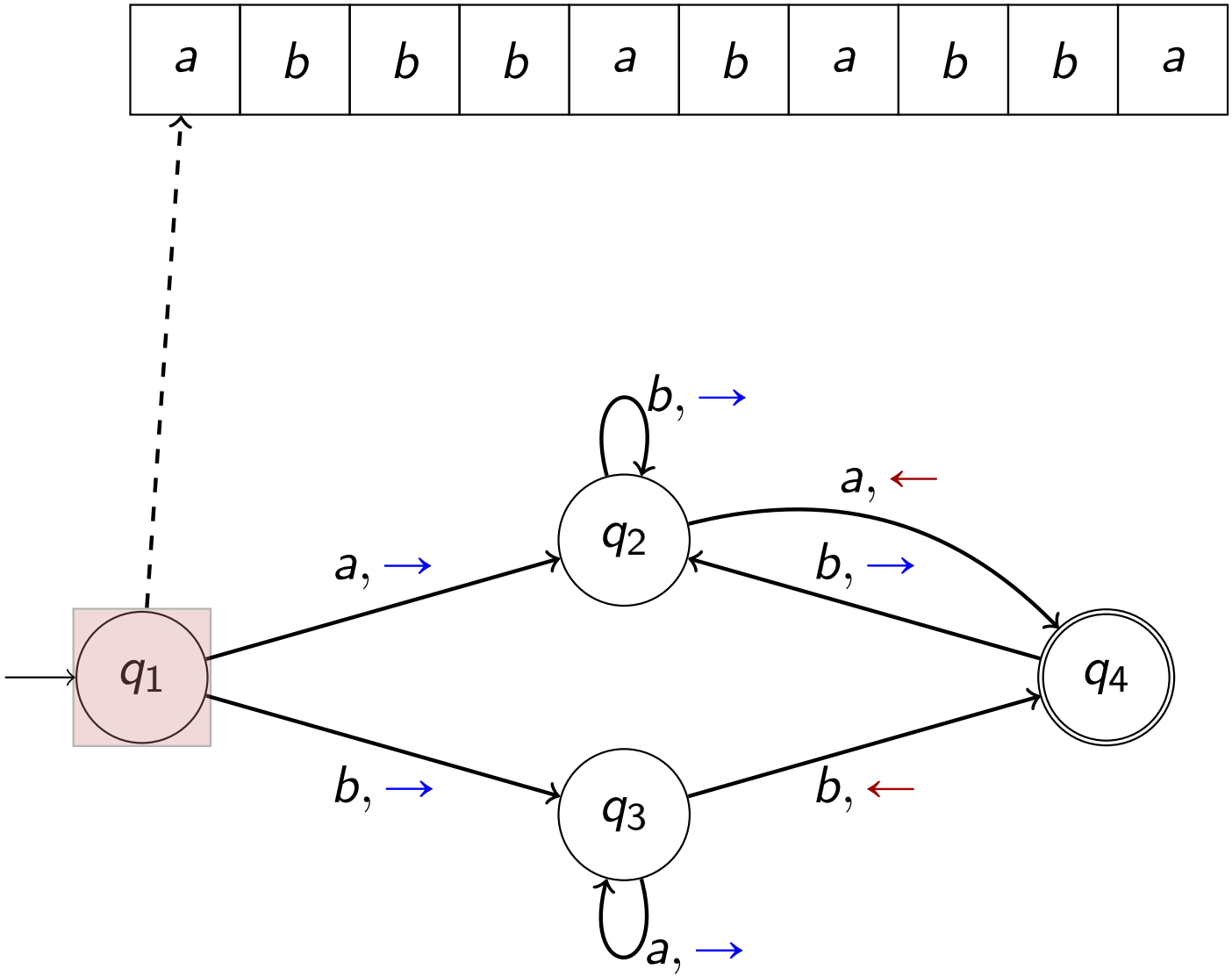


Automates *two ways*

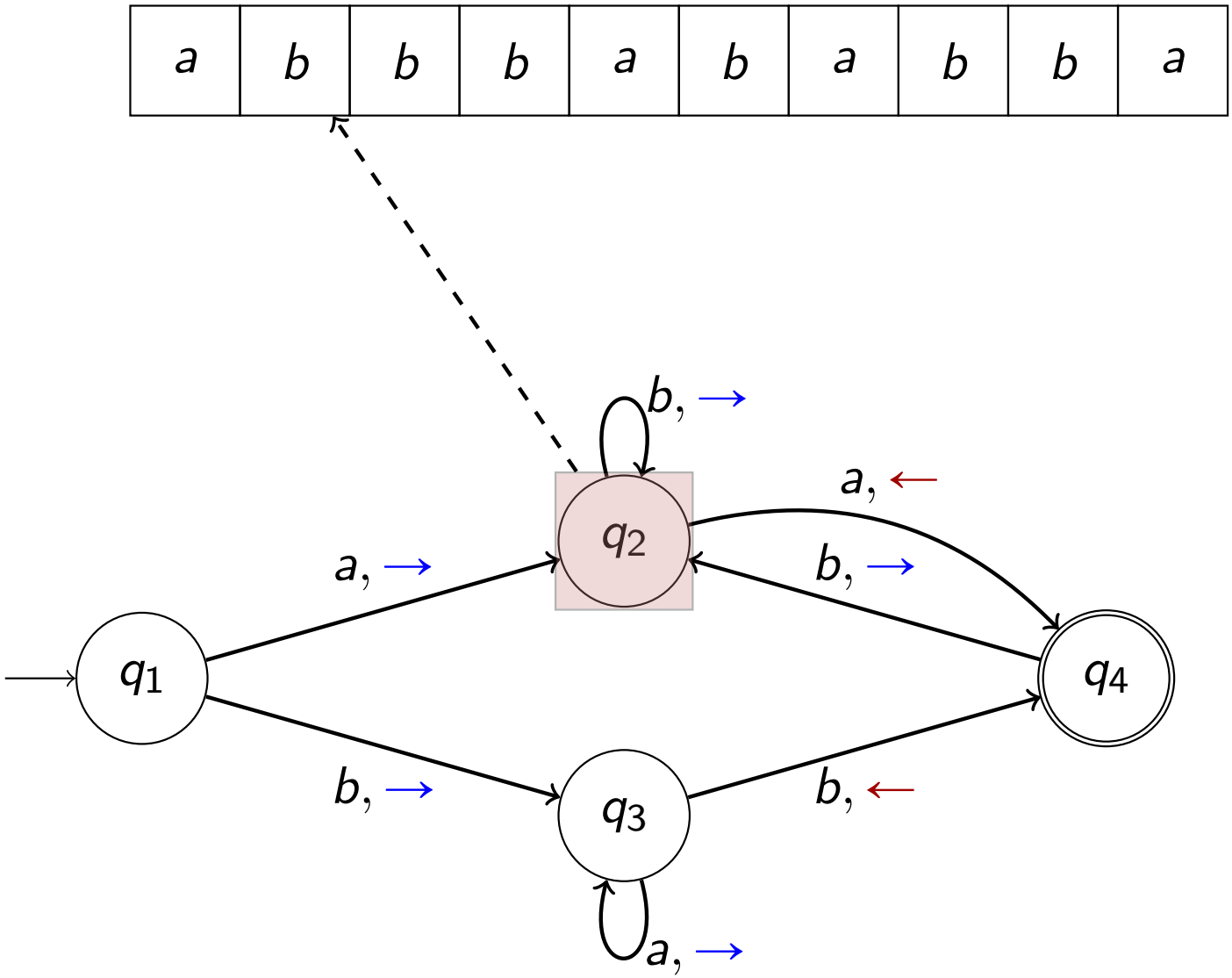
a	b	b	b	a	b	a	b	b	a



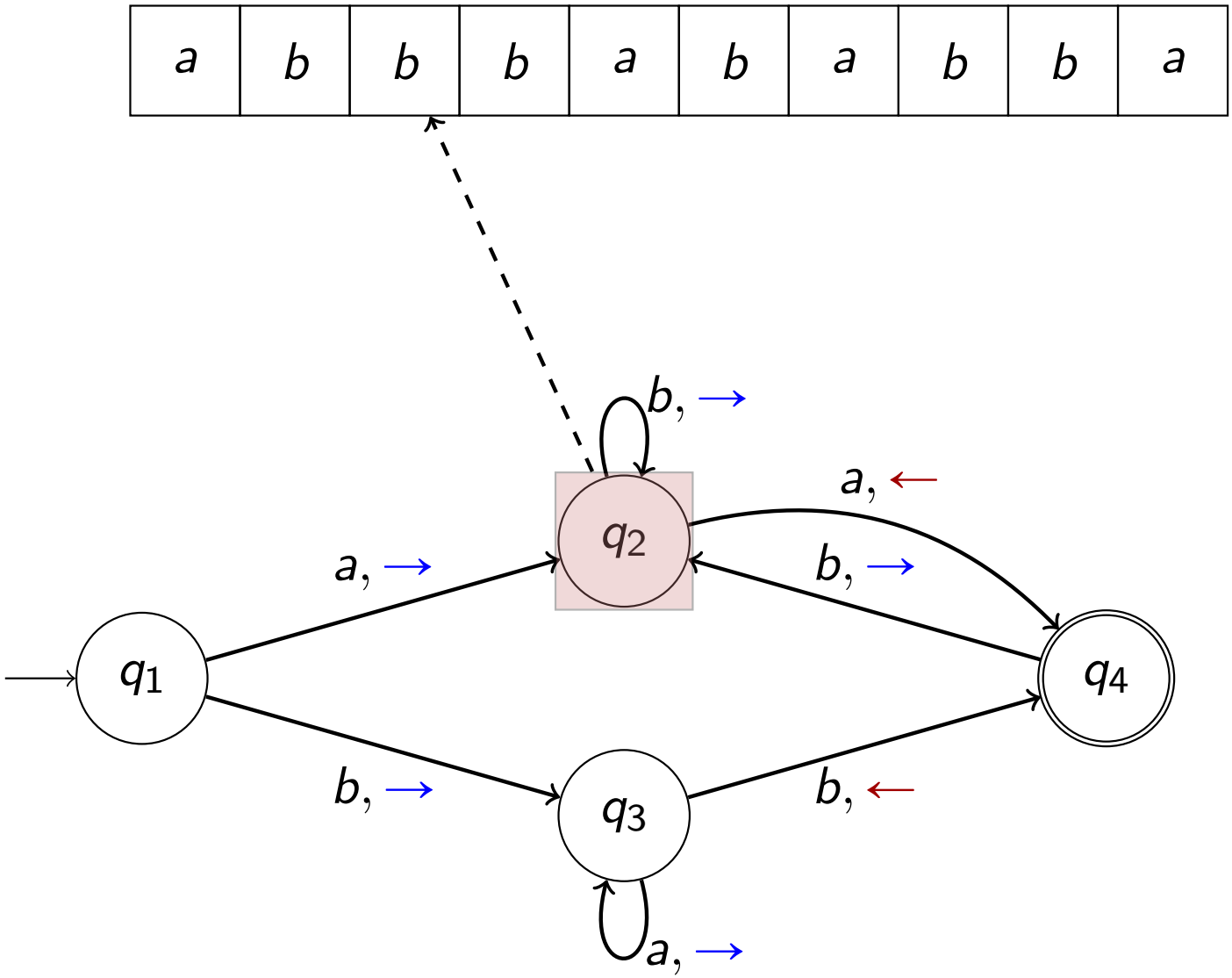
Automates *two ways*



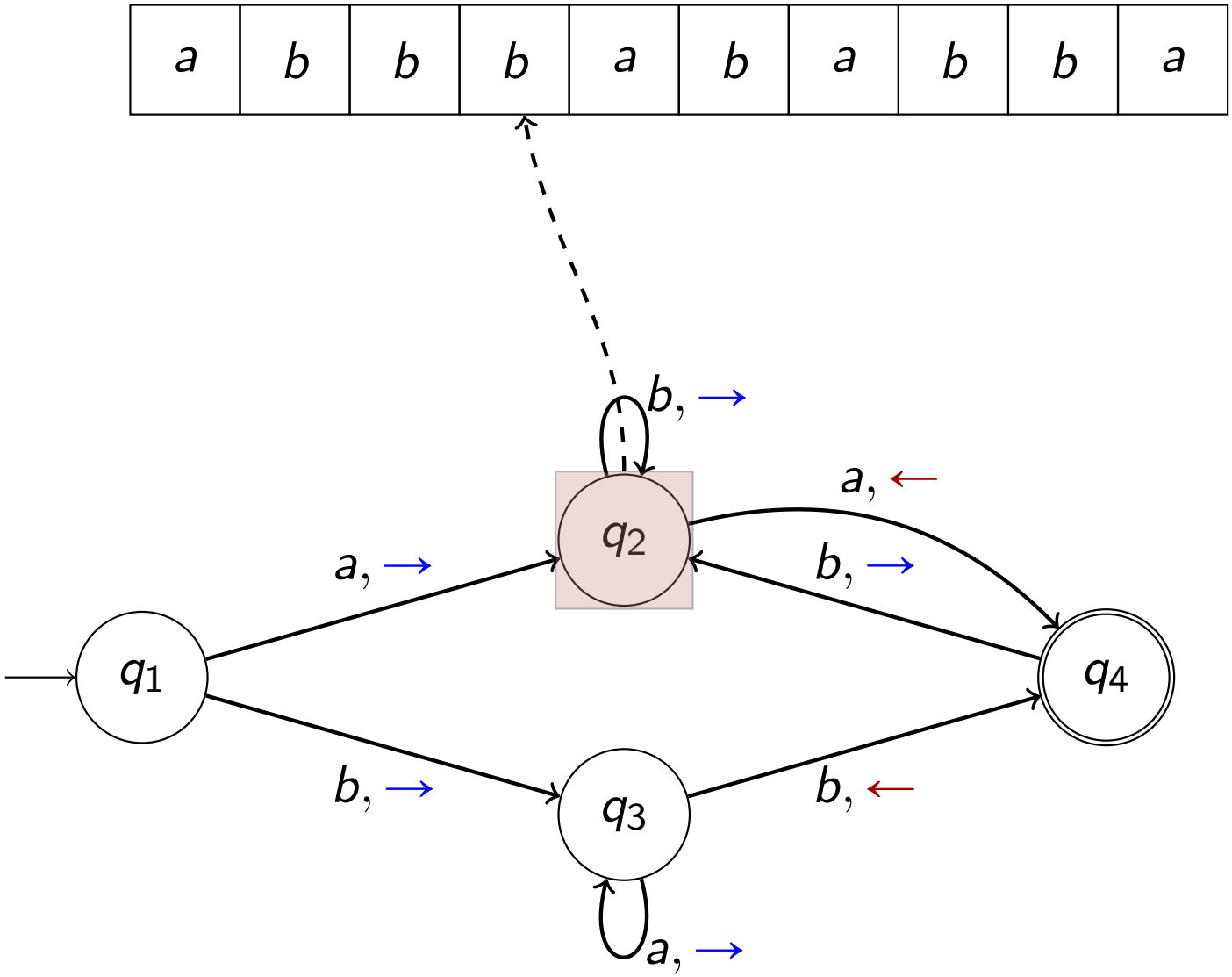
Automates *two ways*



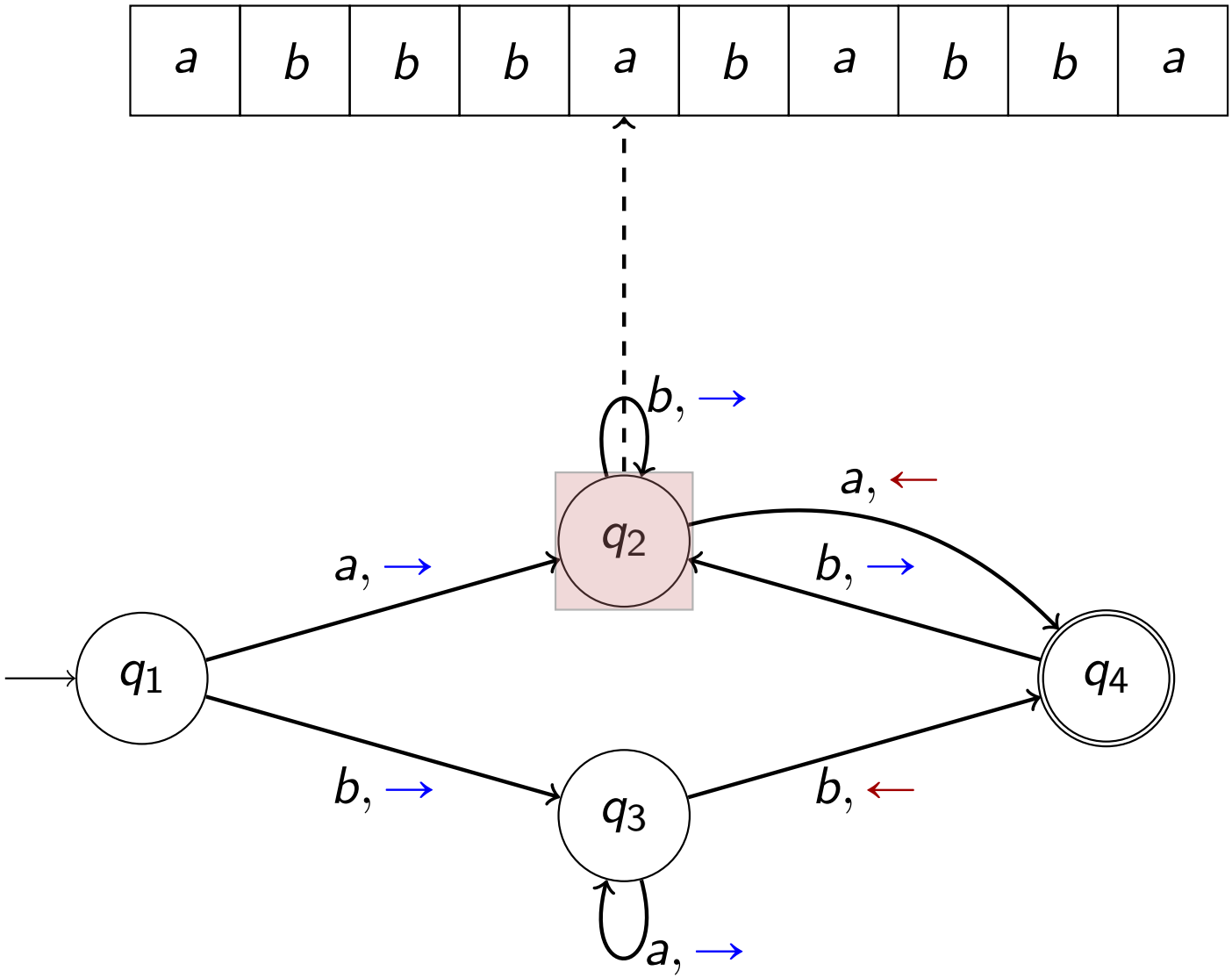
Automates *two ways*



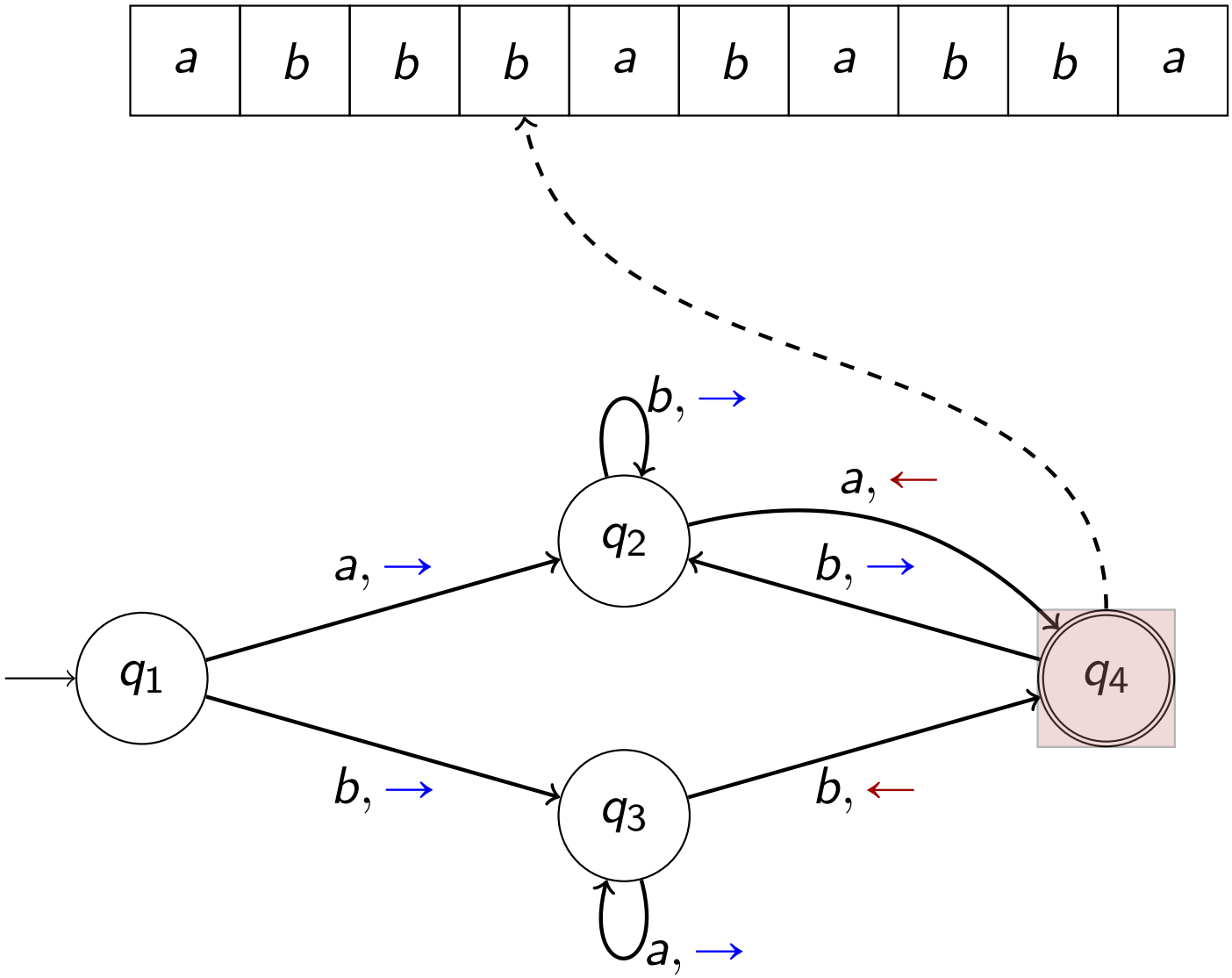
Automates *two ways*



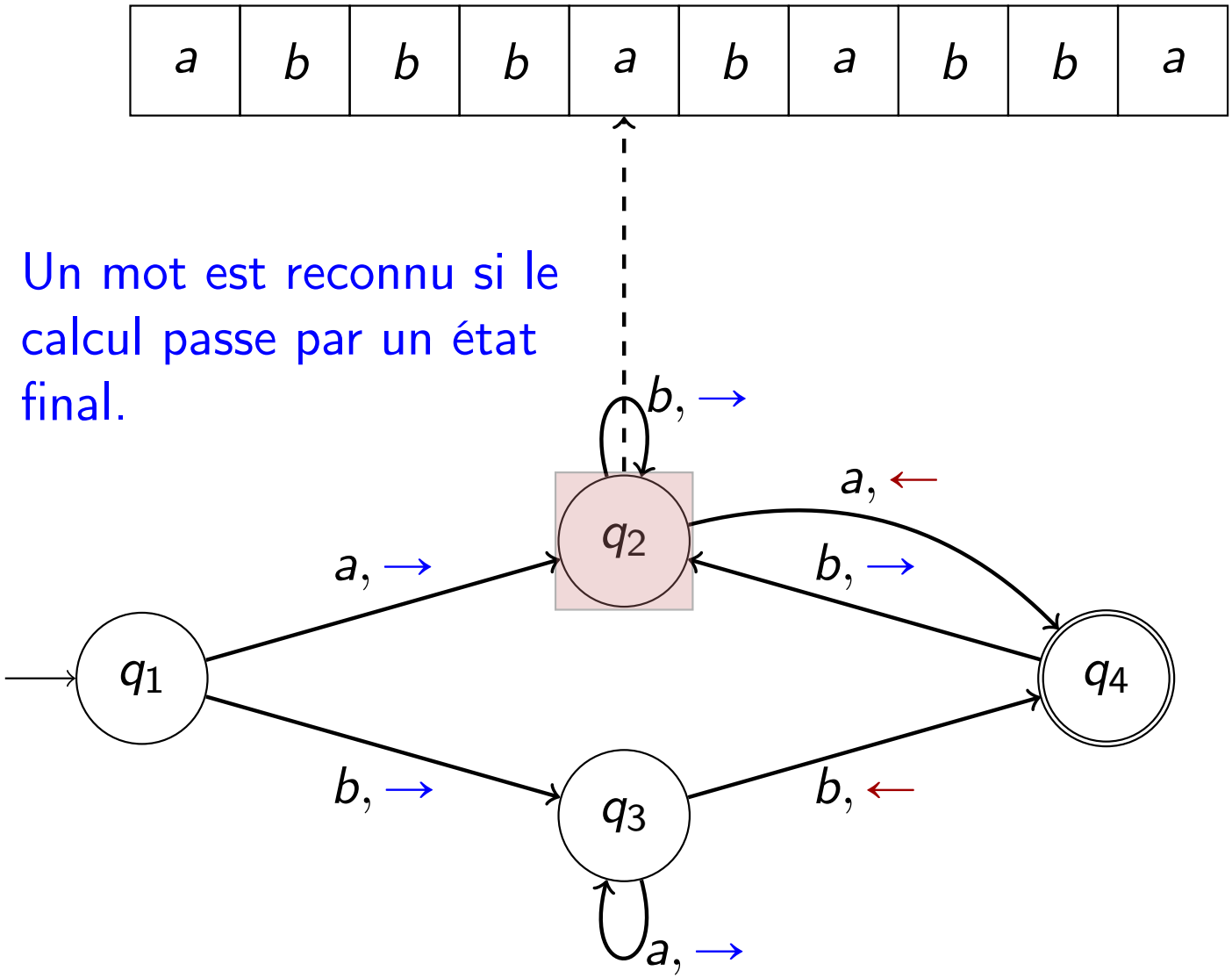
Automates *two ways*



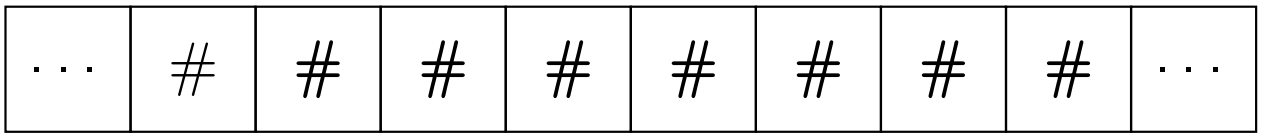
Automates *two ways*



Automates *two ways*

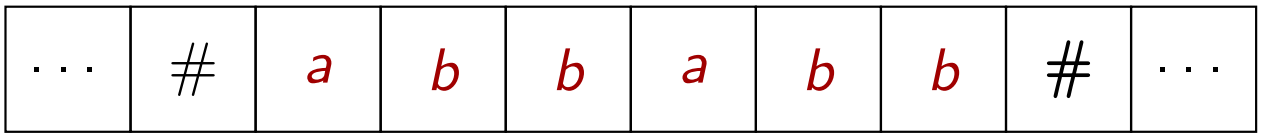


Machine de Turing, exemple 1



On dispose d'un ruban infini (des deux cotés), et d'un symbole pour le vide, ici #.

Machine de Turing, exemple 1

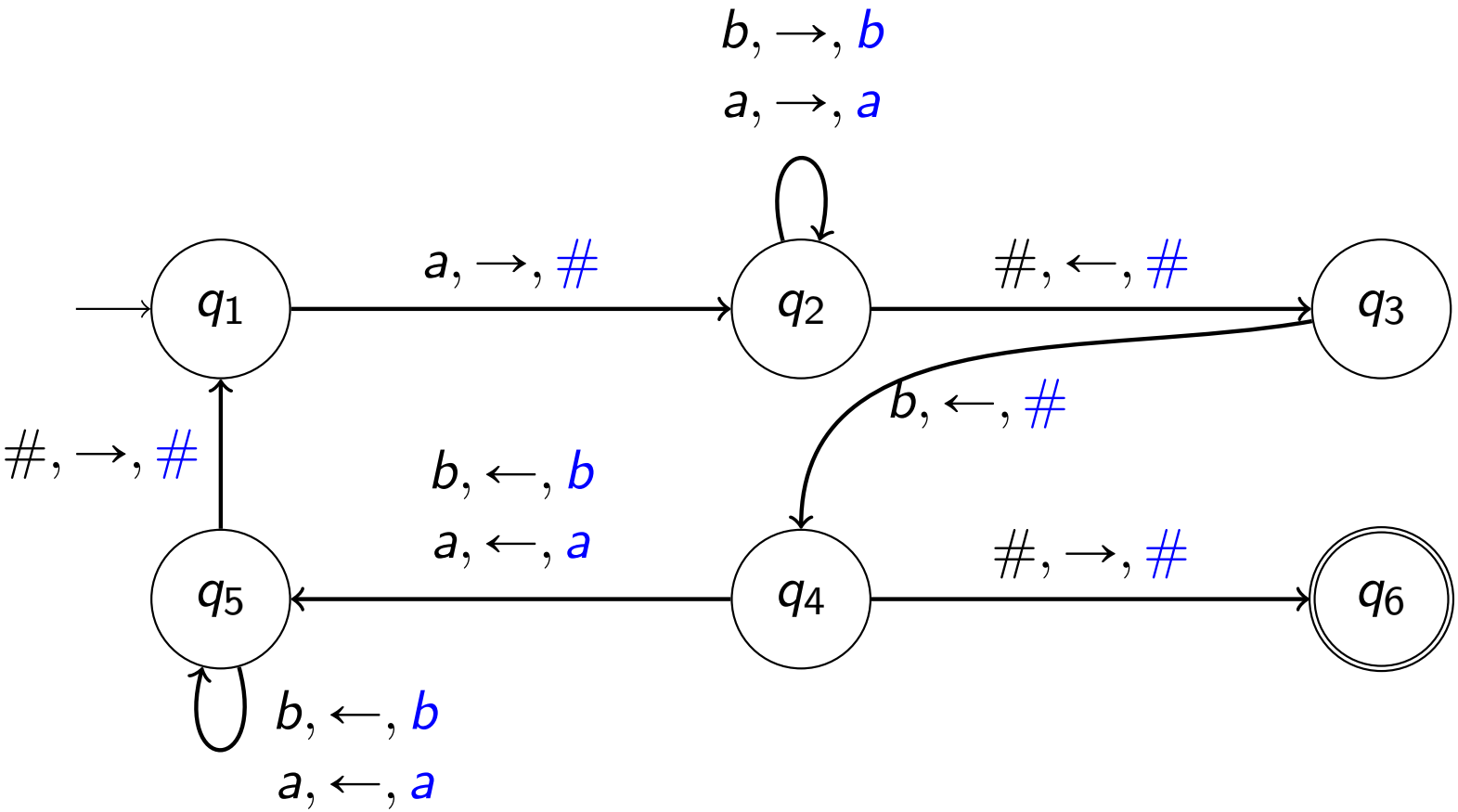


On dispose d'un ruban infini (des deux cotés), et d'un symbole pour le vide, ici #.

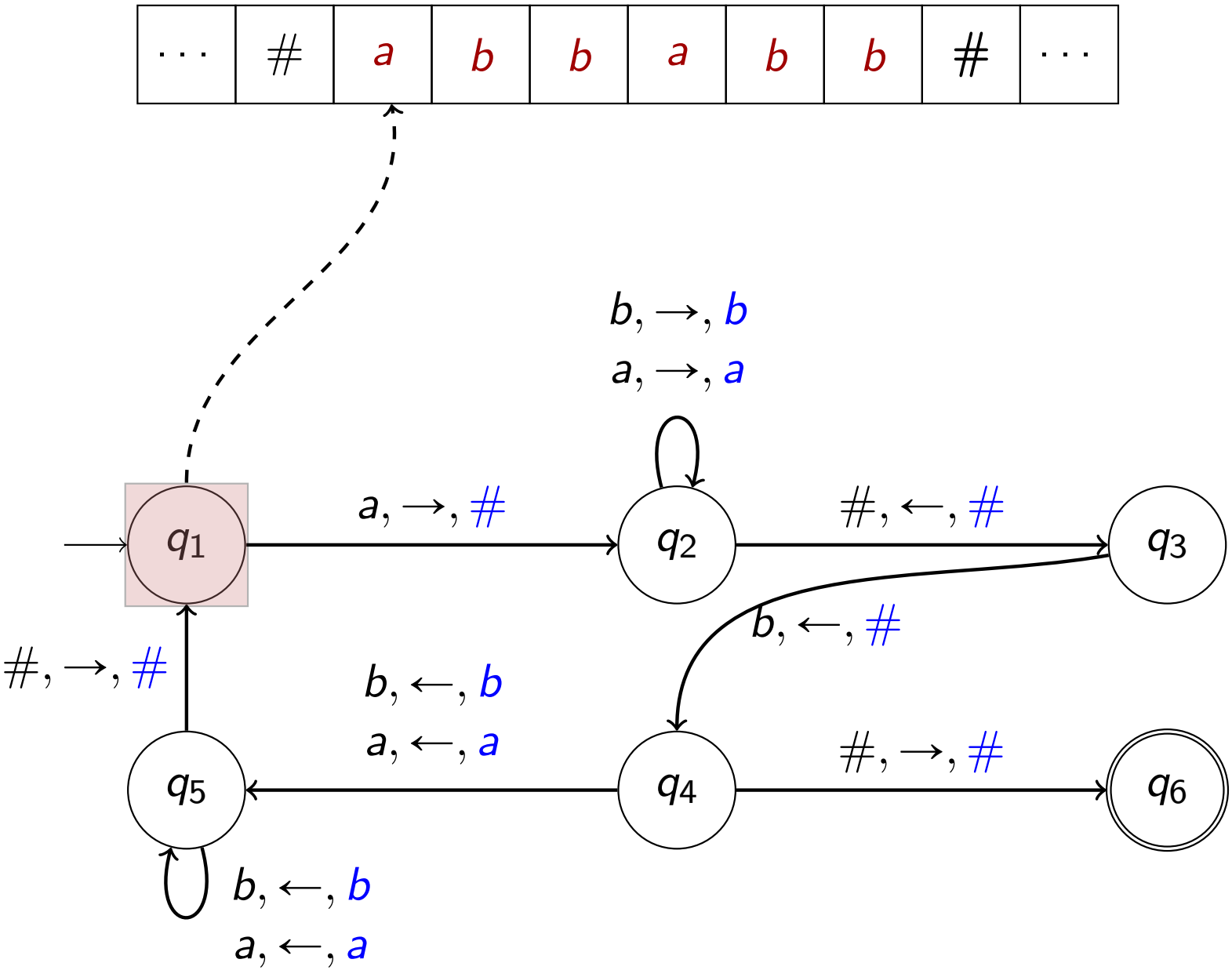
On écrit le mot d'entrée sur le ruban.

Machine de Turing, exemple 1

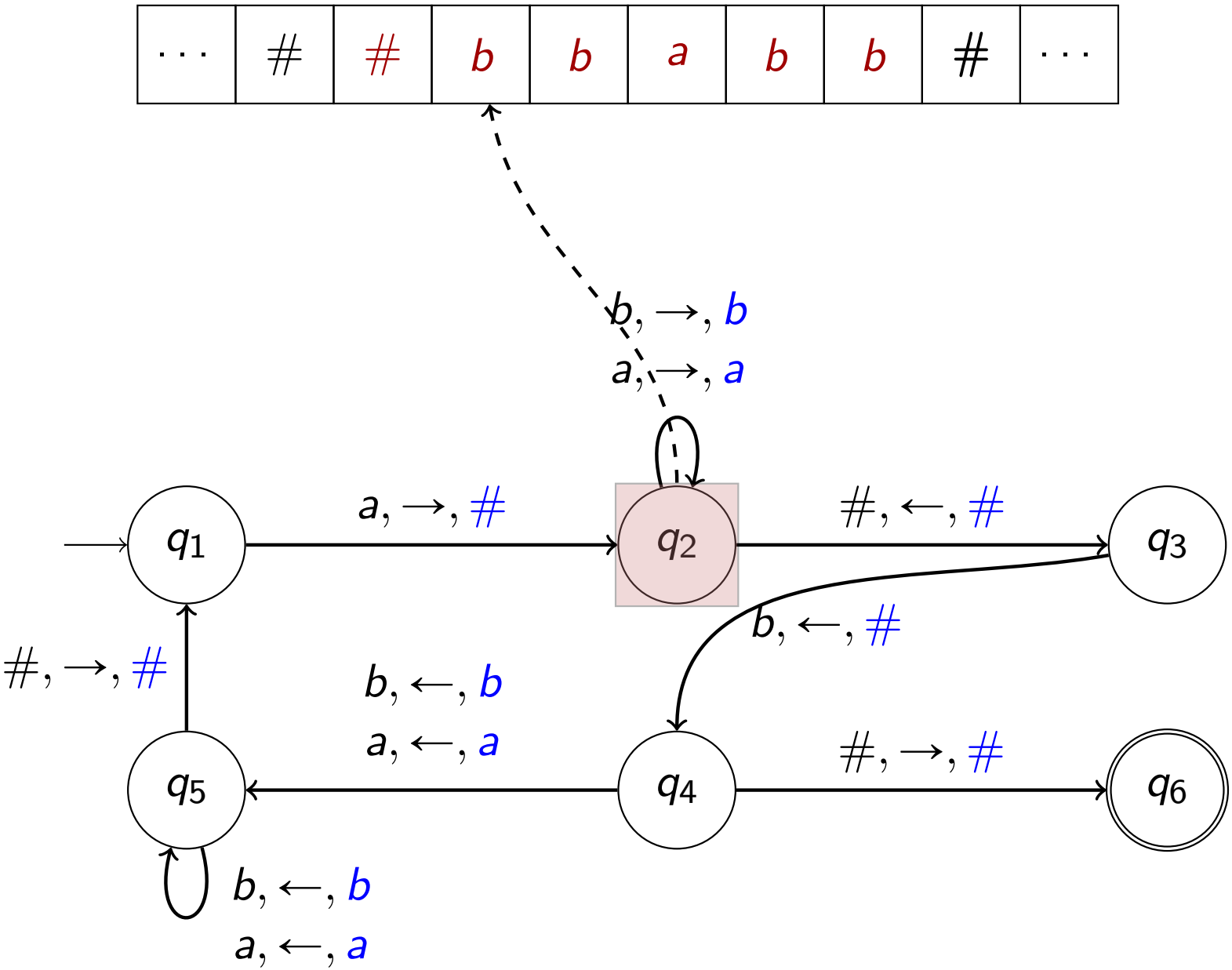
$\cdots$	#	a	b	b	a	b	b	#	$\cdots$



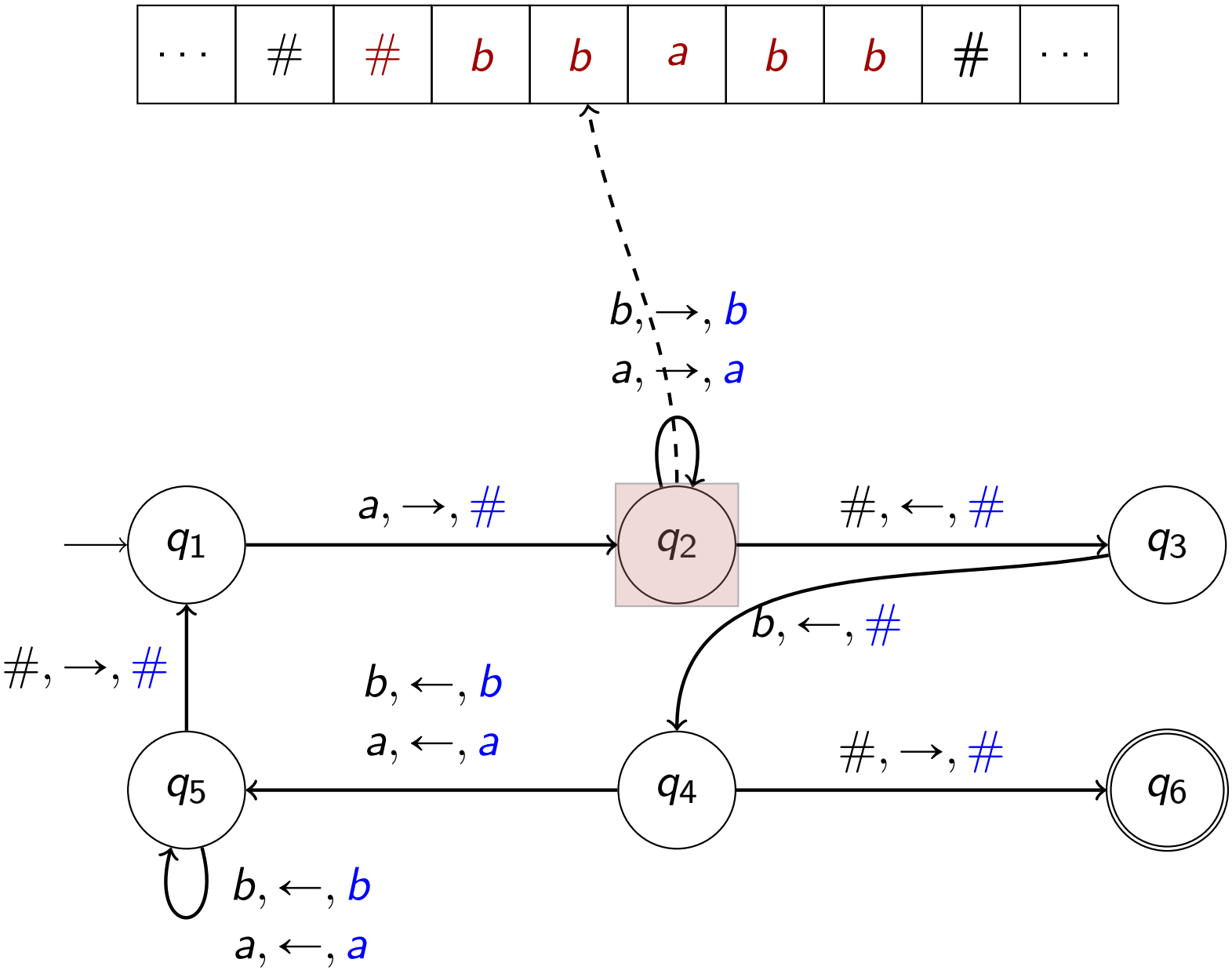
Machine de Turing, exemple 1



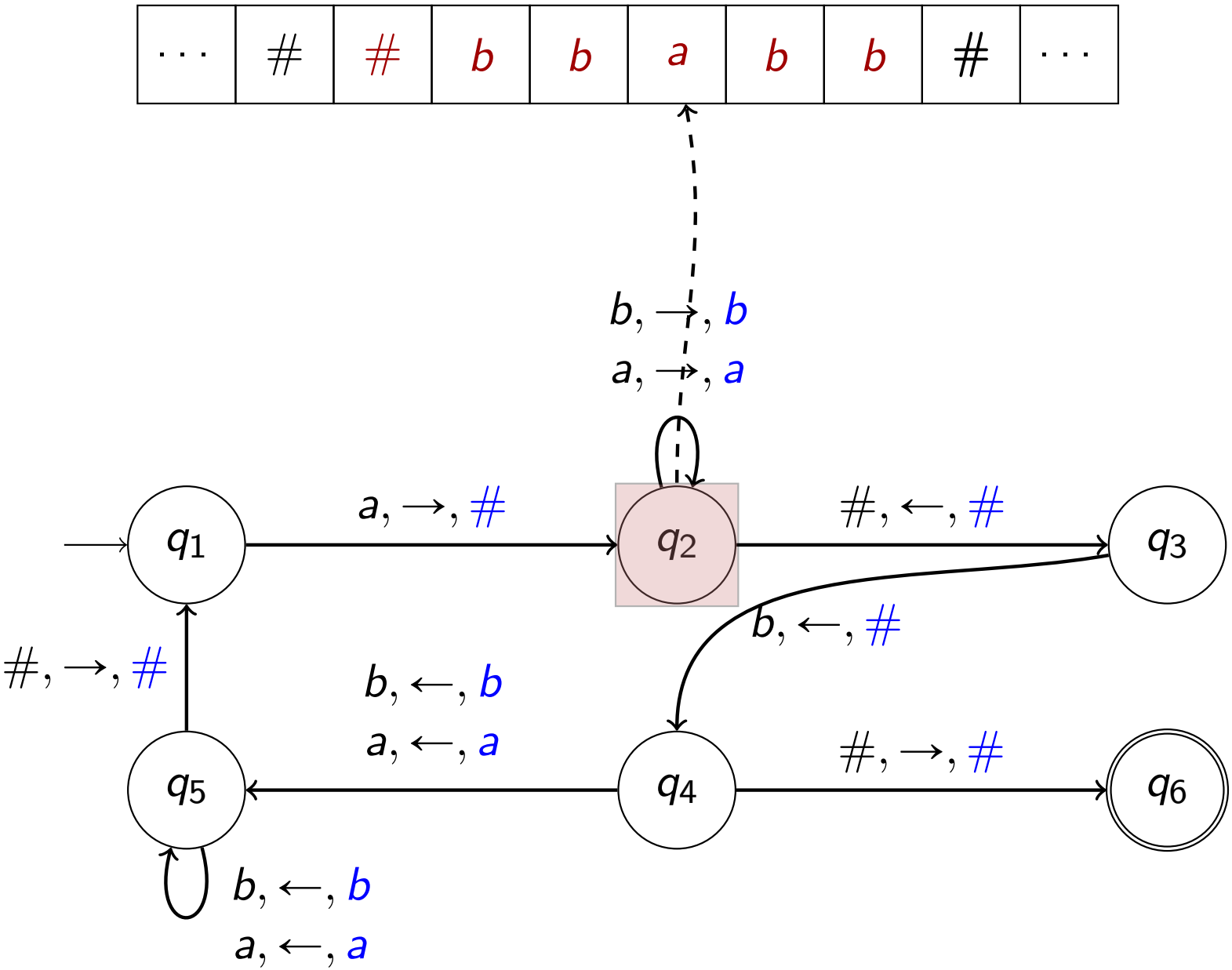
Machine de Turing, exemple 1



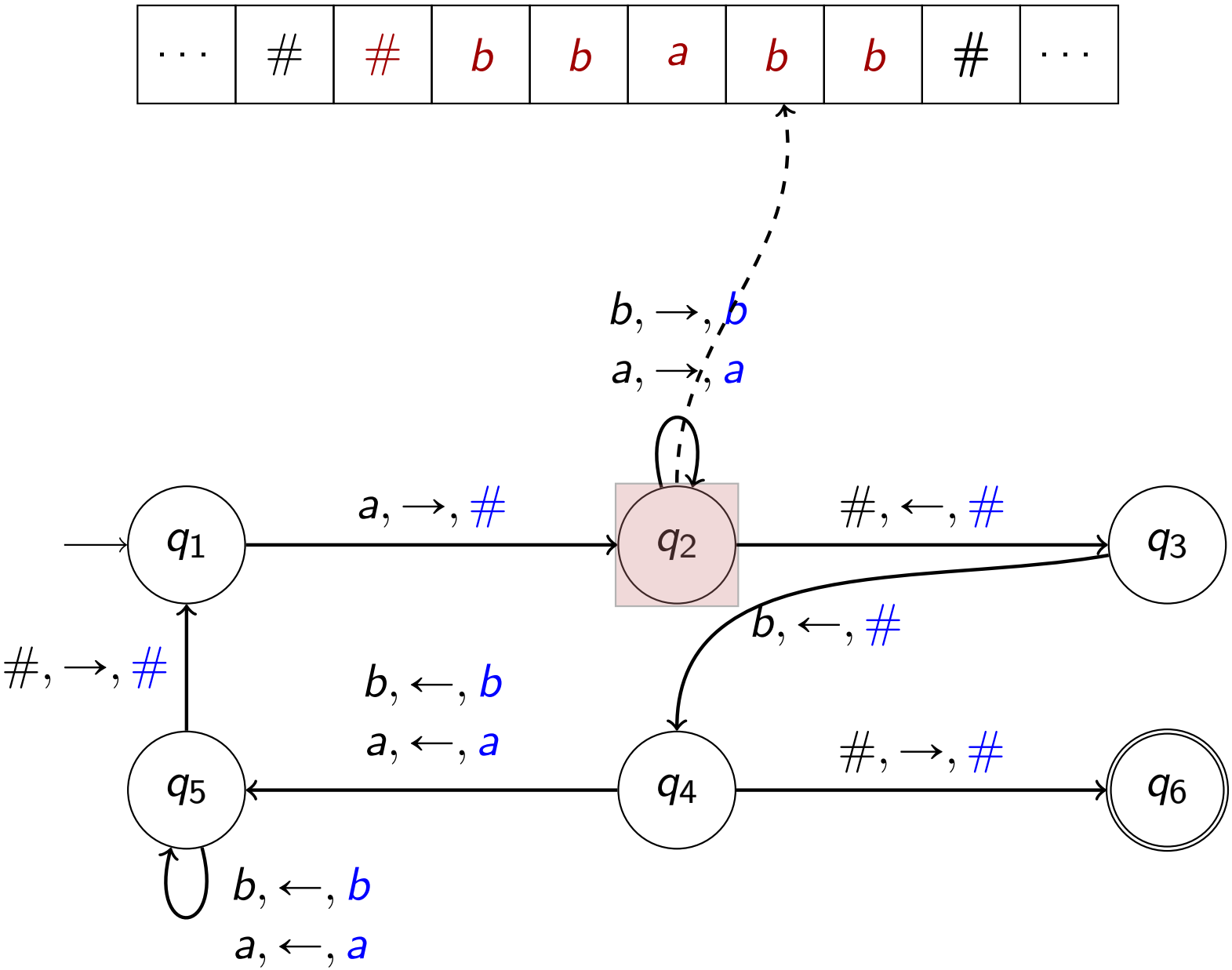
Machine de Turing, exemple 1



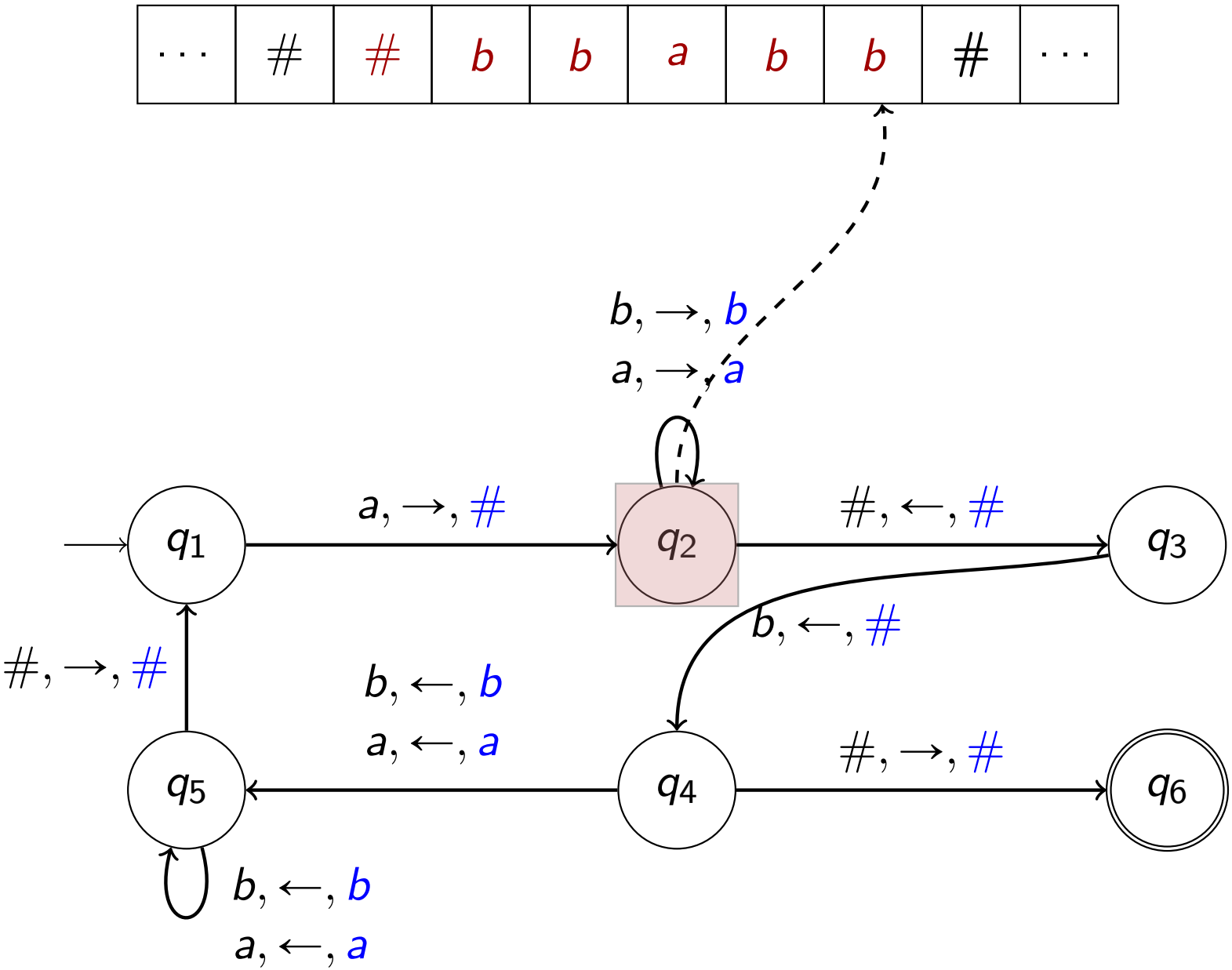
Machine de Turing, exemple 1



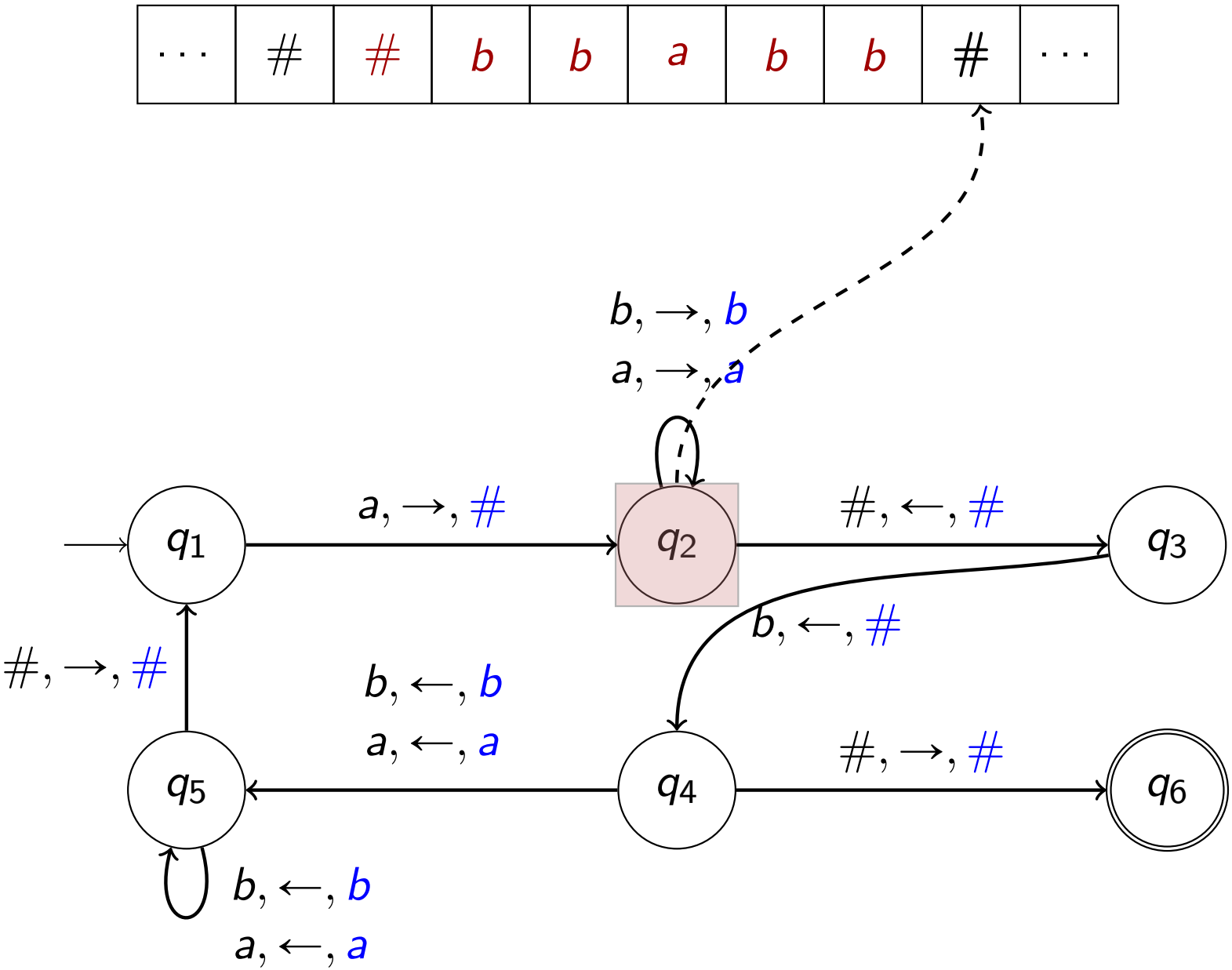
Machine de Turing, exemple 1



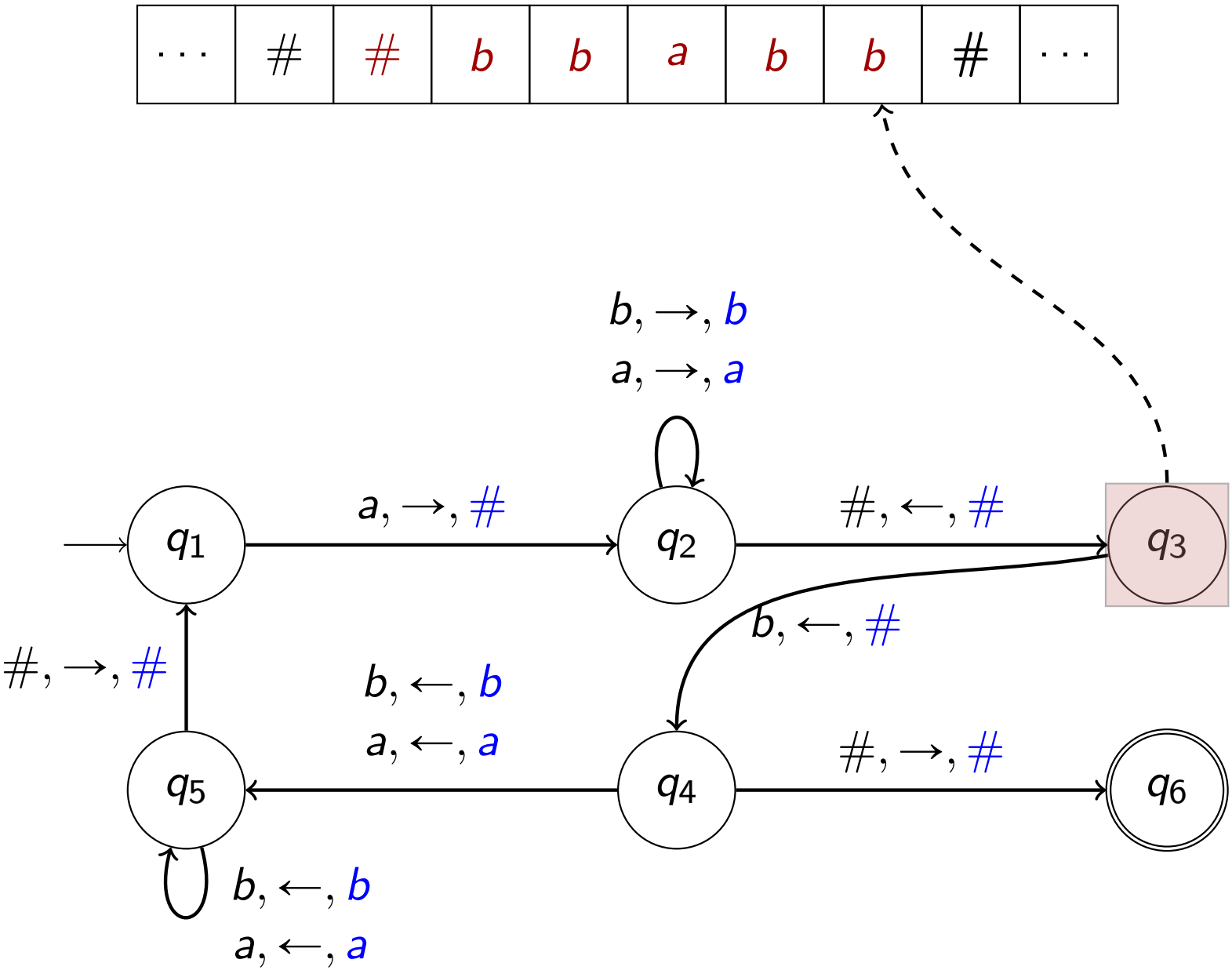
Machine de Turing, exemple 1



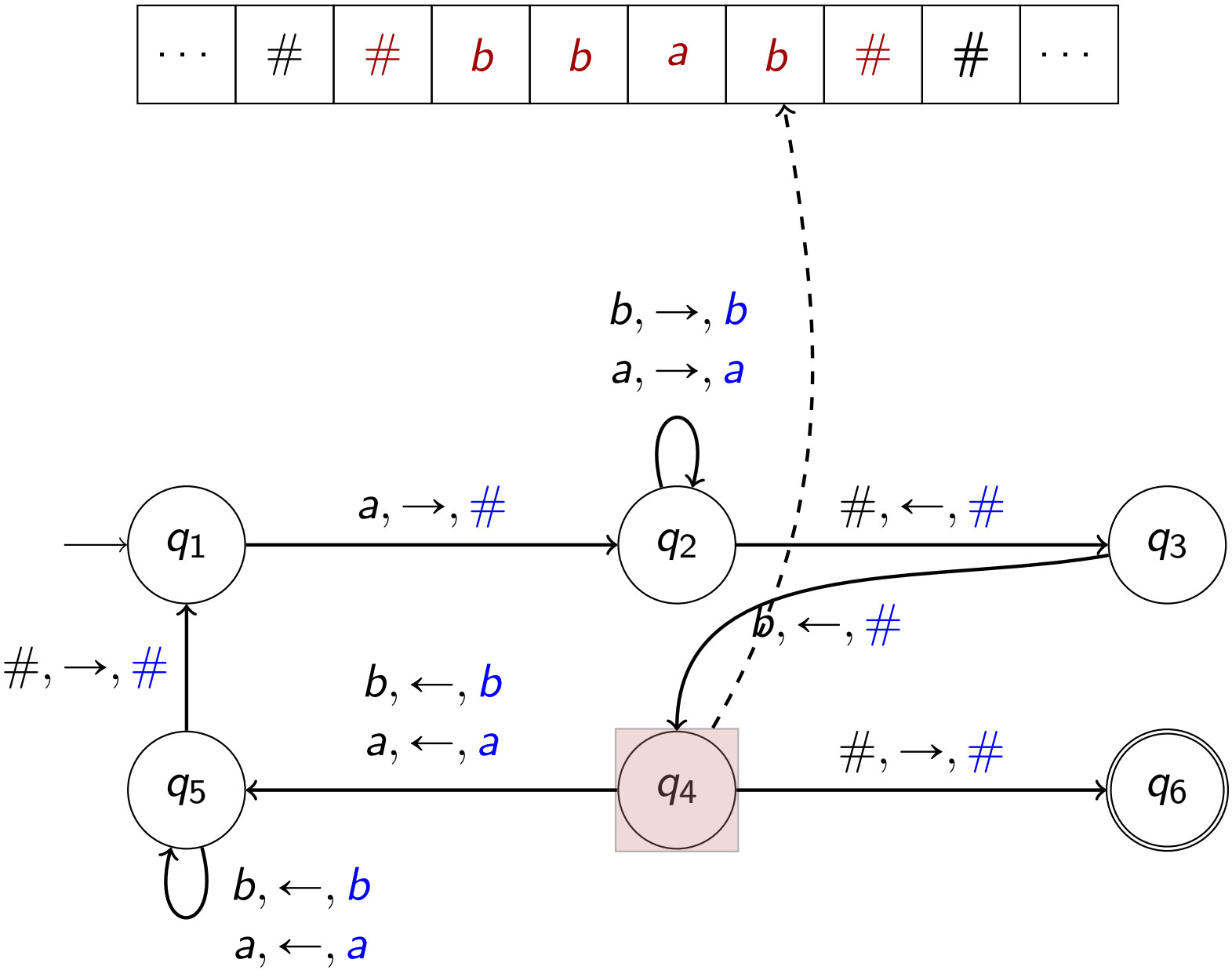
Machine de Turing, exemple 1



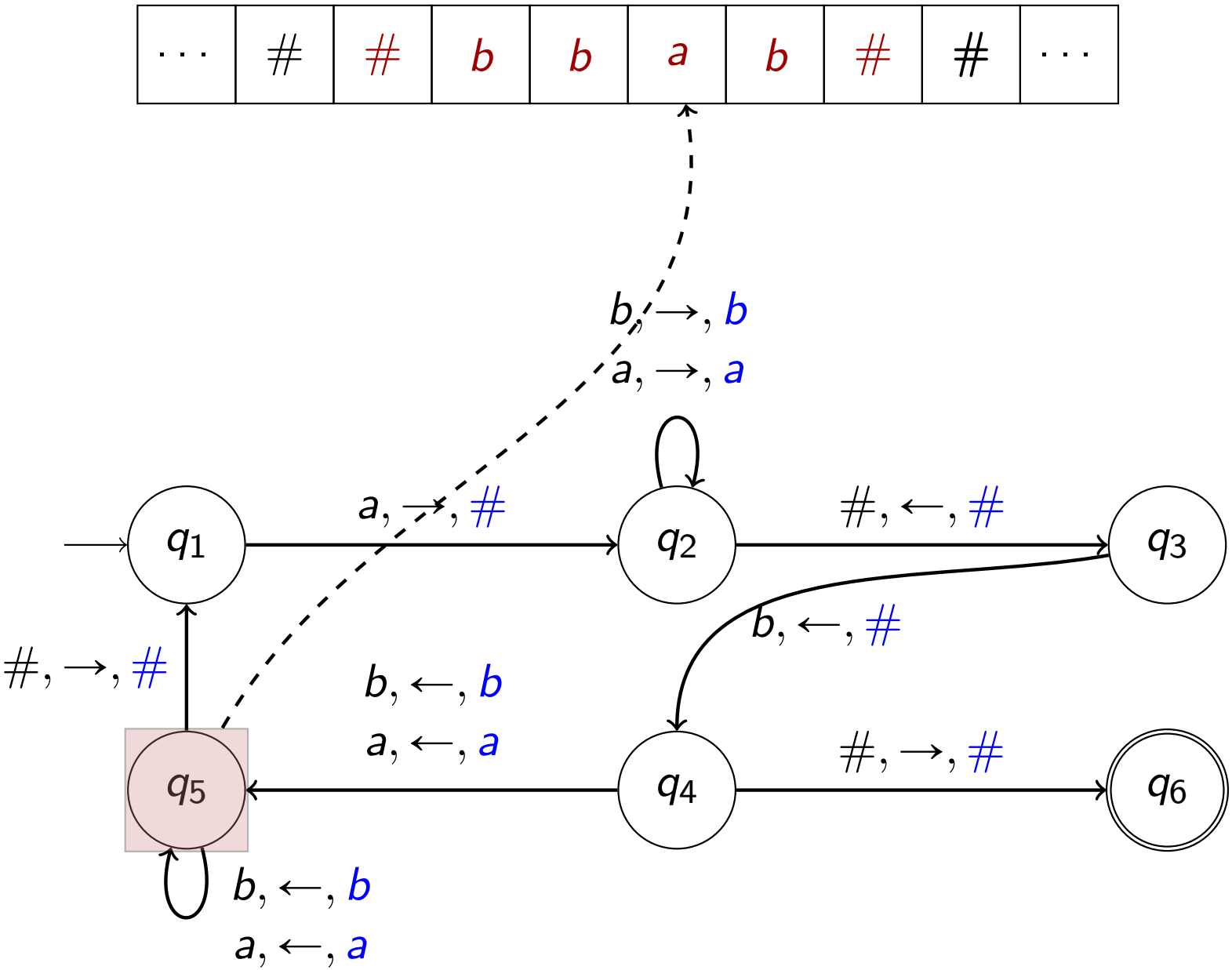
Machine de Turing, exemple 1



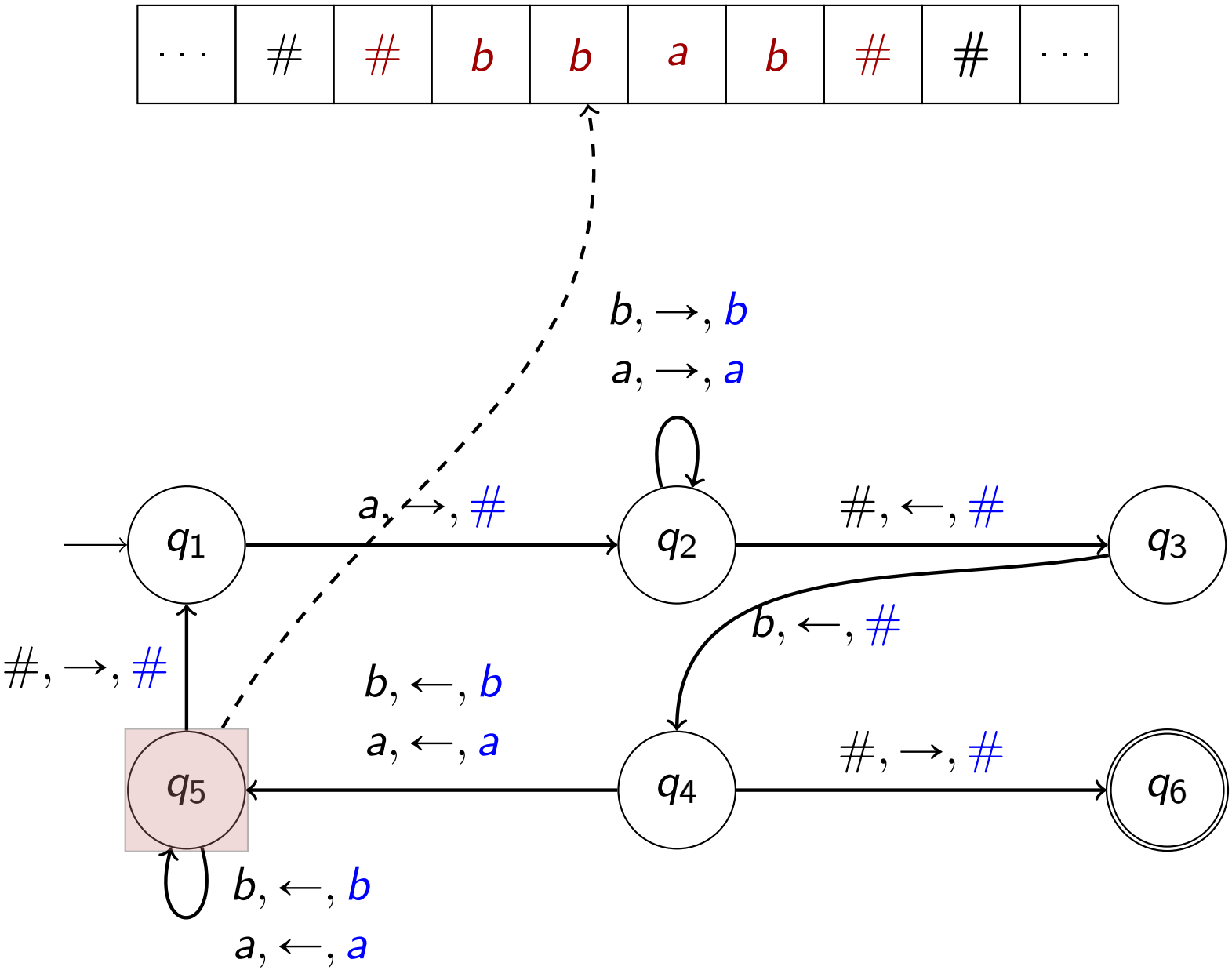
Machine de Turing, exemple 1



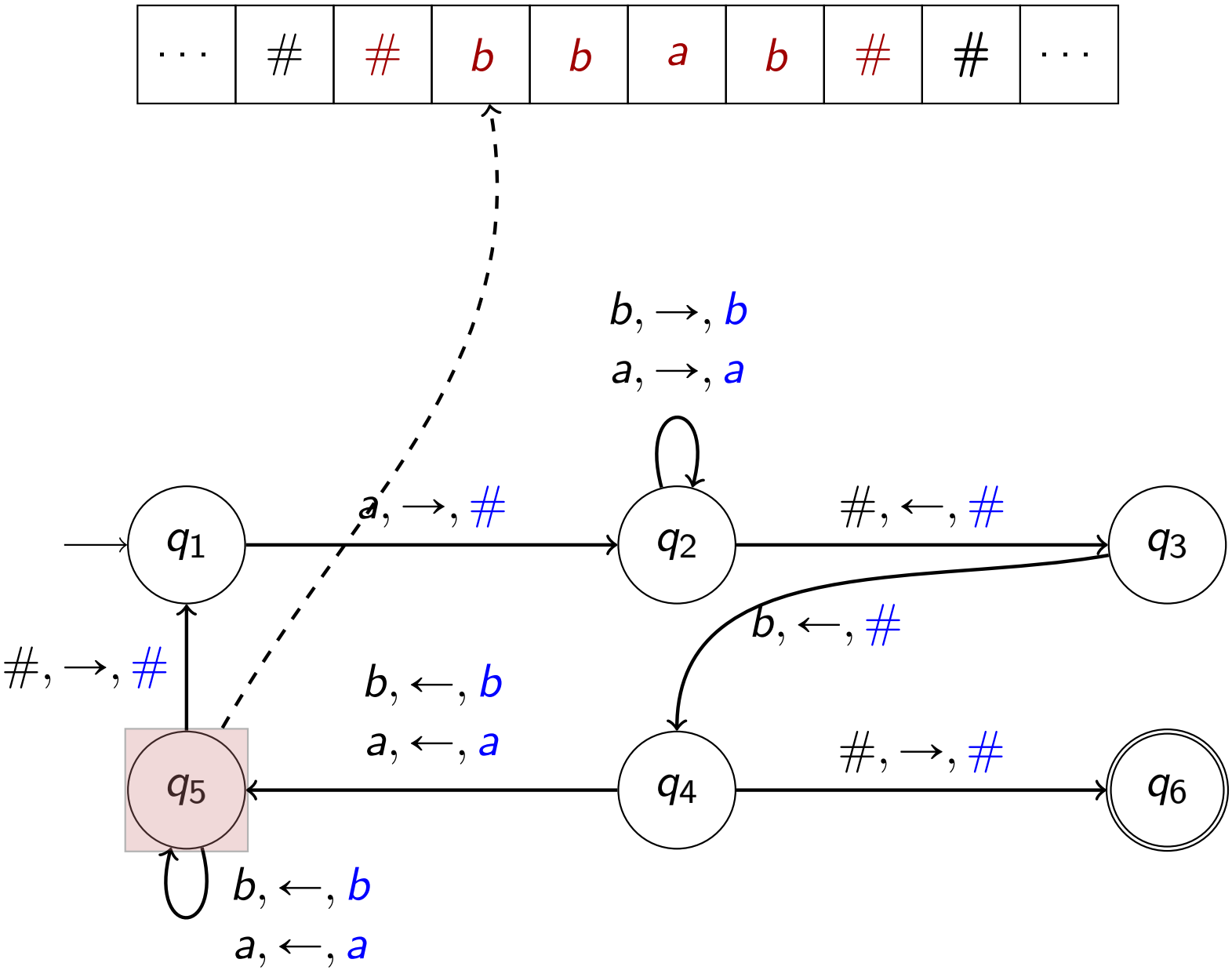
Machine de Turing, exemple 1



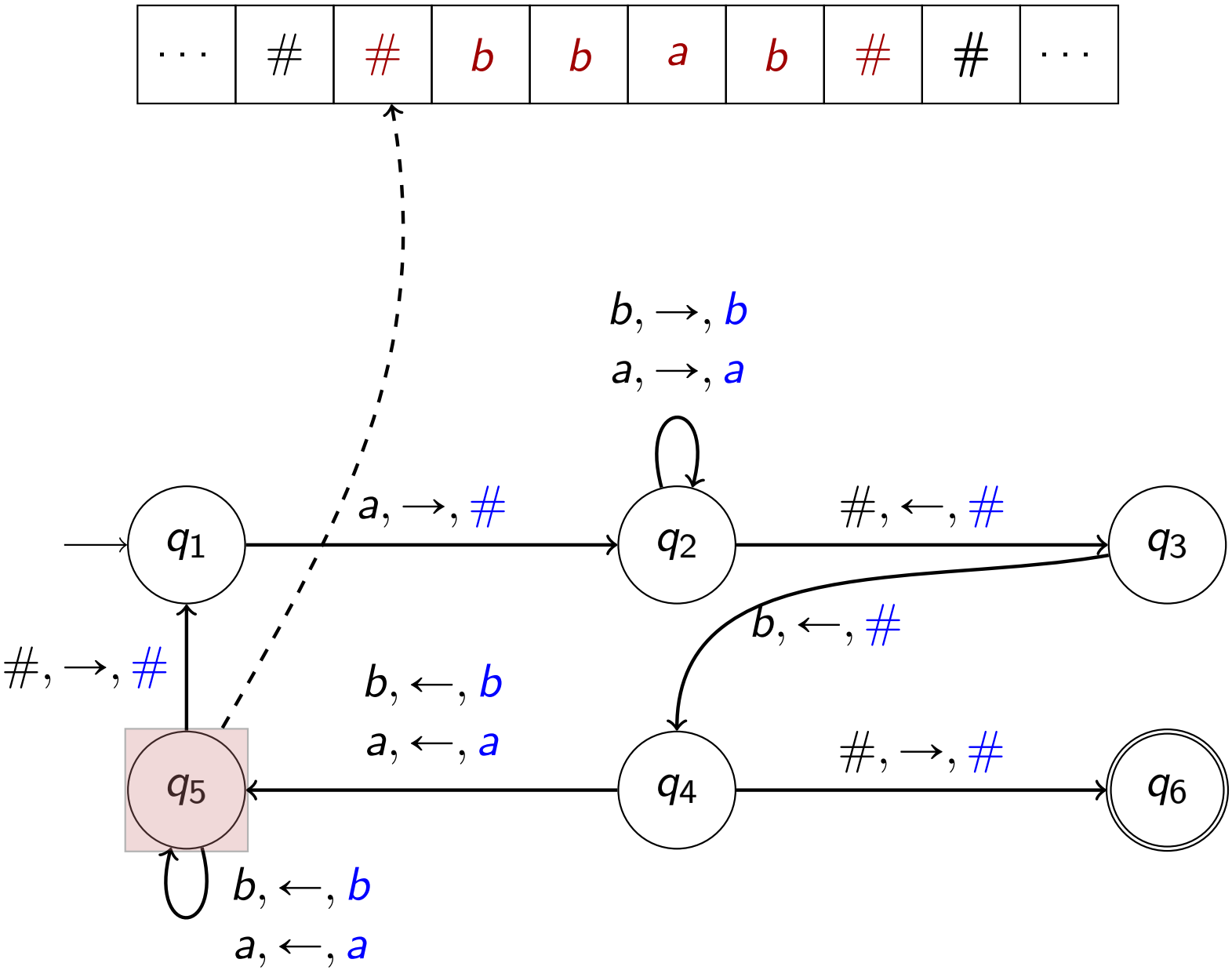
Machine de Turing, exemple 1



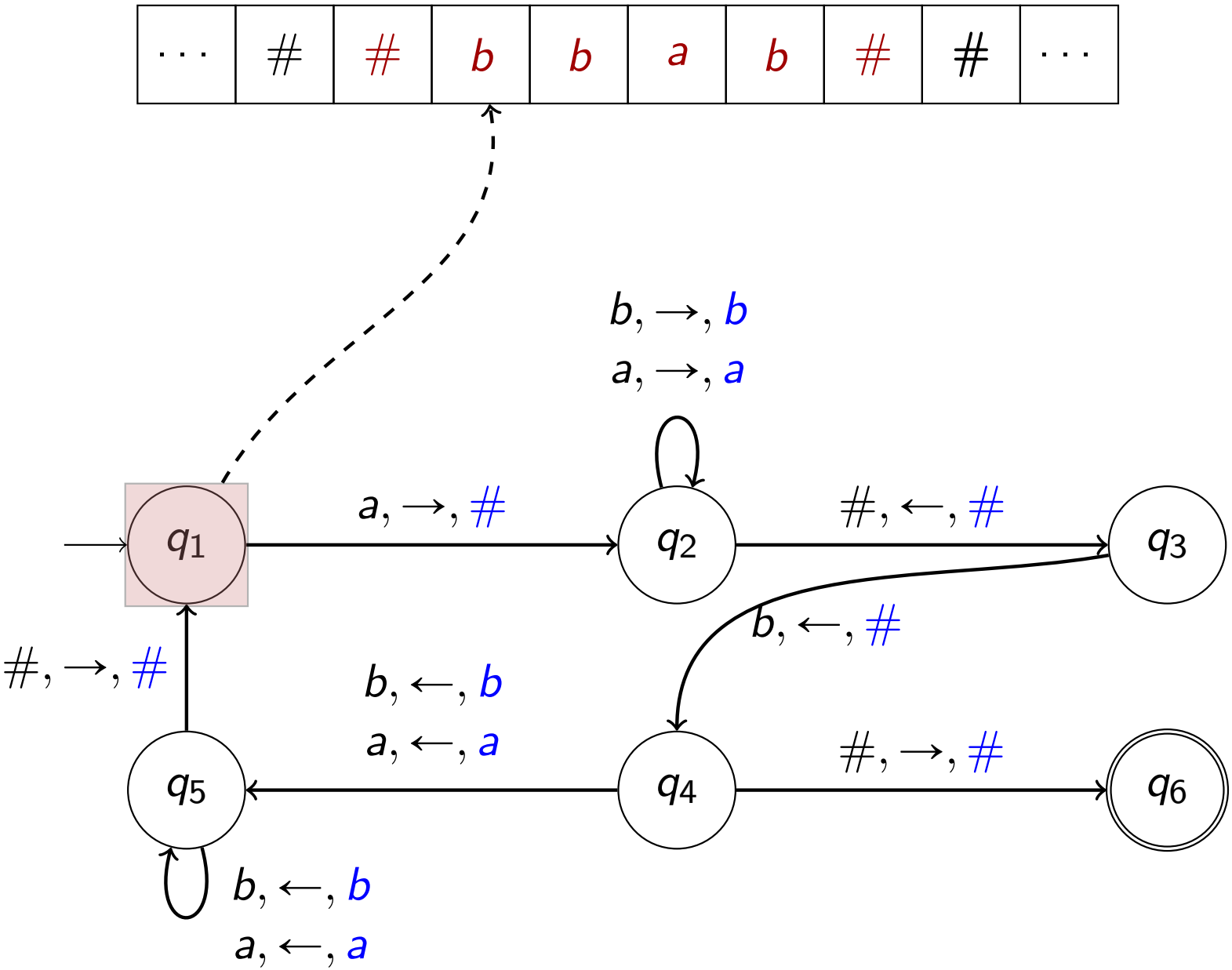
Machine de Turing, exemple 1



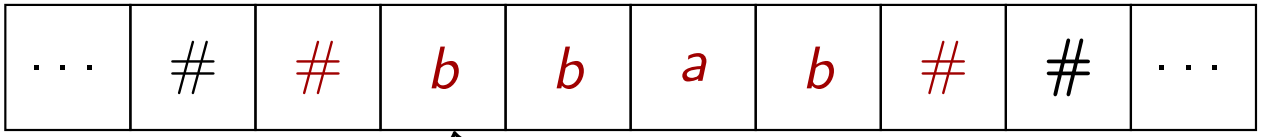
Machine de Turing, exemple 1



Machine de Turing, exemple 1



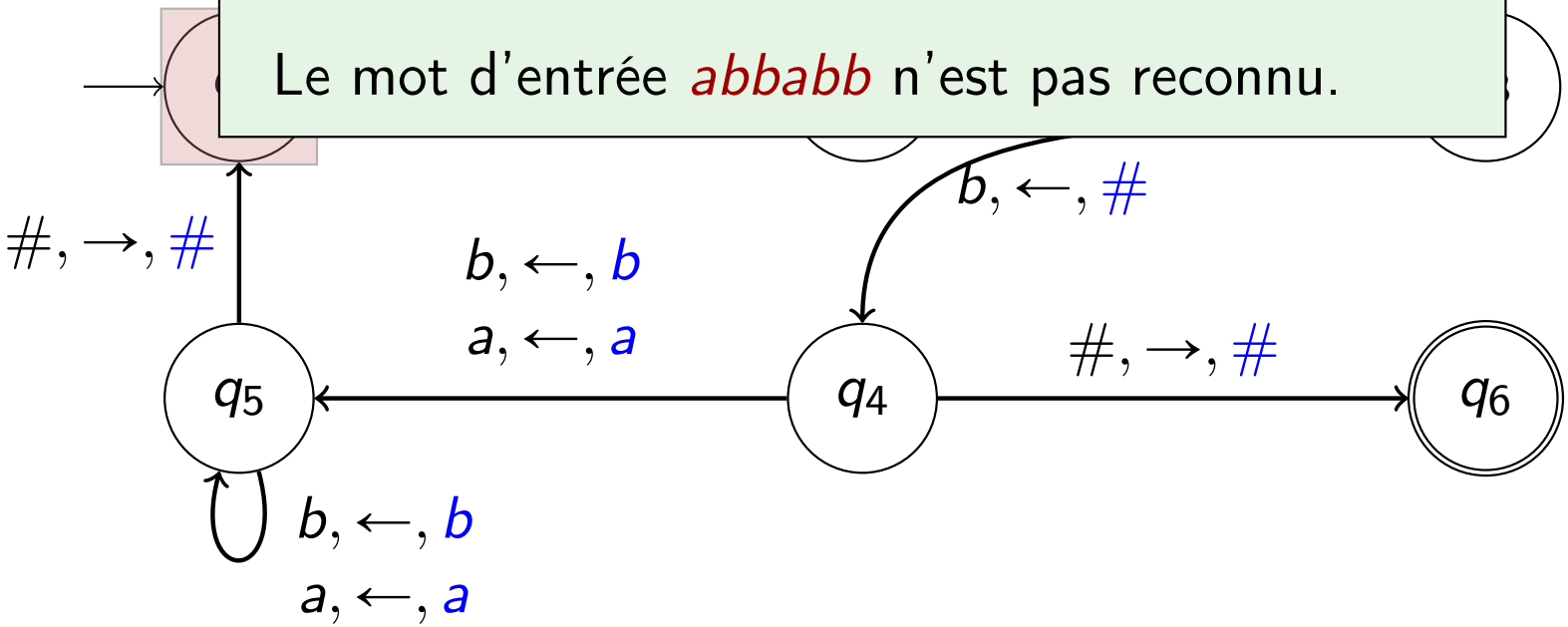
Machine de Turing, exemple 1



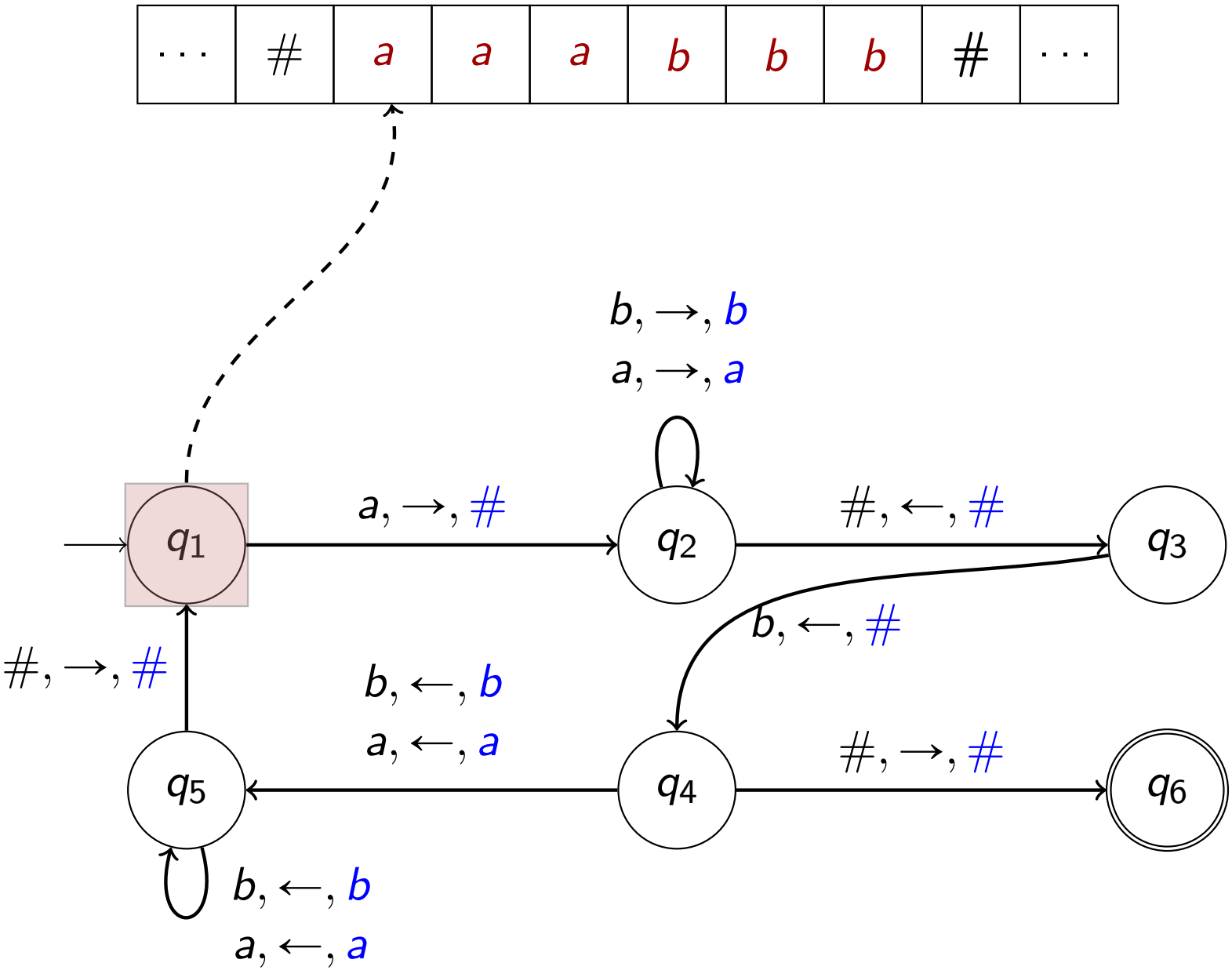
Le calcul s'arrete (on ne peut pas lire #)

On n'est pas passé par un état final

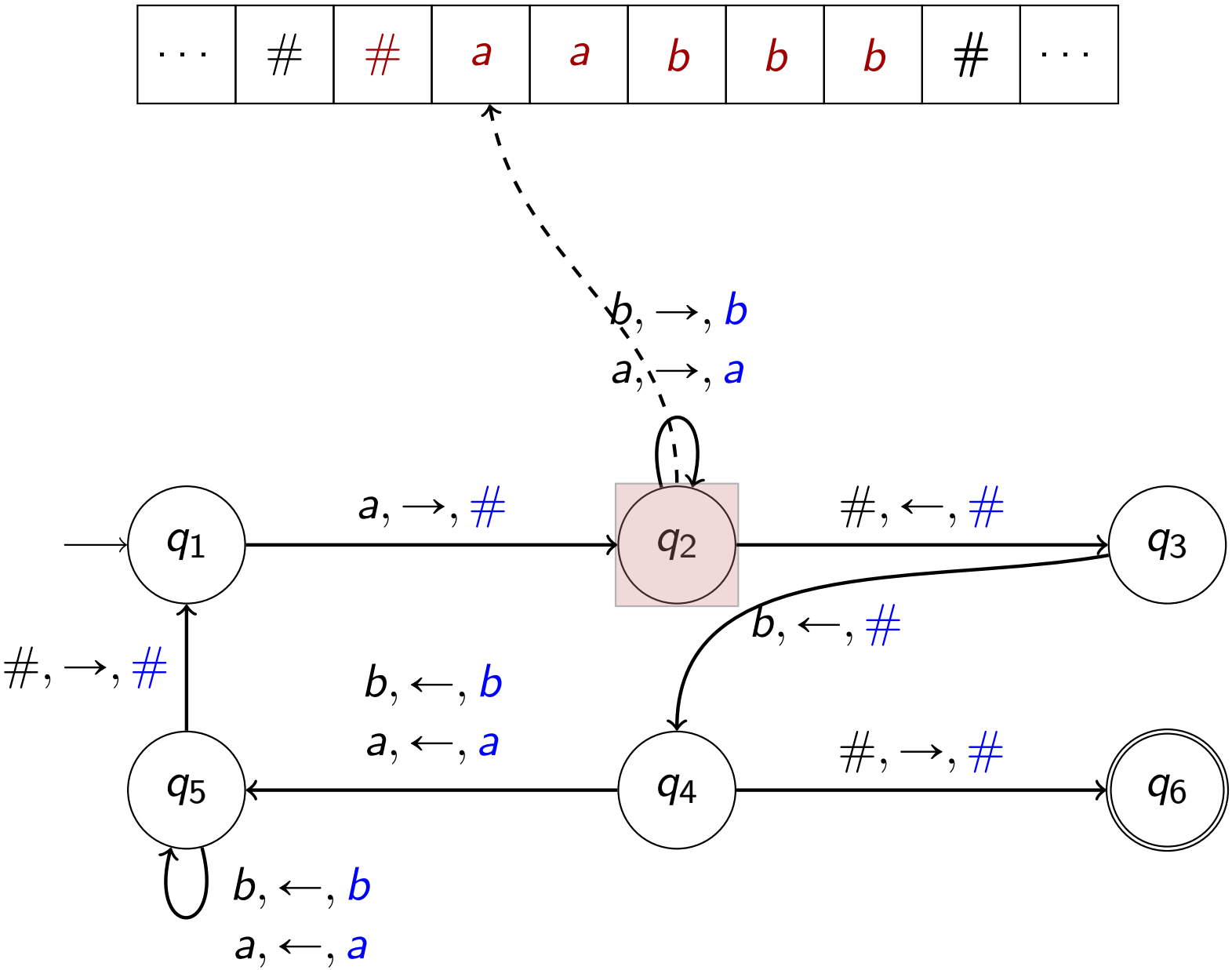
Le mot d'entrée *abbabb* n'est pas reconnu.



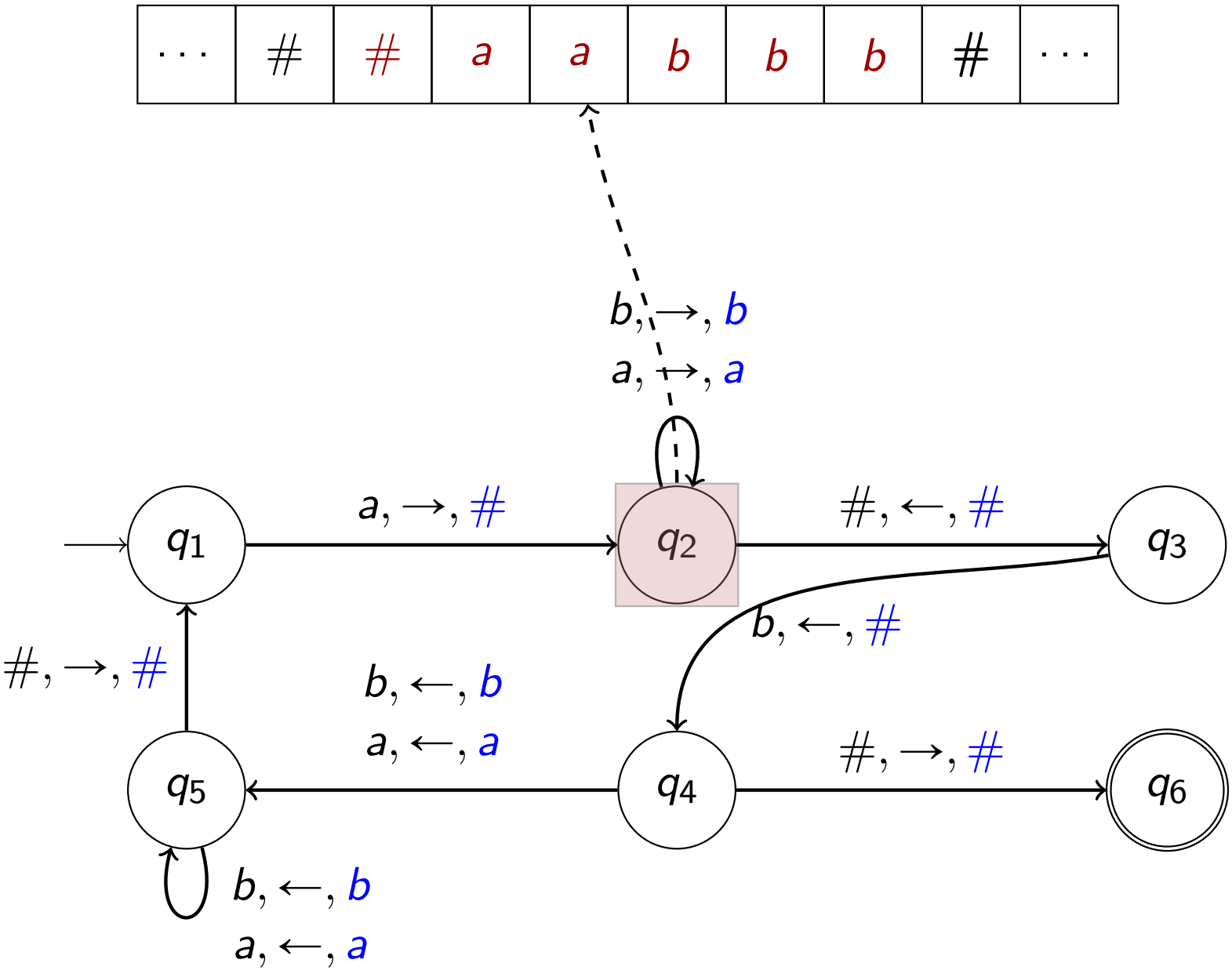
Machine de Turing, exemple 2



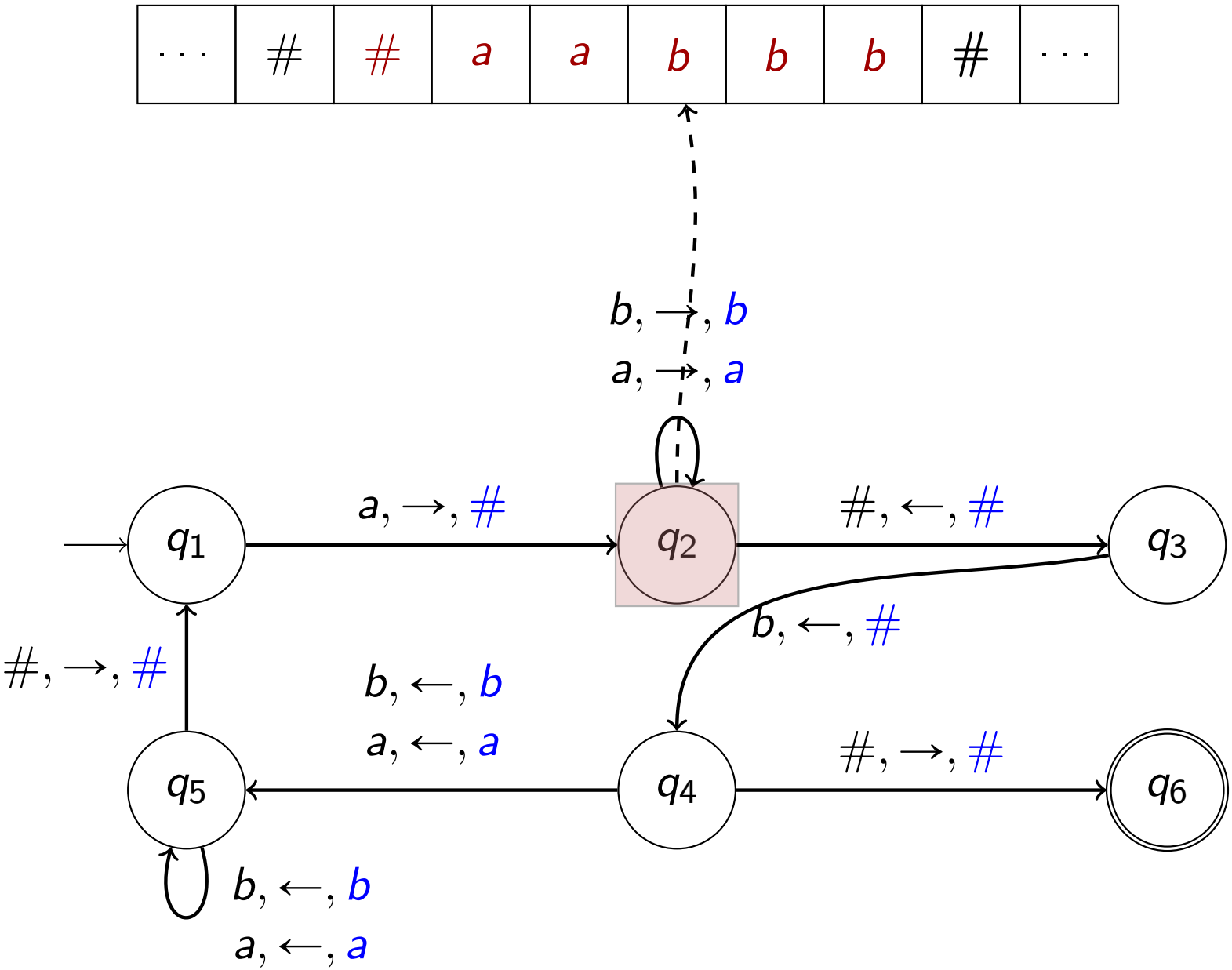
Machine de Turing, exemple 2



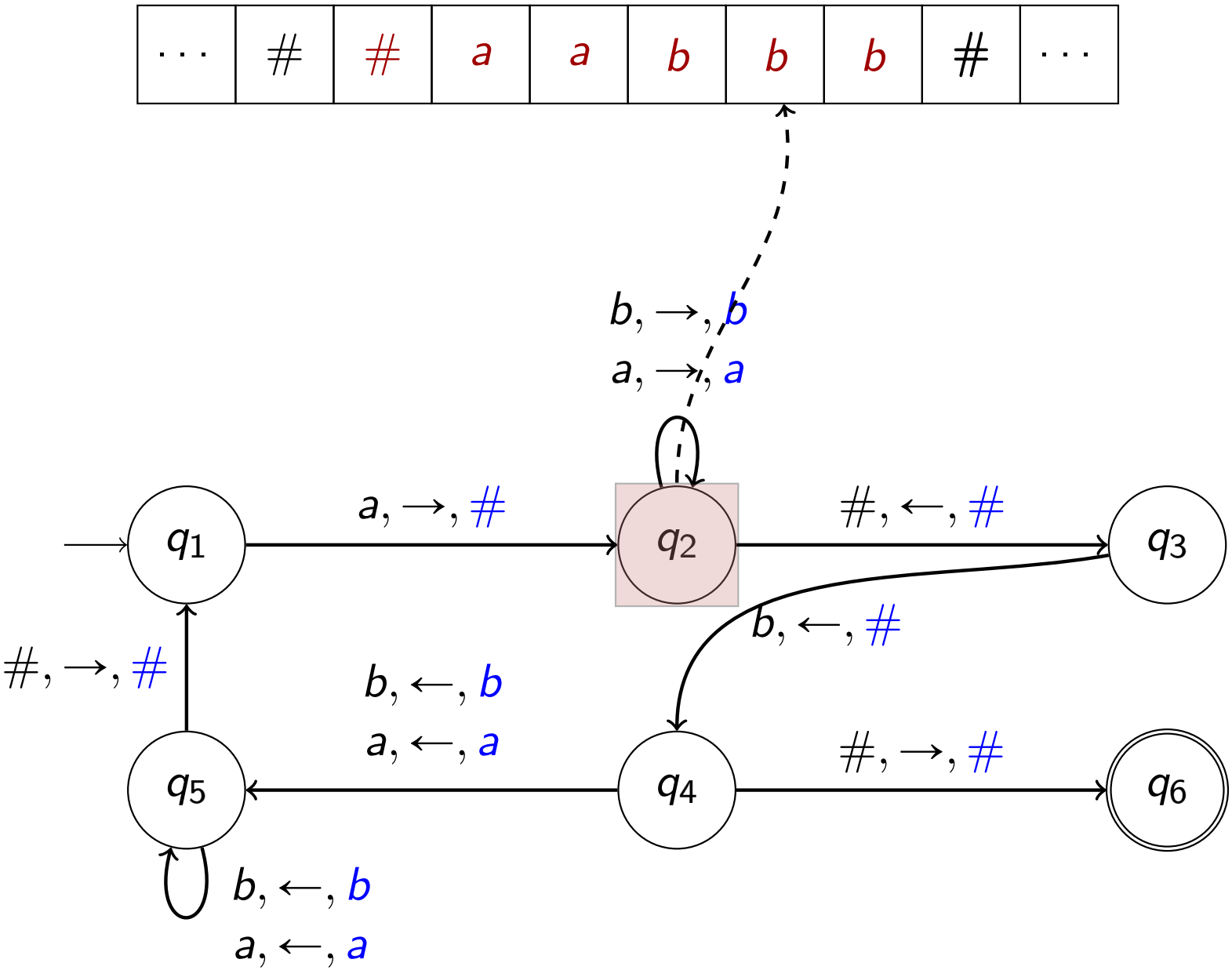
Machine de Turing, exemple 2



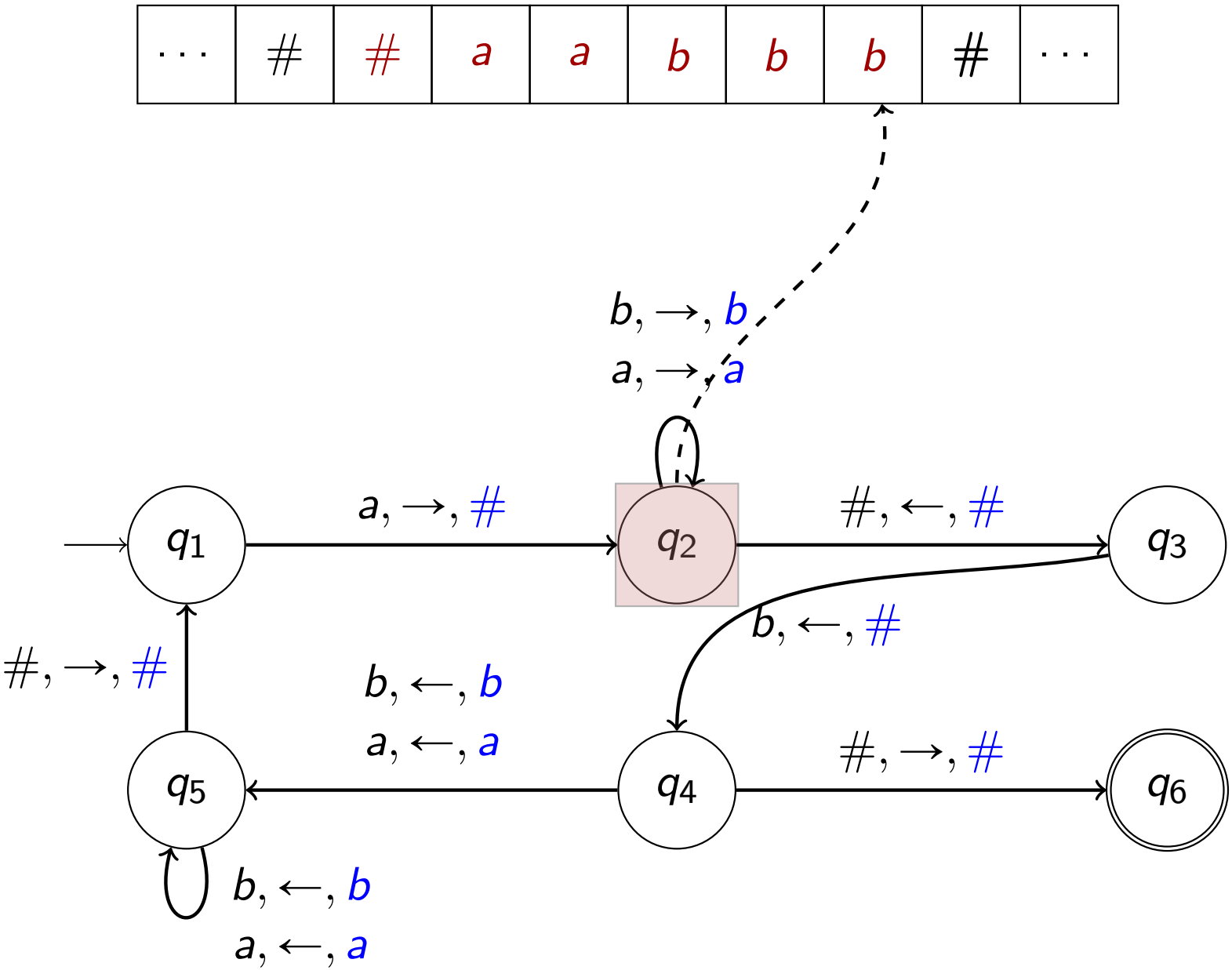
Machine de Turing, exemple 2



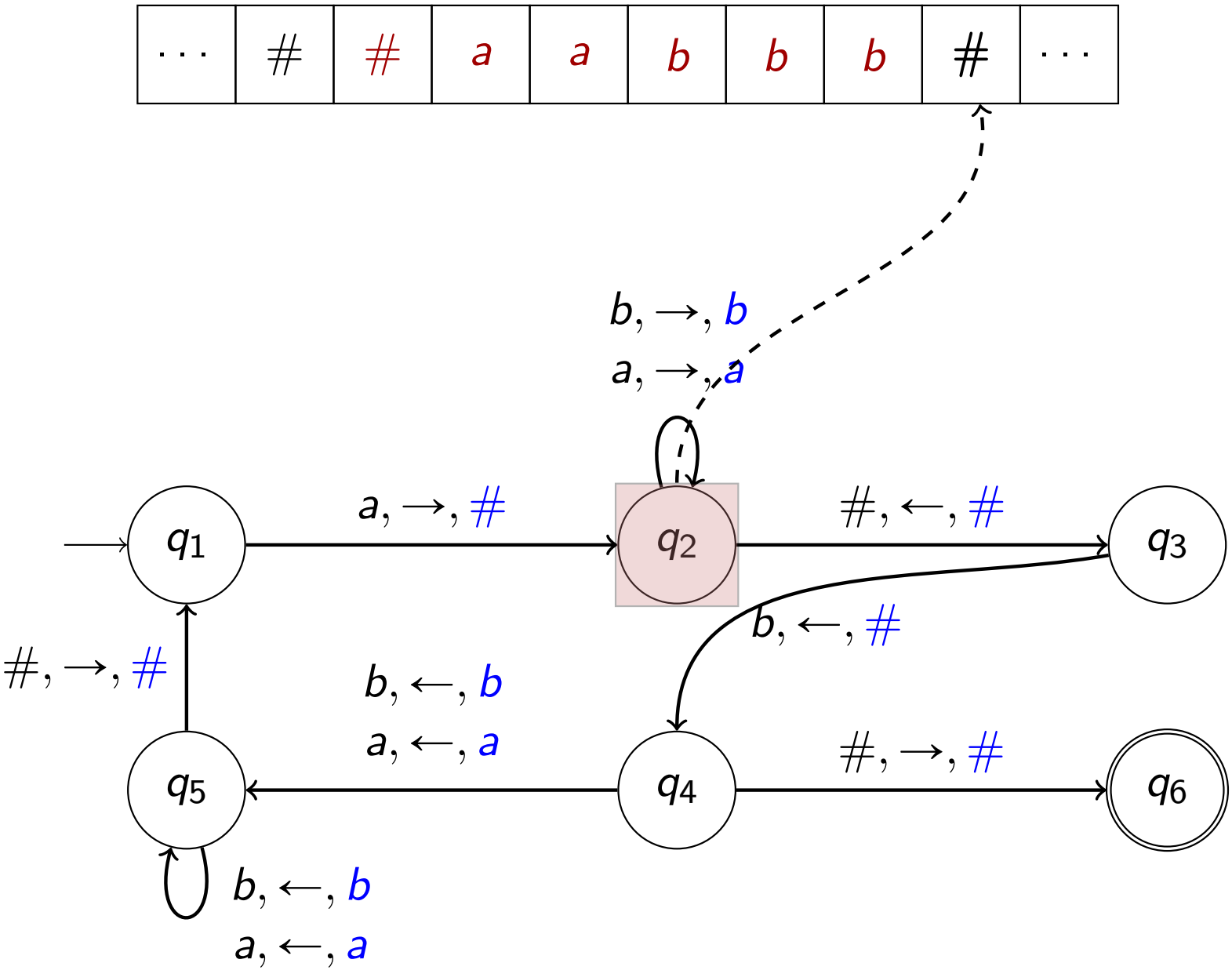
Machine de Turing, exemple 2



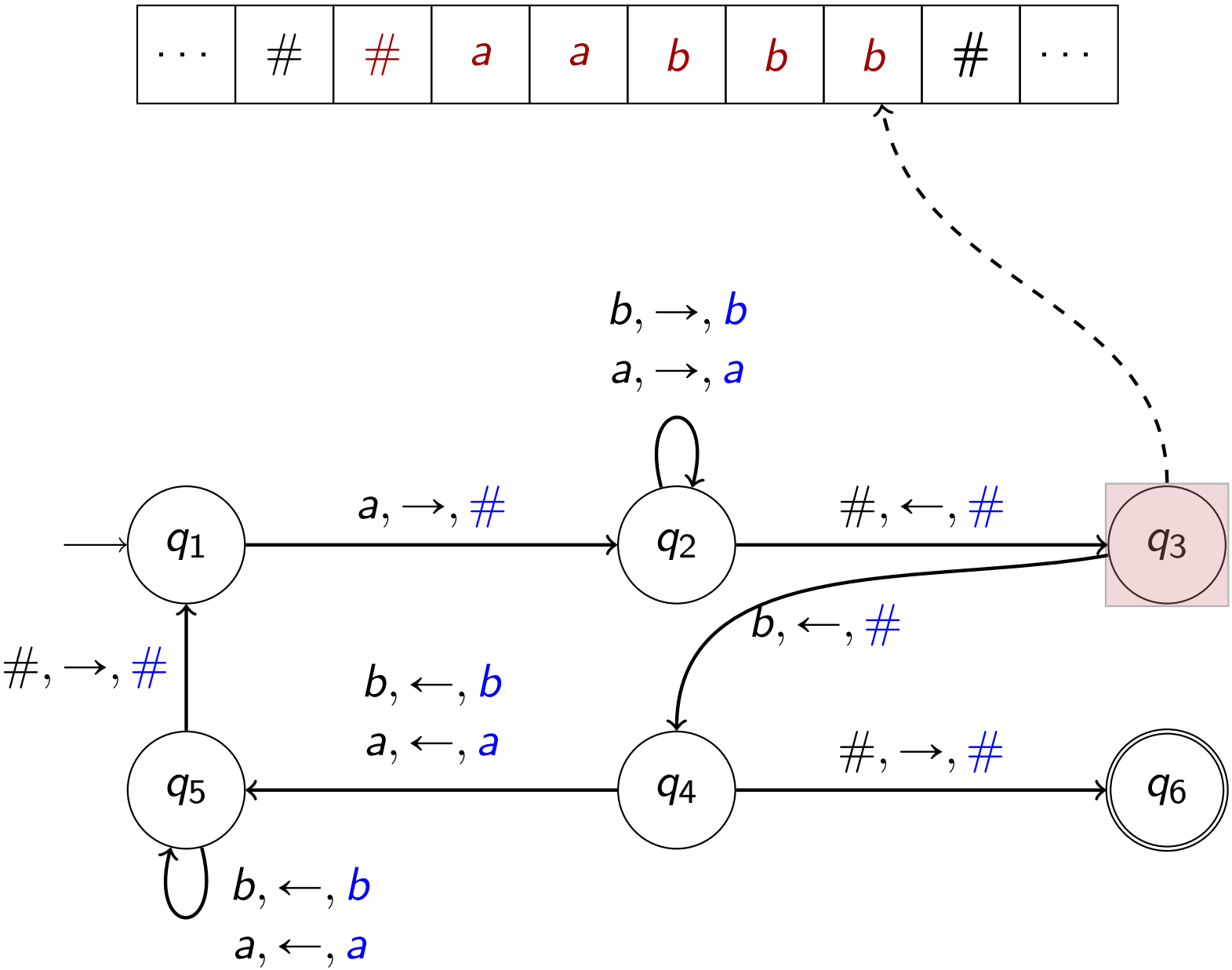
Machine de Turing, exemple 2



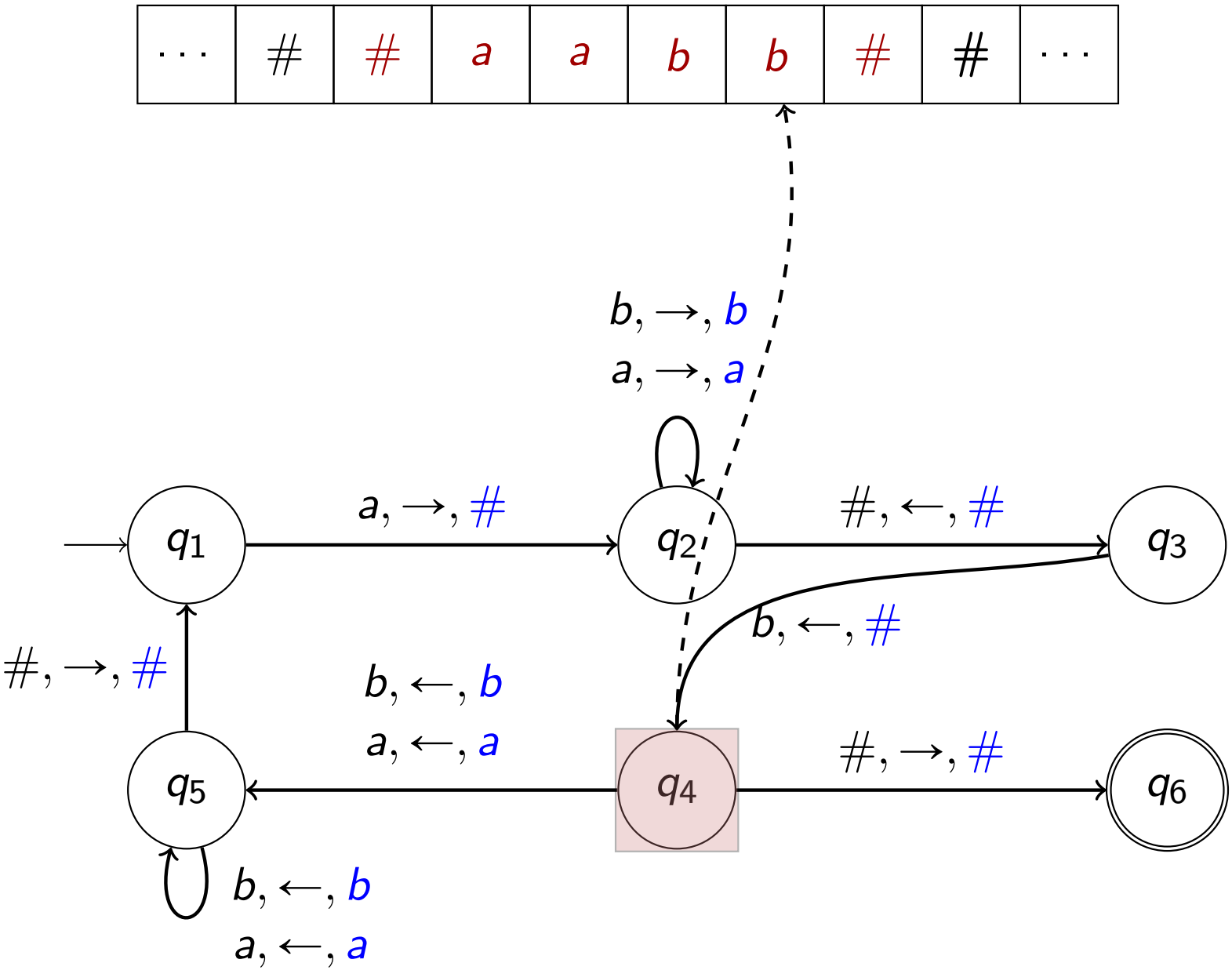
Machine de Turing, exemple 2



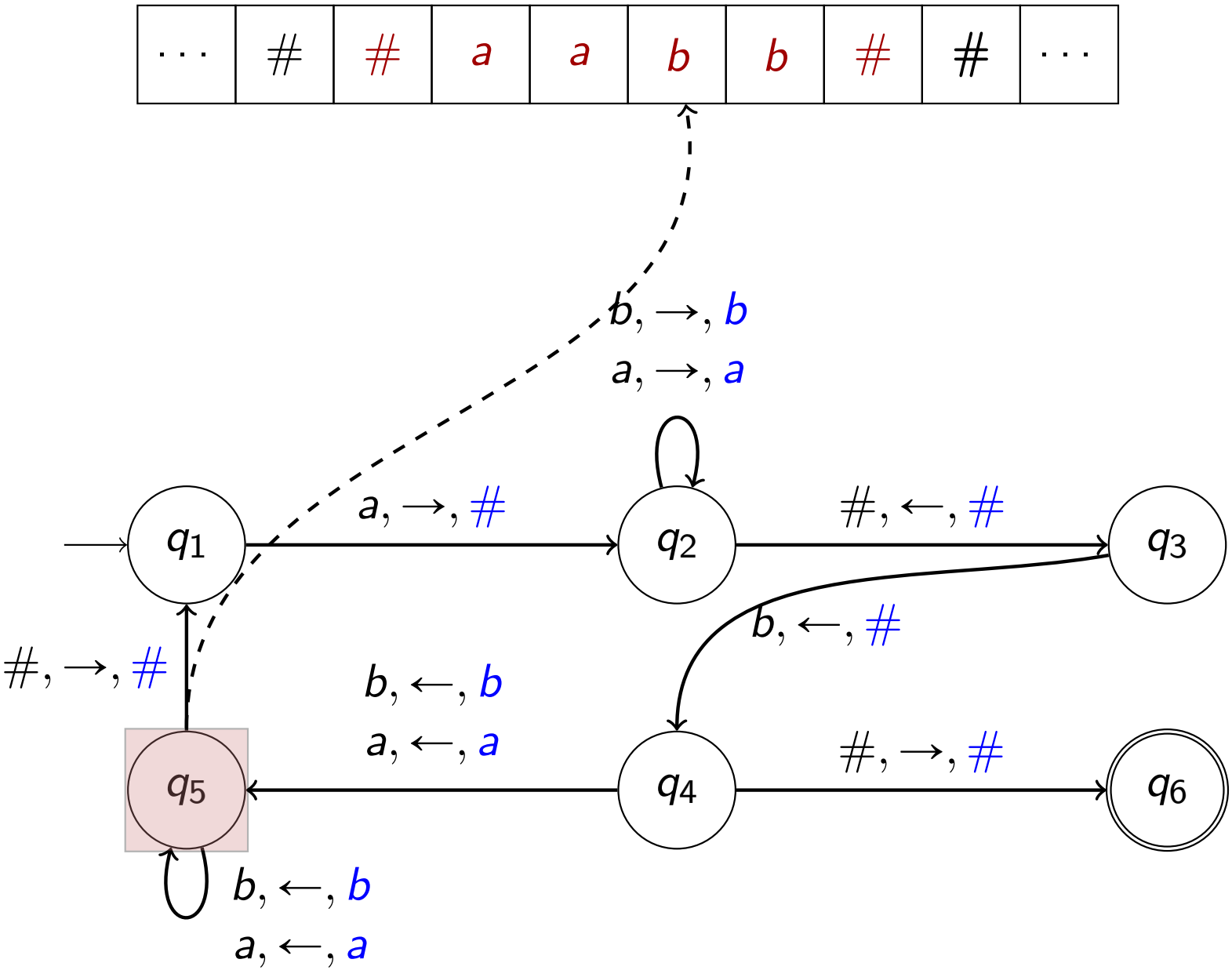
Machine de Turing, exemple 2



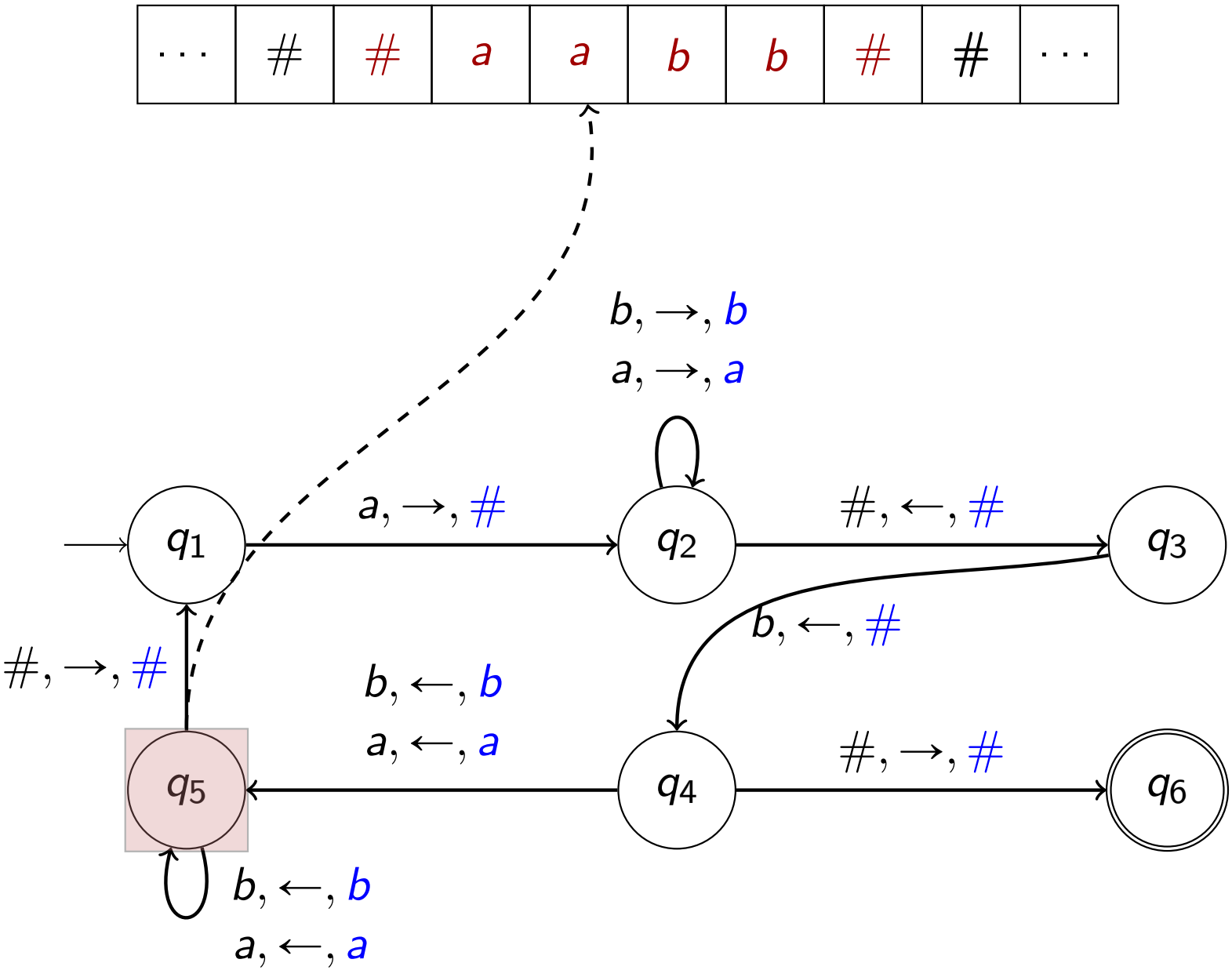
Machine de Turing, exemple 2



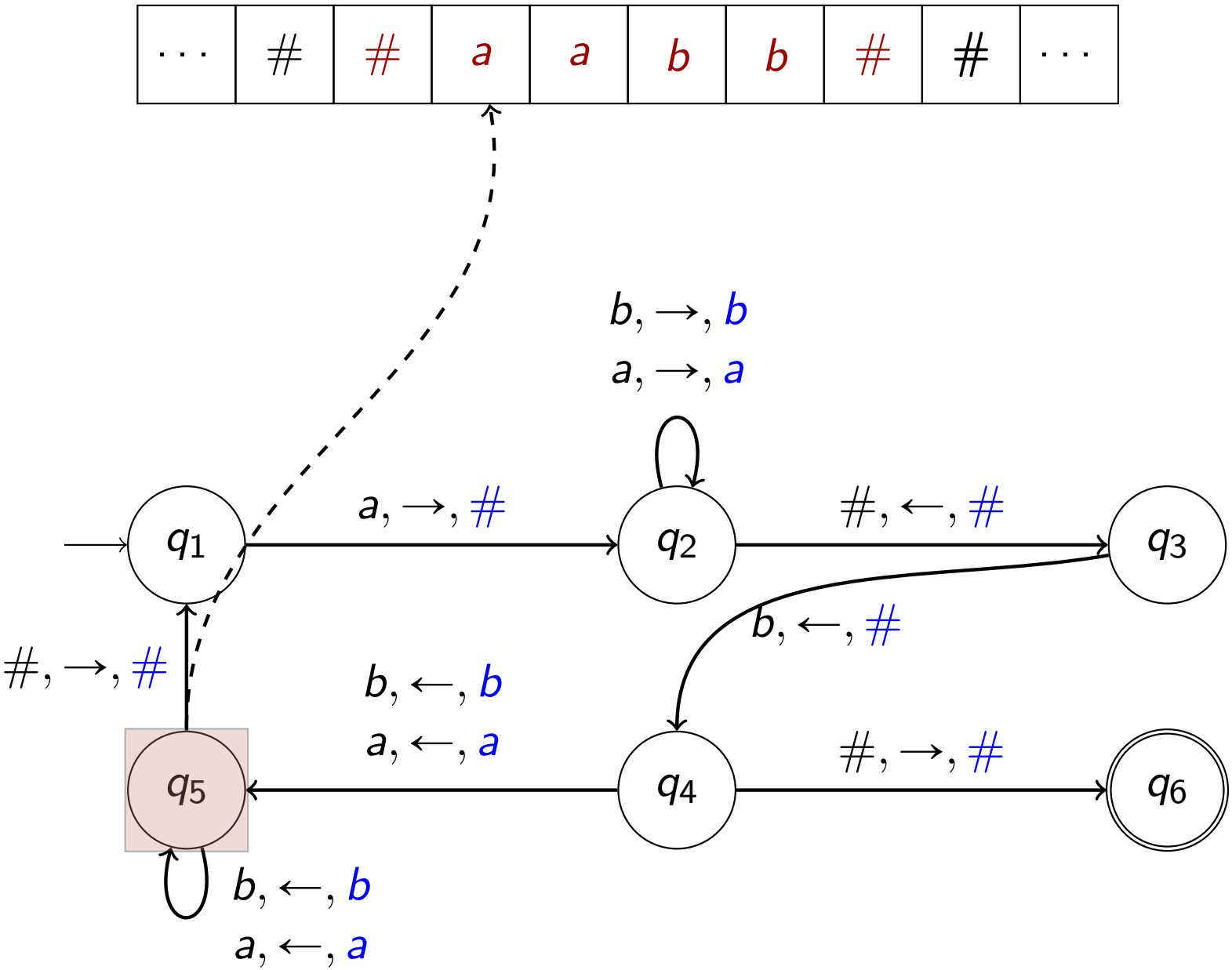
Machine de Turing, exemple 2



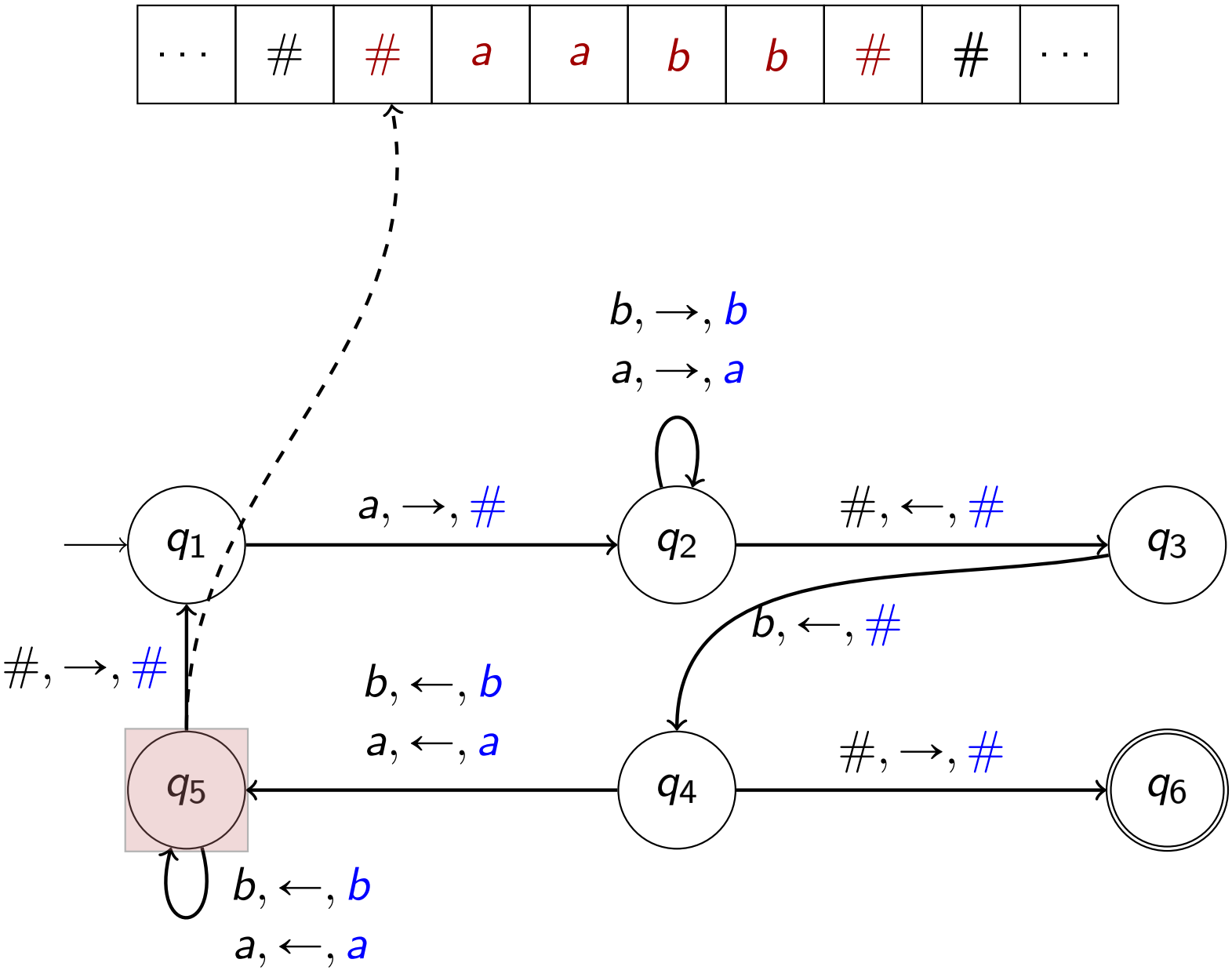
Machine de Turing, exemple 2



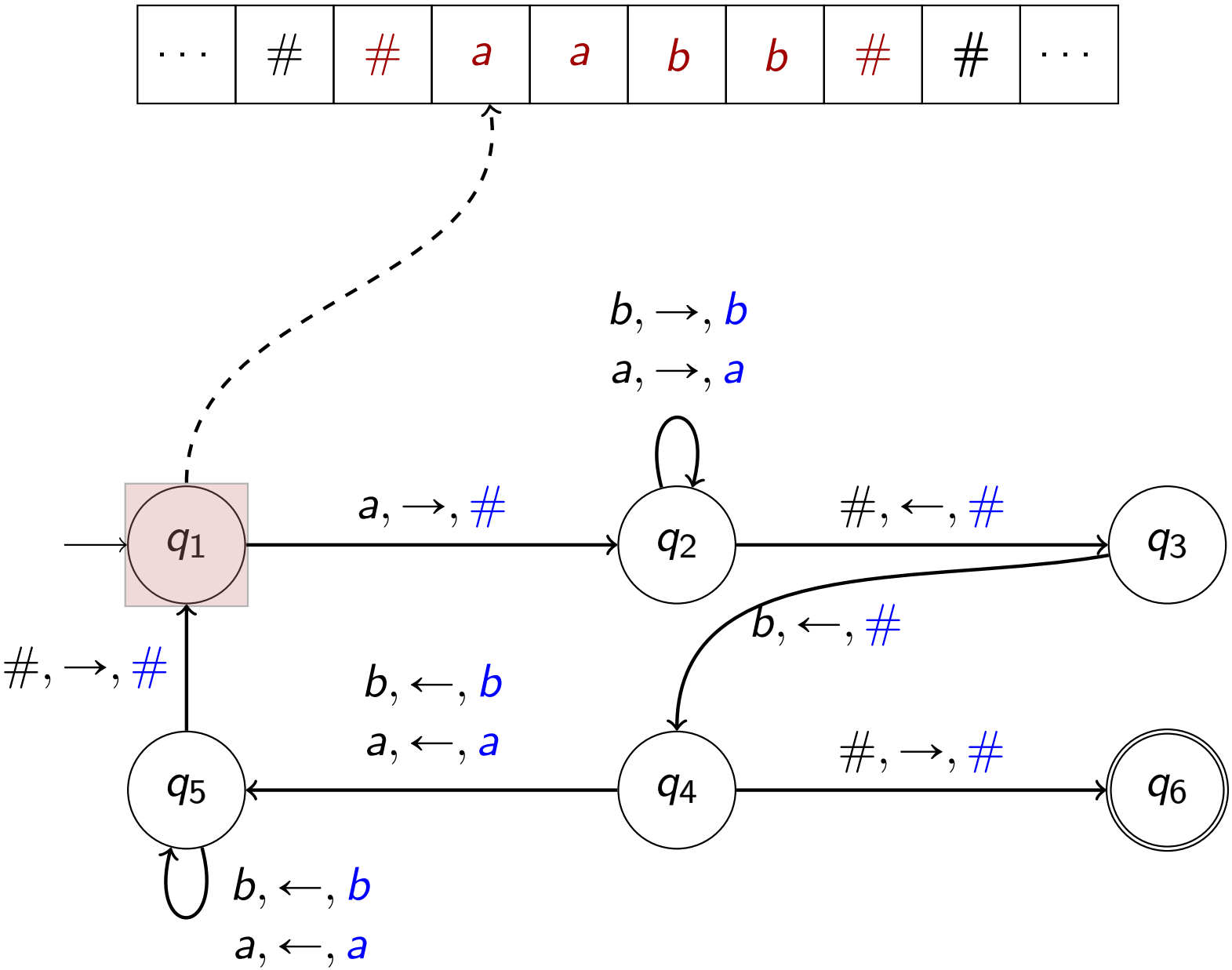
Machine de Turing, exemple 2



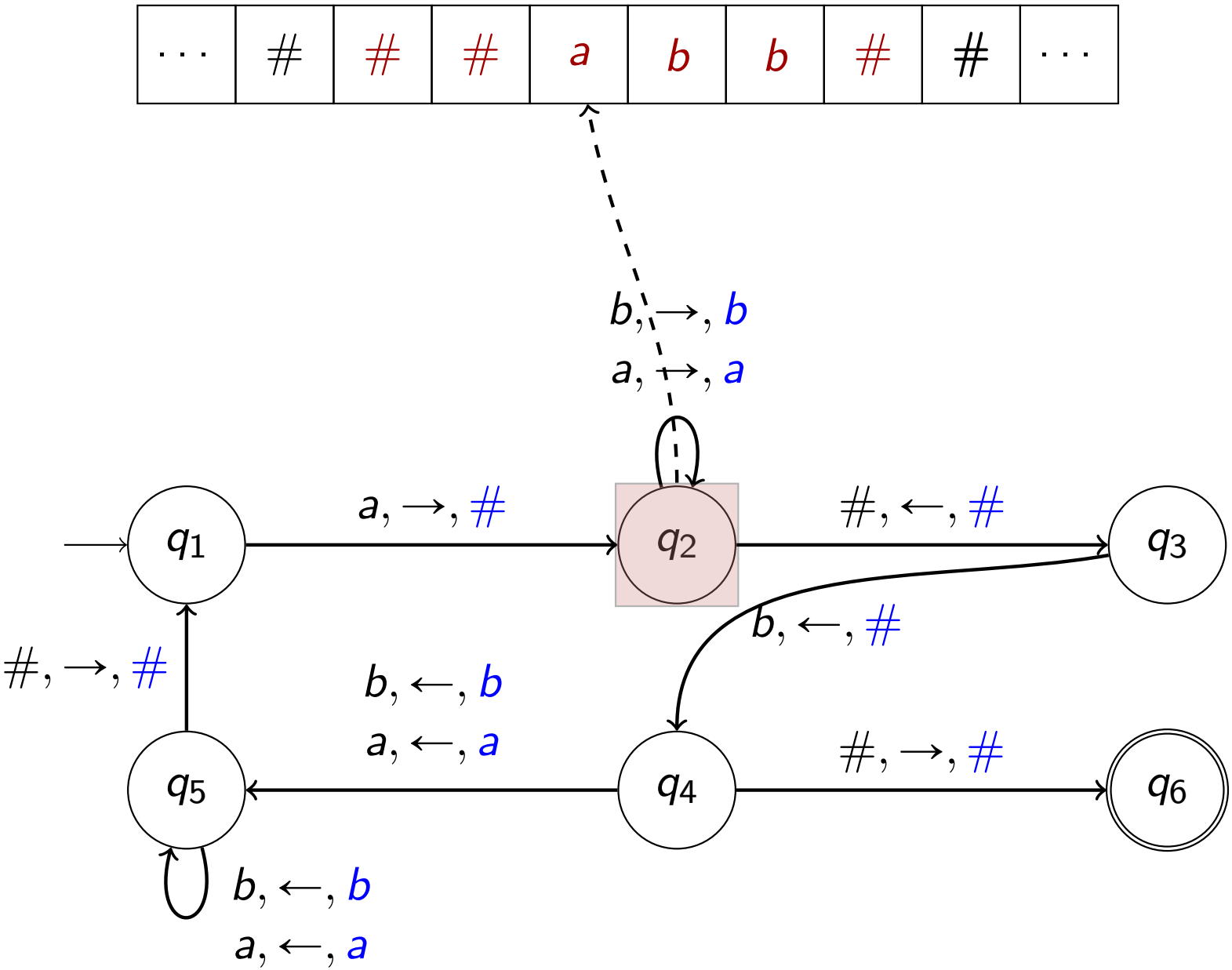
Machine de Turing, exemple 2



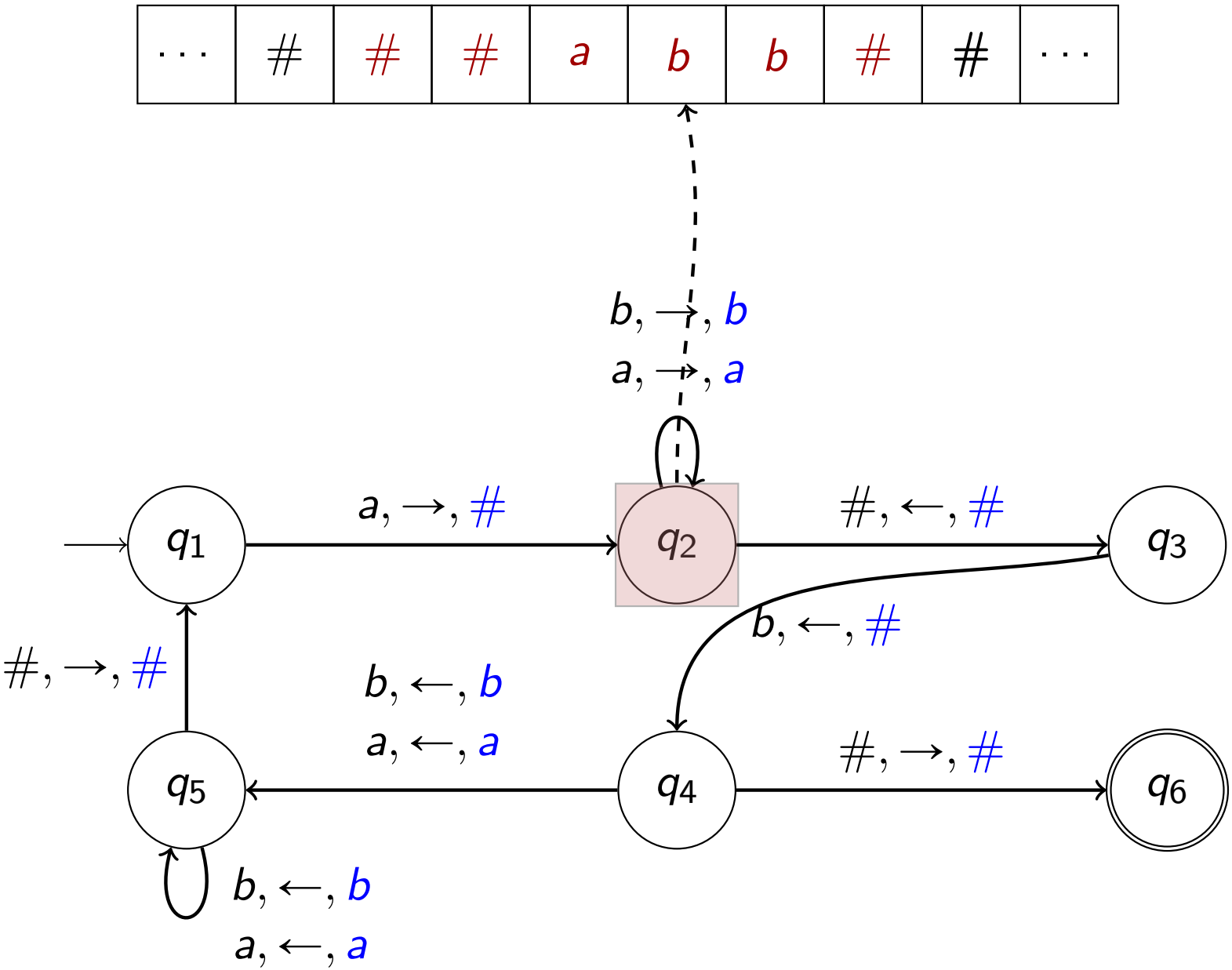
Machine de Turing, exemple 2



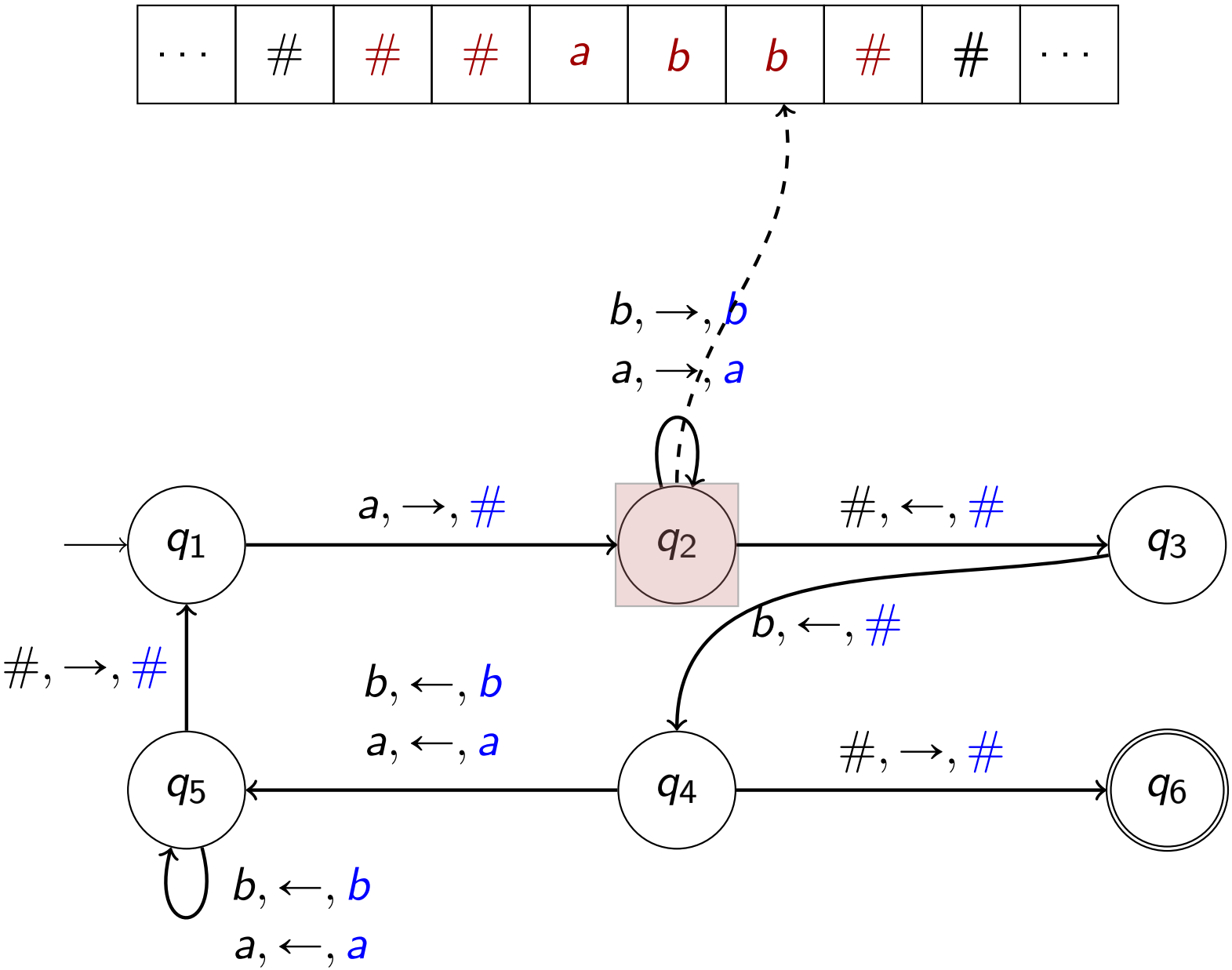
Machine de Turing, exemple 2



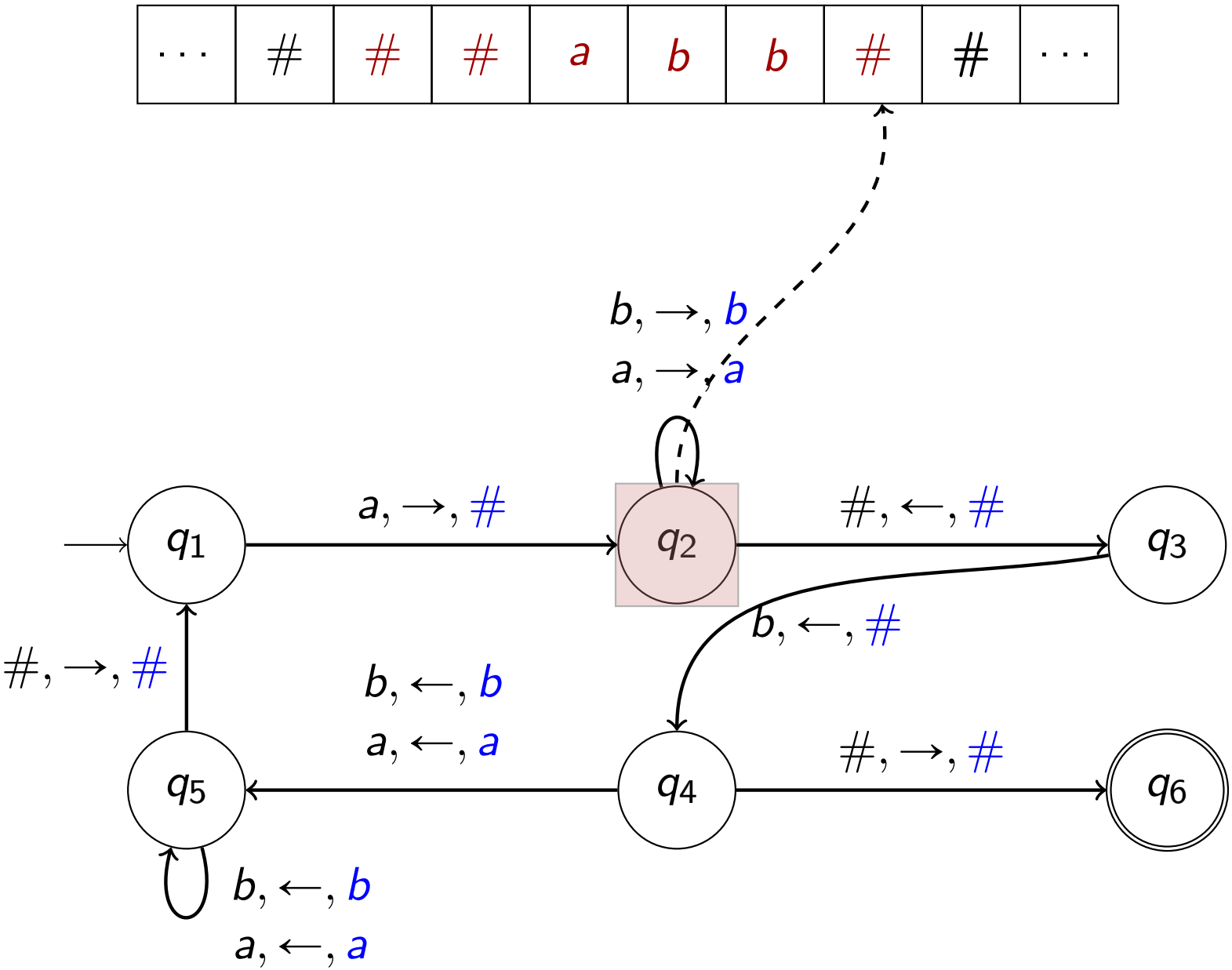
Machine de Turing, exemple 2



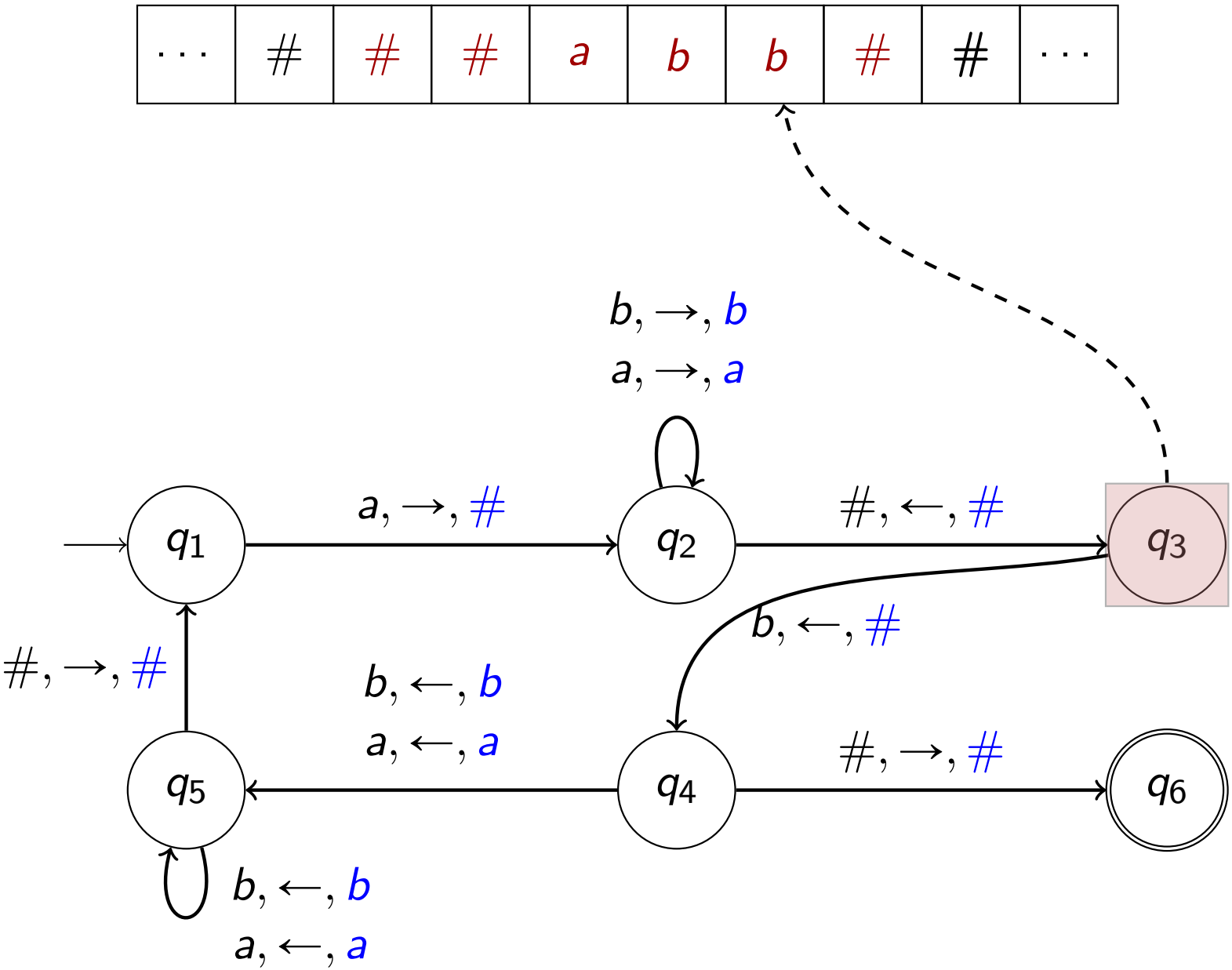
Machine de Turing, exemple 2



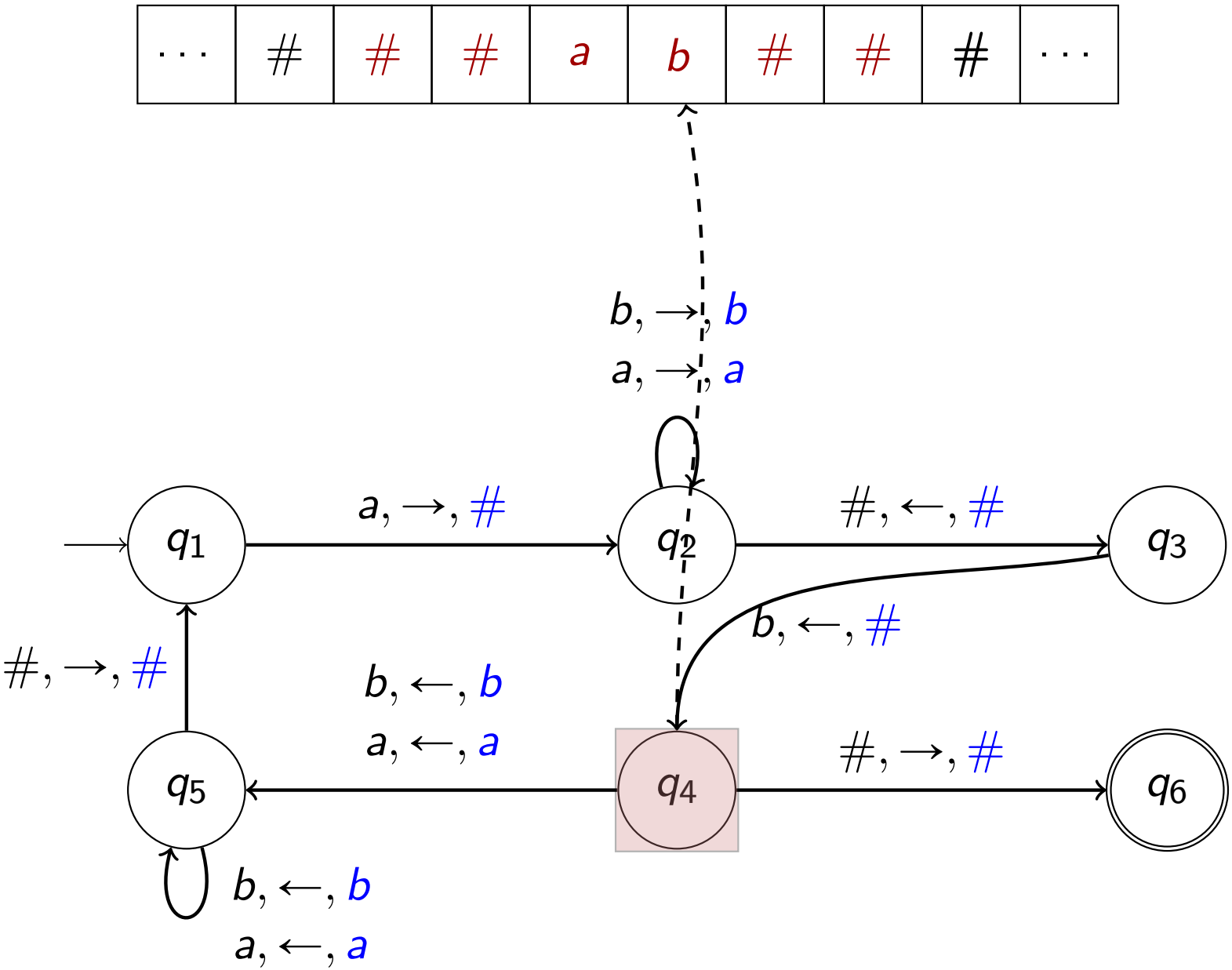
Machine de Turing, exemple 2



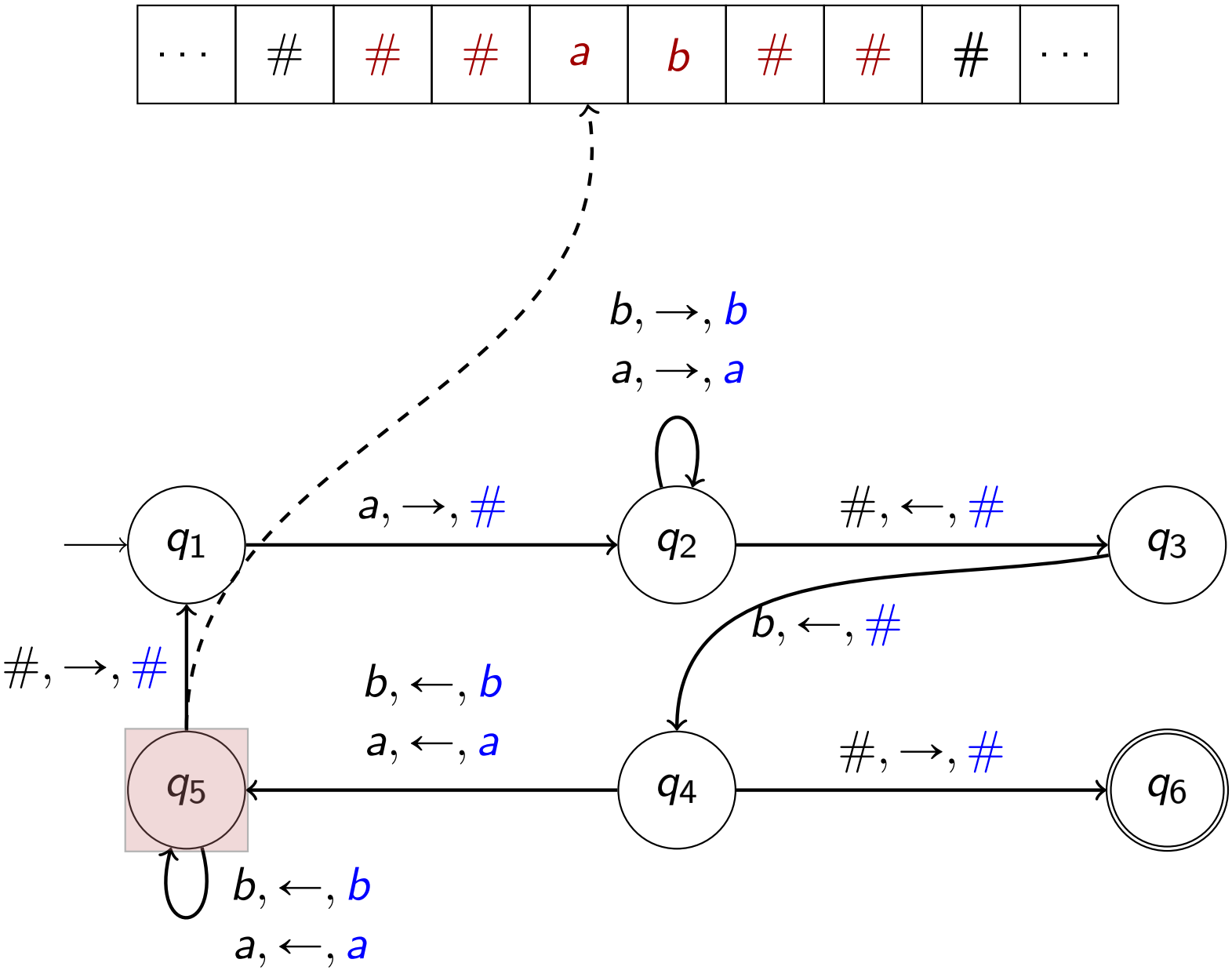
Machine de Turing, exemple 2



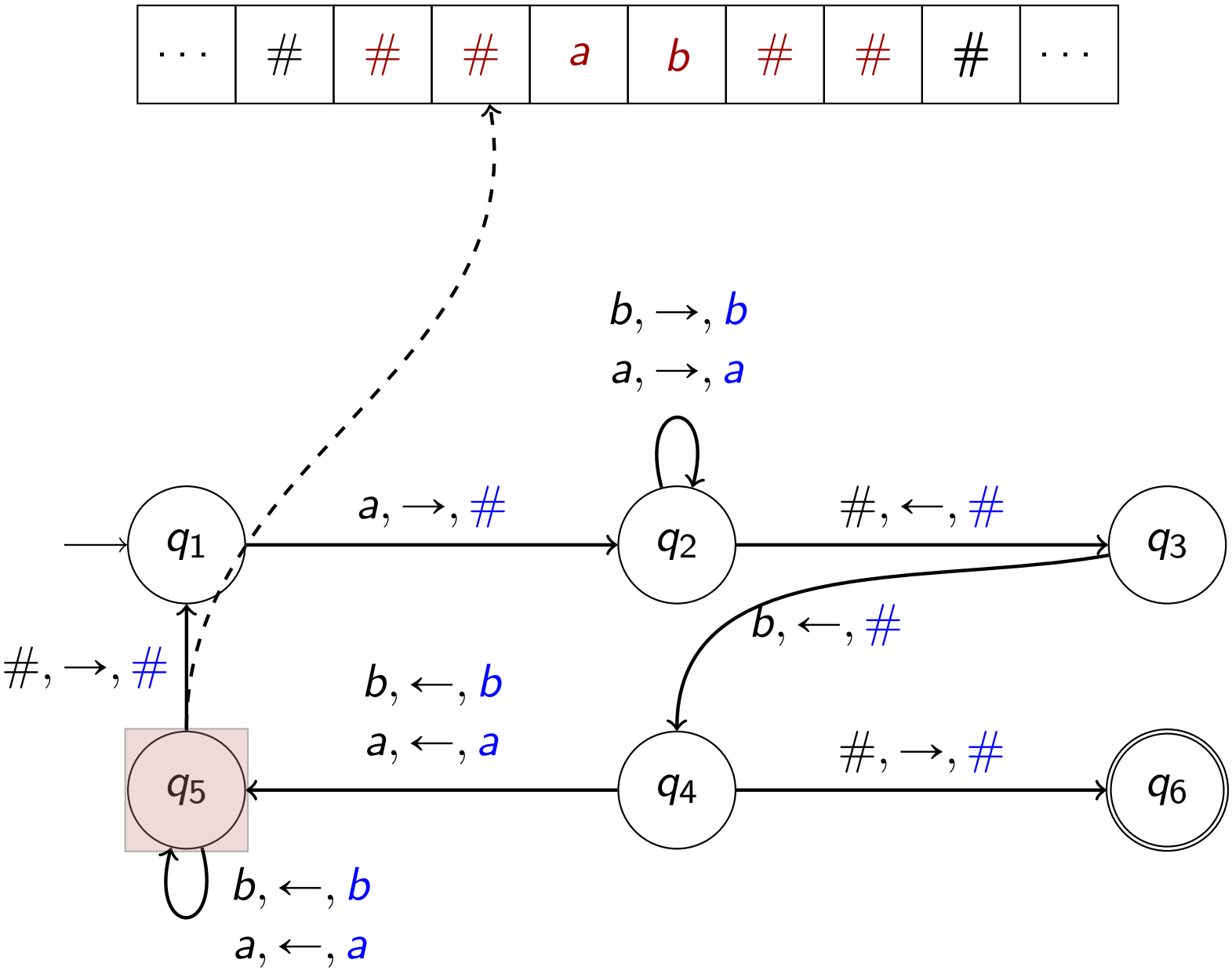
Machine de Turing, exemple 2



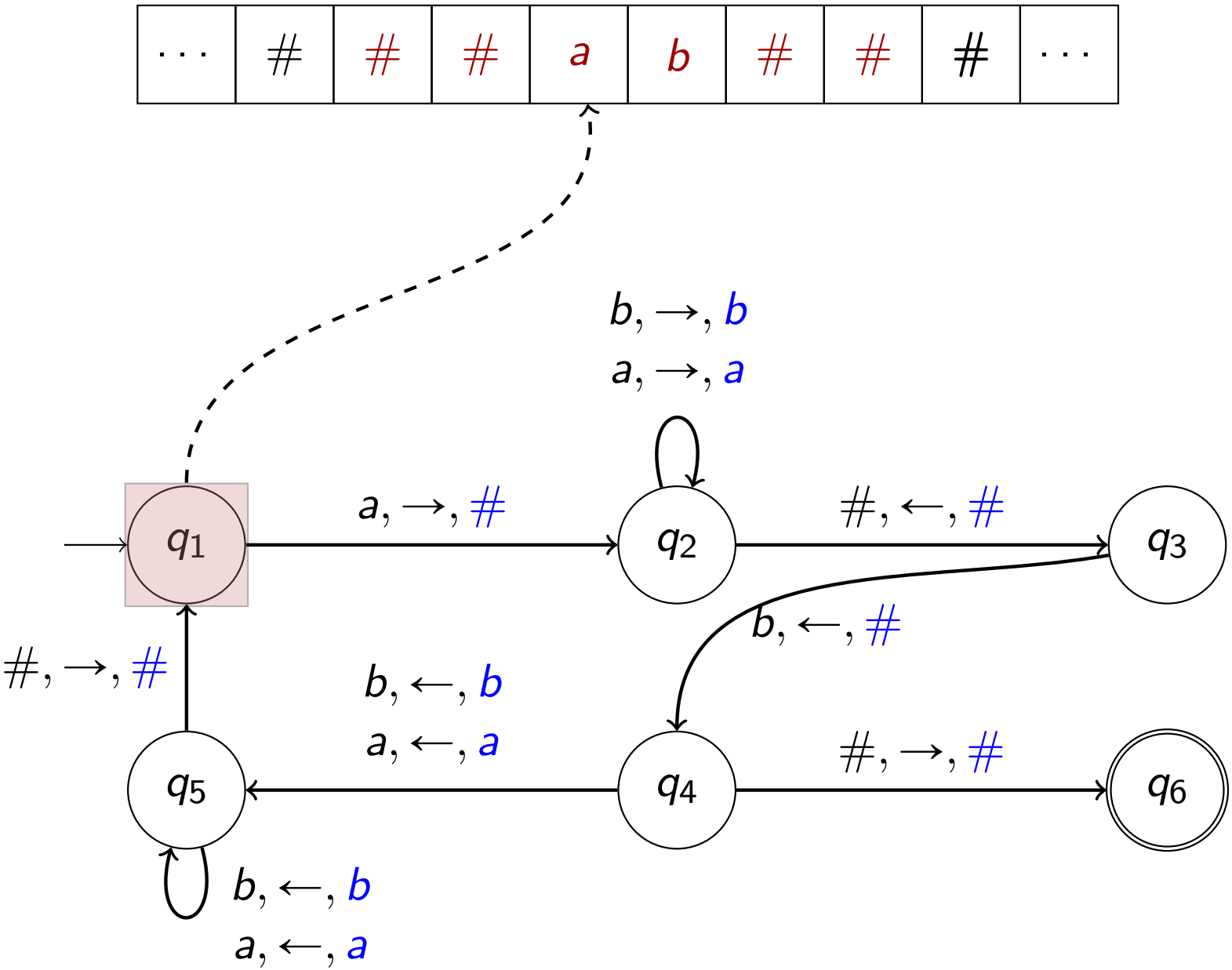
Machine de Turing, exemple 2



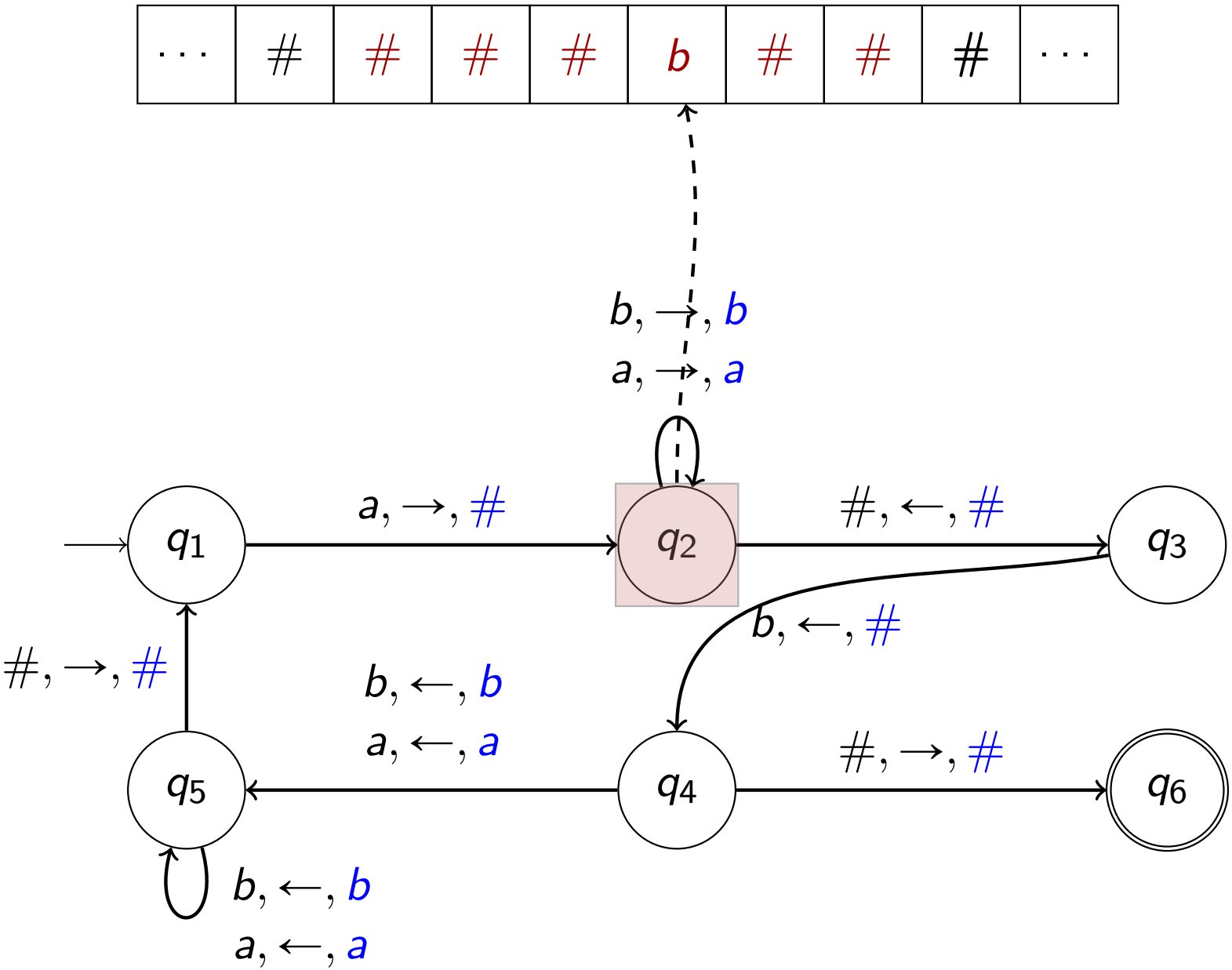
Machine de Turing, exemple 2



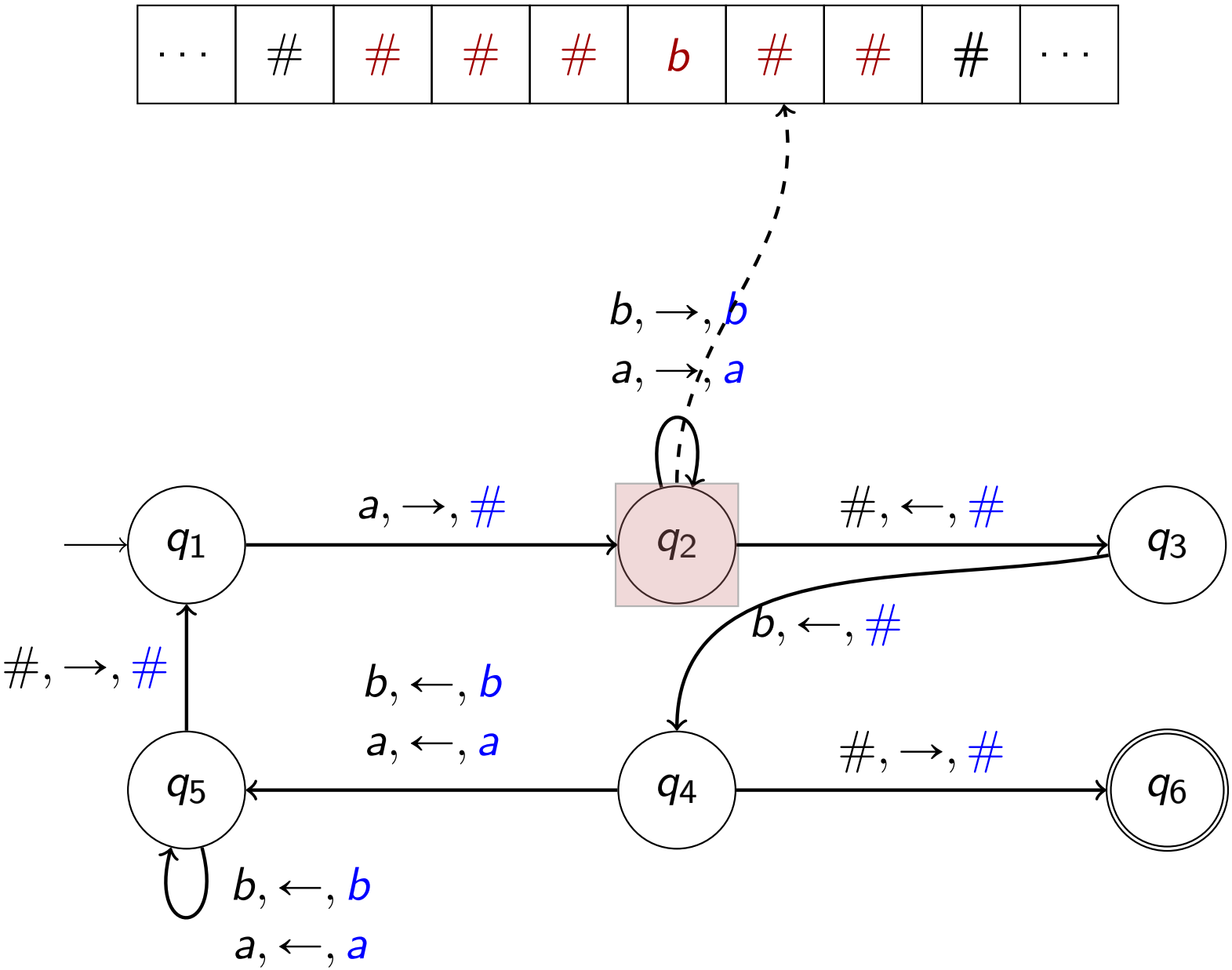
Machine de Turing, exemple 2



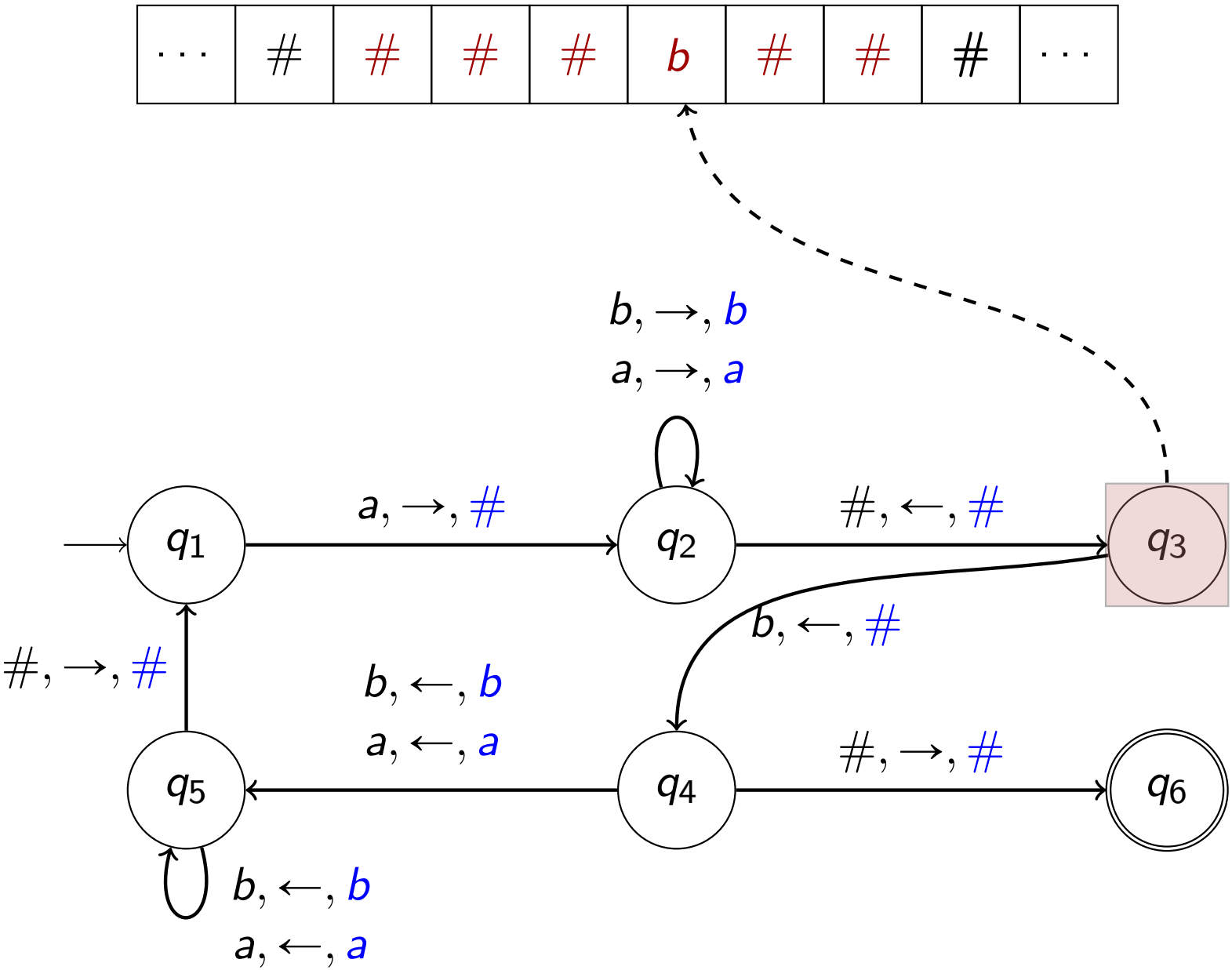
Machine de Turing, exemple 2



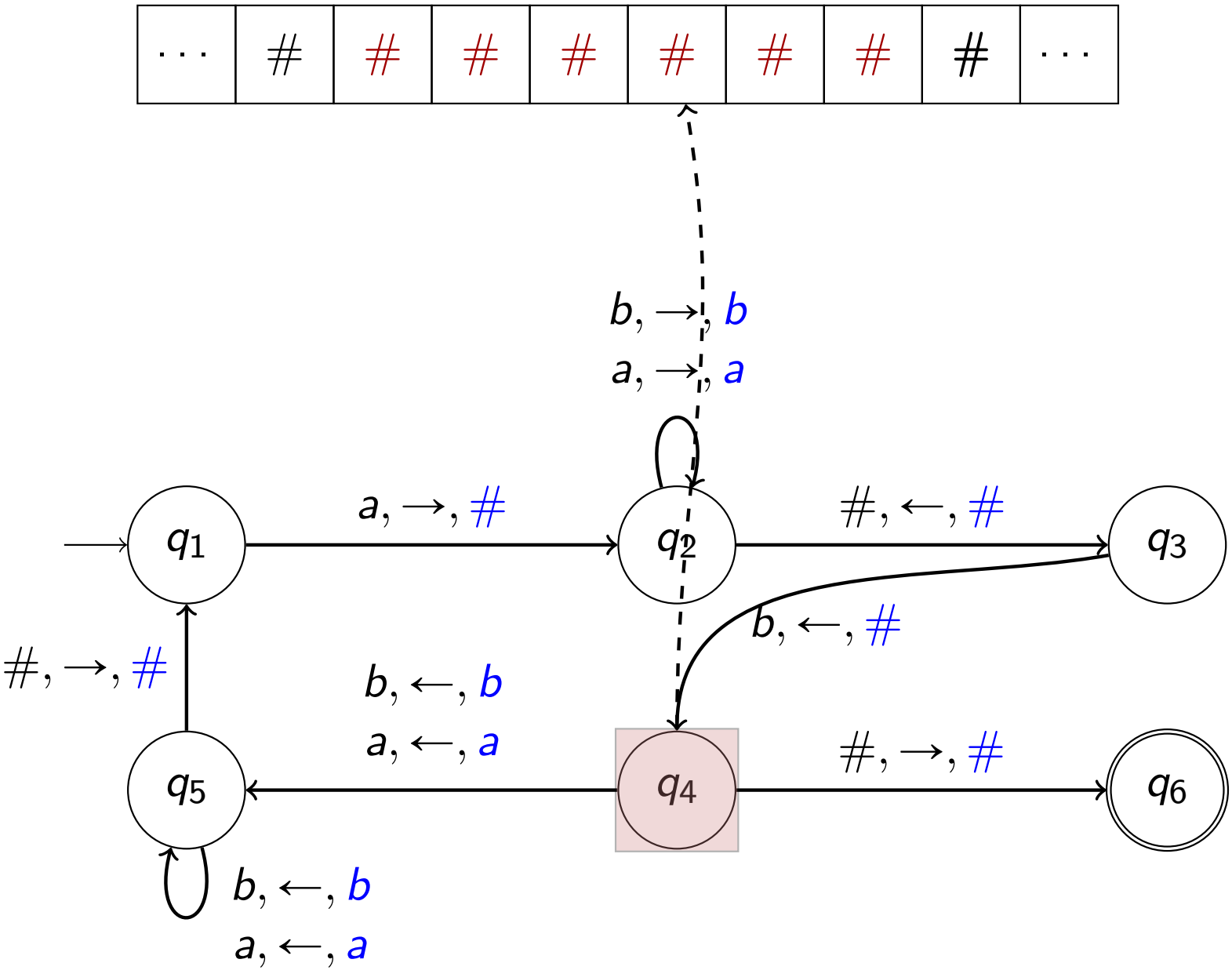
Machine de Turing, exemple 2



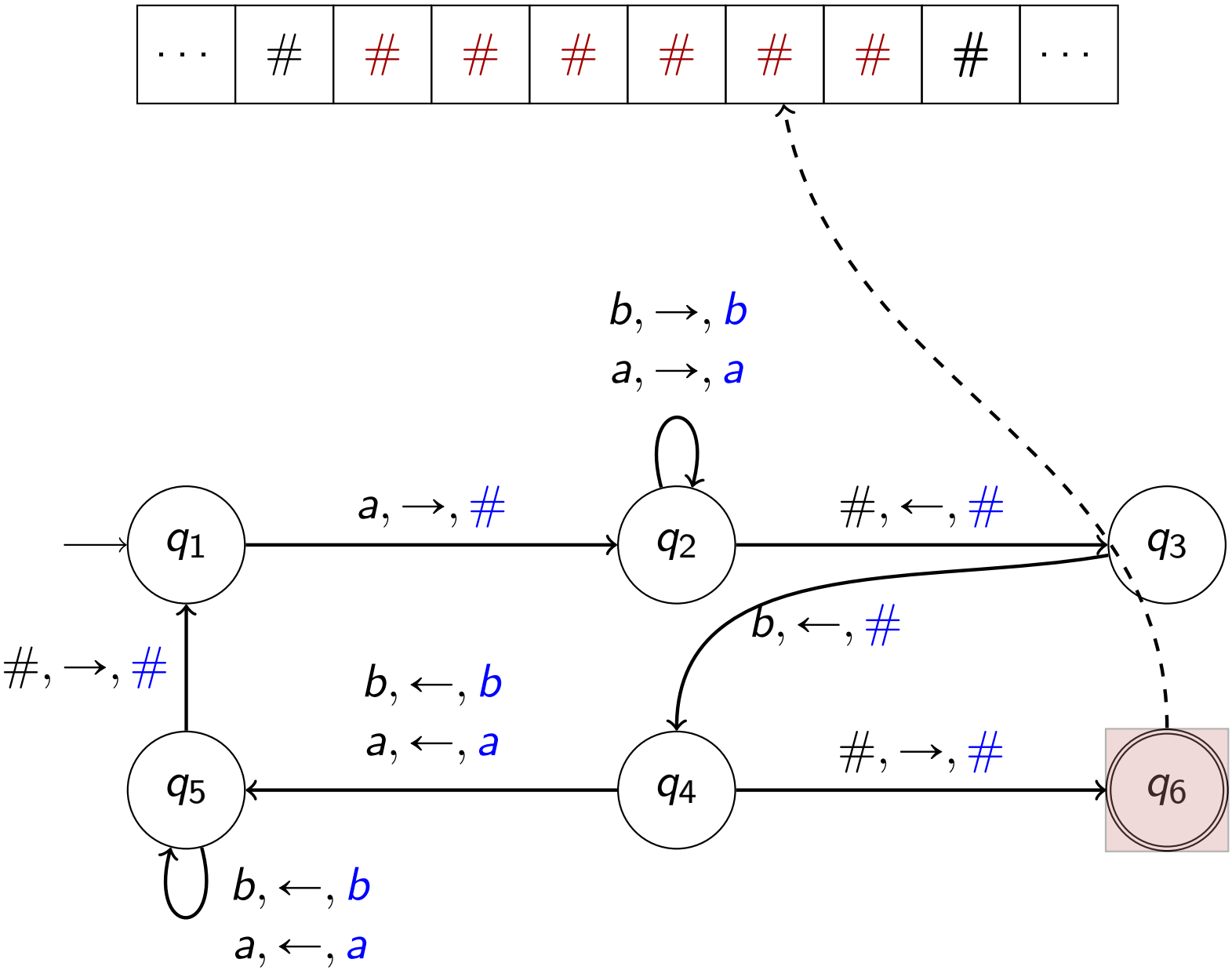
Machine de Turing, exemple 2



Machine de Turing, exemple 2



Machine de Turing, exemple 2



Machine de Turing, exemple 2

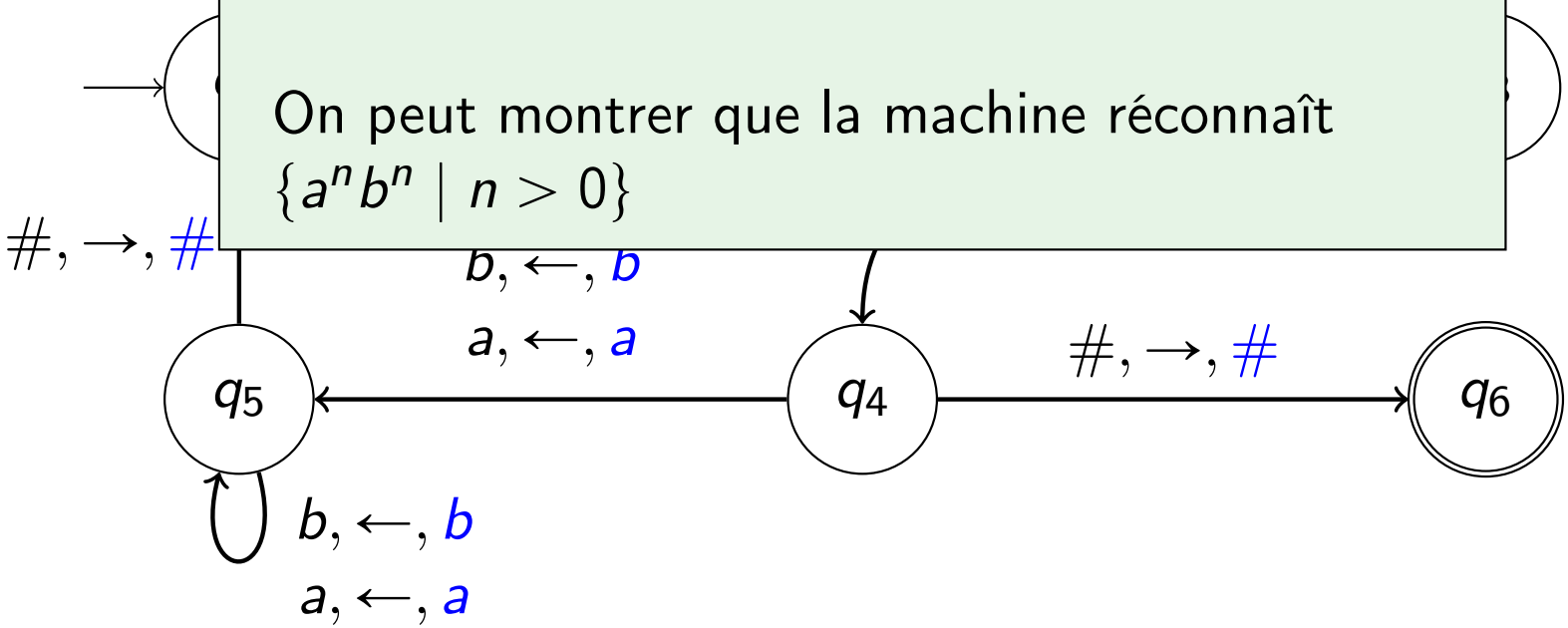
$\cdots$	#	#	#	#	#	#	#	#	$\cdots$

Le calcul s'arrete (on ne peut pas lire #)

On n'est pas passé par un état final

Le mot d'entrée *aaabbb* est reconnu.

On peut montrer que la machine réconnaît $\{a^n b^n \mid n > 0\}$



Machine de Truing déterministe : définition

Une machine de Turing déterministe est un tuple $(Q, \Sigma, \Gamma, \#, E, q_0, F)$ où :

- Q est un ensemble fini d'états,
- Σ est l'alphabet d'entrée,
- Γ est l'alphabet de travail,
- $\#$ est le symbole du blanc,
- $q_0 \in Q$ est l'état initial,
- $F \subseteq Q$ est l'ensemble des états finaux,
- $E : Q \times (\Sigma \cup \Gamma \cup \{\#\}) \to Q \times (\Sigma \cup \Gamma \cup \{\#\}) \times \{\leftarrow, \rightarrow\}$, est l'ensemble des transitions,

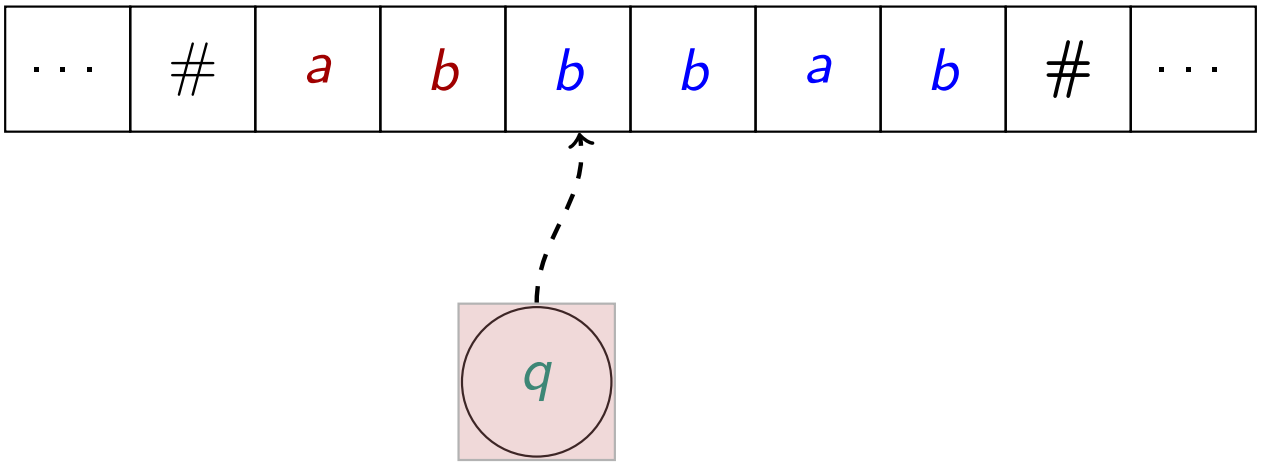
avec $\Sigma \cap \Gamma = \varnothing$ et $\# \notin \Gamma$ et $\# \notin \Sigma$.

Configurations

Une configuration d'une machine de Turing est un triplet

$$(u, q, v)$$

où u et v sont des mots des $(\Sigma \cup \Gamma \cup \{\#\})^*$ et $q \in Q$.



La première lettre de u, si elle existe n'est pas $\#$. De même pour la dernière lette de v

Calculs

Configurations initiales et finales

La configuration initiale pour v est (ε, q_0, v). Une configuration finale est une configuration de la forme (u, q, v), avec $q \in F$.

Calculs

Configurations initiales et finales

La configuration initiale pour v est (ε, q_0, v). Une configuration finale est une configuration de la forme (u, q, v), avec $q \in F$.

Configurations consécutives

Deux configurations sont consécutives s'il elles corresepondent à l'activation d'une transition dans la machine de Turing. On note $C_1 \models C_2$.

Calculs

Configurations initiales et finales

La configuration initiale pour v est (ε, q_0, v). Une configuration finale est une configuration de la forme (u, q, v), avec $q \in F$.

Configurations consécutives

Deux configurations sont consécutives s'il elles corresepondent à l'activation d'une transition dans la machine de Turing. On note $C_1 \models C_2$.

Calcul

Un calcul sur un mot w est une suite de configurations consécutives partant de la configuration initiale de w.
Dans une machine de Turing, il y a un unique calcul maximal (qui ne peut pas être prolongé) pour chaque mot w.

Acceptation

Mot accepté

Un mot est accepté par une machine de Turing déterministe si le calcul maximal sur ce mot s'arrete **et** passe par un état final.

Mot non accepté

Un mot n'est accepté pa rune machine de Turing déterministe si le calcul maximal sur ce mot est infini **ou** ne passe par aucun état final.

Acceptation

Mot accepté

Un mot est accepté par une machine de Turing déterministe si le calcul maximal sur ce mot s'arrete **et** passe par un état final.

Mot non accepté

Un mot n'est accepté pa rune machine de Turing déterministe si le calcul maximal sur ce mot est infini **ou** ne passe par aucun état final.

Il y a une nuance entre langage accepté et langage décidé.

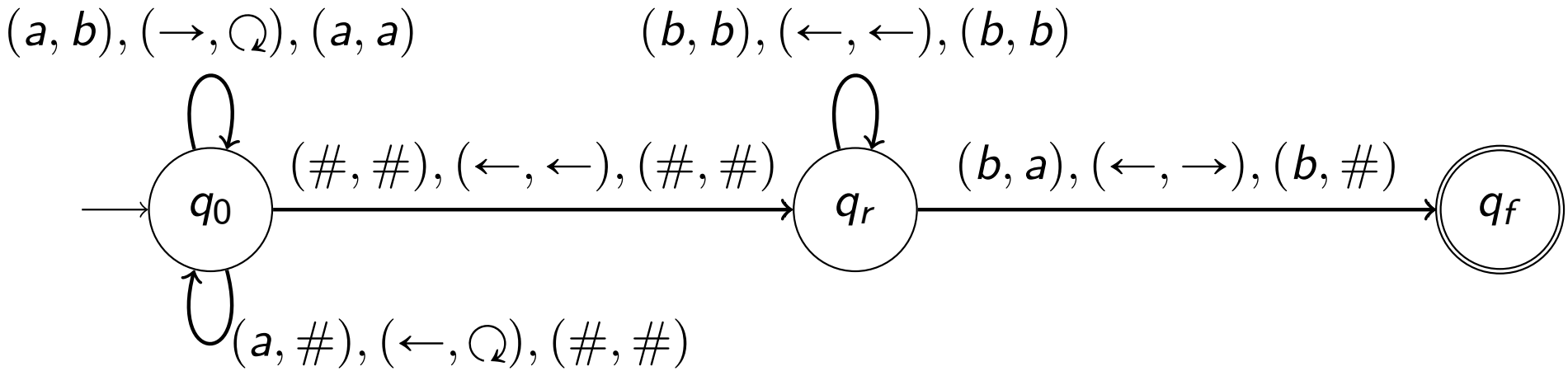
Exercice

Dessiner et décrire une machine de Turing déterministe sur $\{a, b\}$ qui reconnait les palindrômes (mots qui se lisent de la même façon de droite à gauche et de gauche à droite).

Variations

Sans changer l'expressivité (cela se démontre), on peut :

- Faire du sur place,
- Utiliser plusieurs rubans.



Exercices

Exercice 1

Décrire des machines de Turing déterministes sur $\{a, b\}$ qui reconnassent les langages suivants :

1. Ensemble des mots ayant autant de a que de b.
2. Ensemble des mots de la forme uu où u est un mot.
3. Ensemble des mots de la forme $uaav$ ou $ubbv$ avec $|u| = |v|$.

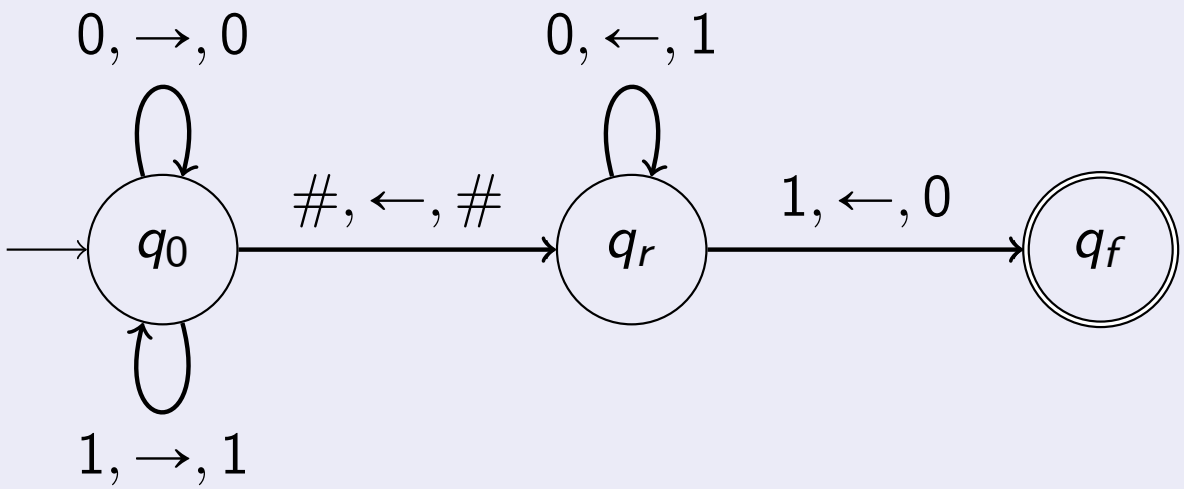
Exercice 2

Justifier en décrivant une construction que *le sur place* n'augmente pas l'expressivité des machines de Turing déterministes.

Exercice

Exercice 3

On considère la machine suivante :



1. Ecrire le calul maximal sur 1001001. Que lit-on sur le ruban à la fin ?
2. Même question avec 1011100.
3. Quel est le lien entre le mot d'entrée et ce qui est écrit à la fin du calcul sur le ruban ?

Décidabilité, thèse de Chruch

Définition

Un langage est décidable s'il existe une machine de Turing déterministe qui accepte tous les mots du langage et telle que, pour tout mot non reconnu, le calcul maximal termine.

Décidabilité, thèse de Chruch

Définition

Un langage est décidable s'il existe une machine de Turing déterministe qui accepte tous les mots du langage et telle que, pour tout mot non reconnu, le calcul maximal termine.

Thèse de Church (cela ne se démontre pas)

Un langage est reconnaissable par un méchanisme automatique de décision si et seulement s'il est décidable.

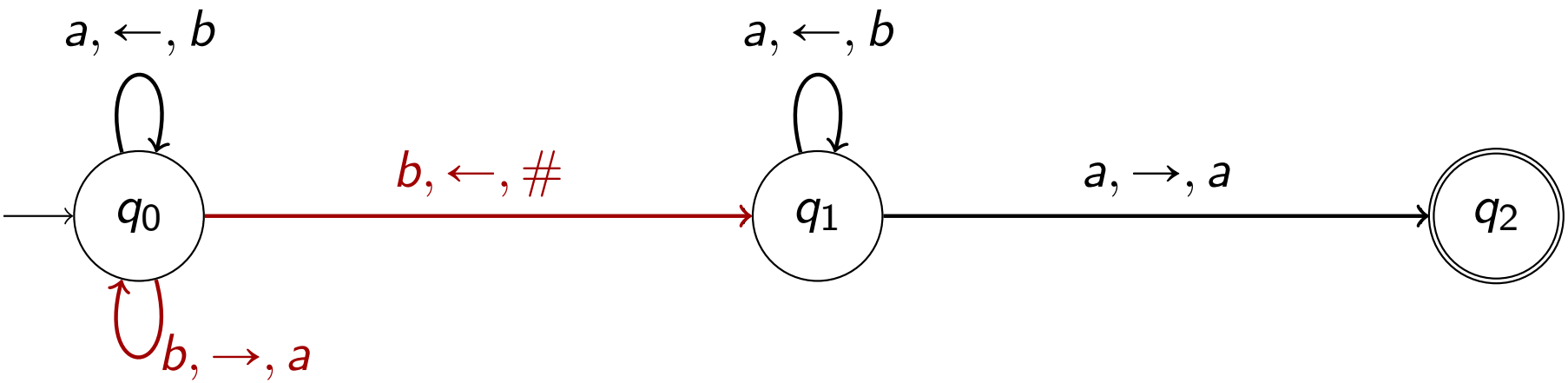
Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Machine non déterminite

Une machine de Turing non-déterministe est une machine de Turing pour laquelle :

- Il peut y avoir plusieurs états initiaux,
- Les transitions ne sont plus nécessairement fonctionnelles.



Mots acceptés

Configurations initiales, finales, consécutives se définissent comme pour les machines déterministes.

Mot accepté

Un mot w est accepté s'il existe un calcul maximal sur ce mot qui termine et qui passe par un état final.

Théorème

Un langage est accepté par une machine de Turing déterministe si et seulement s'il l'est par une machine non déterministe. ‘

Pourquoi le non déterminisme ?

Exercice

1. Dessiner un automate fini sur $\{a, b\}$ reconnaissant l'ensemble des mots dont la 6ième lettre à partir de la fin est un a.
2. Même question avec un automate déterministe.

Pourquoi le non déterminisme ?

Exercice

1. Dessiner un automate fini sur $\{a, b\}$ reconnaissant l'ensemble des mots dont la 6ième lettre à partir de la fin est un a.
2. Même question avec un automate déterministe.

Le non-déterminisme permet :

- D'écrire des modèles de calcul beaucoup plus simples et compacts (et lisibles), même si moins fonctionnels,
- De classifier de façon pertinente des problèmes.

L'algorithmique non-déterministe sera vue plus tard dans ce cours.

Plan

1. Théorie des langages (rappels)
2. Problèmes et théorie des langages
3. Automates finis
4. Langages et automates
5. Machines de Turing déterministes
6. Machines de Turing non déterministes
7. Indécidabilité

Indécidabilité

Un langages est indécidable s'il n'existe pas de machine de Turing le décidant.

Indécidabilité

Un langages est indécidable s'il n'existe pas de machine de Turing le décidant.

Il y a un nombre continu de langages sur un alphabet donné. Il y a un nombre dénombrable de machines de Turing. Il existe donc beaucoup de langages indécidables... *mais il y a un nombre dénombrable de problèmes exprimables avec un alphabet donné.*

Indécidabilité

Un langages est indécidable s'il n'existe pas de machine de Turing le décidant.

Il y a un nombre continu de langages sur un alphabet donné. Il y a un nombre dénombrable de machines de Turing. Il existe donc beaucoup de langages indécidables... *mais il y a un nombre dénombrable de problèmes exprimables avec un alphabet donné.*

On peut construire un langage indécidable par un procédé diagonal (on en va pas le faire).

Pavage du plan

On considère le problème suivant :
Données : un ensemble T fini de tuiles carrées dont les bords sont colorés et dont l'orientation est fixée.

Question : Peut-on paver n'importe quelle surface avec des tuiles ayant uniquement des motifs appartenant à T de façon à ce que les couleurs de deux arrêtes de tuiles qui se touchent soient les mêmes ?

Deux instances de ce problème :

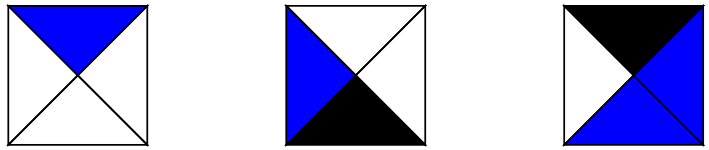


Figure – une instance du problème de pavage pour laquelle une solution existe.

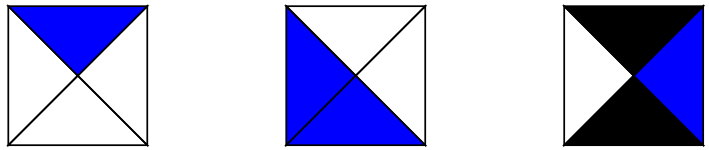


Figure – une instance du problème de pavage pour laquelle aucune solution n'existe.

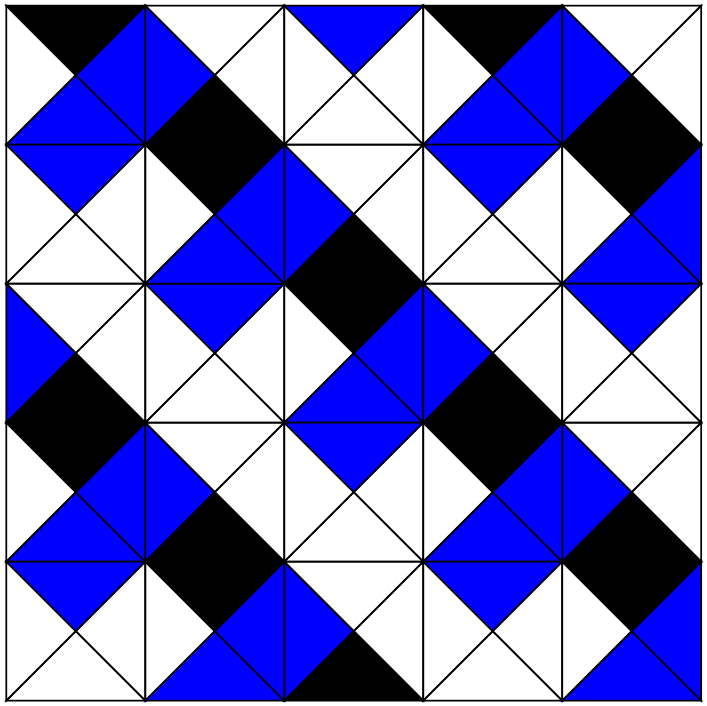


Figure – solution pour la première instance.

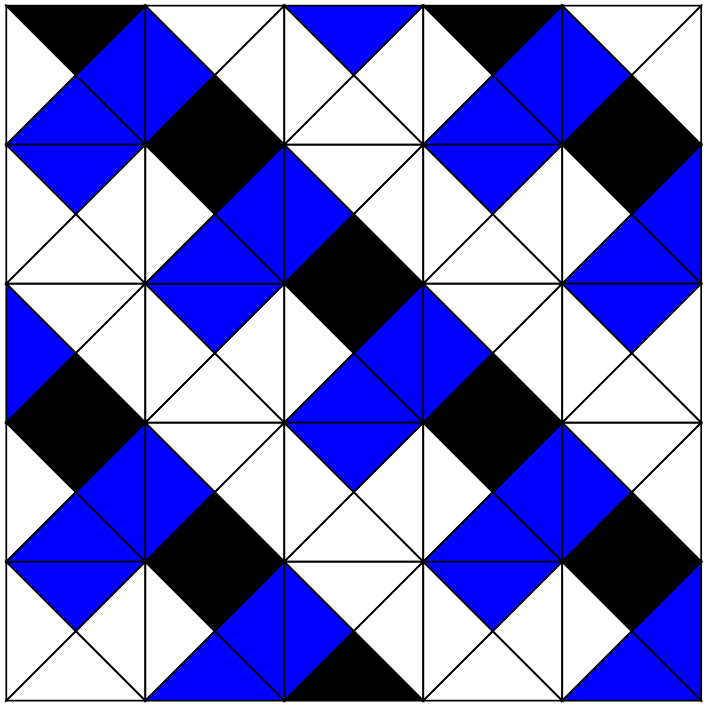


Figure – solution pour la première instance.

Ce problème est indécidable.

Problème de l'arrêt

On considère le problème suivant, connu sous le nom de problème de l'arrêt que l'on peut montrer indécidable.

Problème de l'arrêt

Données : Une machine de Turing déterministe (ou un programme) et une entrée w de cette machine (Programme).
Question : L'éxecution de la machine sur l'entrée termine-t-elle ?

Problème de l'arrêt

On considère le problème suivant, connu sous le nom de problème de l'arrêt que l'on peut montrer indécidable.

Problème de l'arrêt

Données : Une machine de Turing déterministe (ou un programme) et une entrée w de cette machine (Programme).
Question : L'éxecution de la machine sur l'entrée termine-t-elle ?

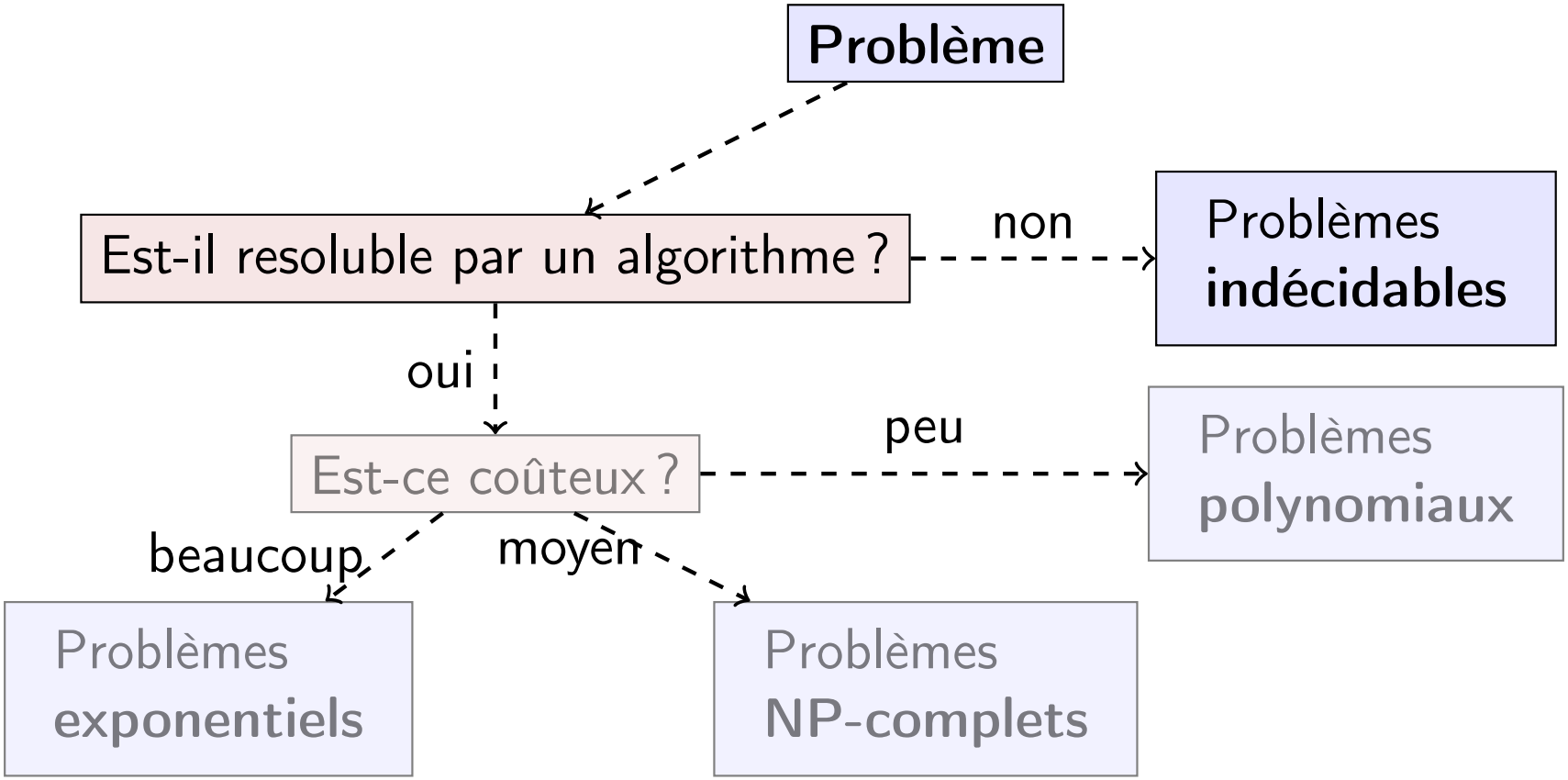
Problème de l'arrêt universel

Données : Une machine de Turing déterministe (ou un programme).
Question : L'éxecution de la machine termine-t-elle pour toute entrée ?

Problème de l'arrêt existentiel

Données : Une machine de Turing déterministe (ou un programme).
Question : Éxiste-t-il une entrée pour laquelle la machine s'arrete ?

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)