Algorithmique et complexité: CM1

Florian Bridoux

Polytech Nice Sophia

2023-2024

Table des matières

Table of Contents

Organisation du cours

- Enseignants:
  - Responsable: Florian Bridoux (CM et TD)
  - E-mail: bridoux@i3s.unice.fr
  - Chargés de TD:
    - Adrien Richard
    - Francois Dore
    - Margaux Schmied
- Séances:
  - 12 CM de 1h et 12 TD de 2h30.
  - 4 séances de calculabilité
  - 4 séances de complexité
  - 4 séances d'algo avancé (programmation dynamique, approximation,...)

Table of Contents

Calculabilité et Complexité: Une définition

Definition

En informatique, un problème est une fonction $f : A \to B$ où A et B sont des ensembles *dénombrables* représentants des objets.

Exemples d'objets: des booléens, des graphes, des entiers, des listes d'entiers, des fractions, etc.

Remarque

Les objets doivent avoir une représentation finie, on ne peut par exemple pas mettre en entrée ou en sortie de la fonction la liste de tous les décimales de π car celle-ci est infinie. On verra (voir TD1) qu'on peut considérer que $A, B \subseteq \mathbb{N}$.

Calculabilité et Complexité: Une définition

Exemples de problèmes informatique:

- Étant donné un entier, est-il pair? ($f : \mathbb{Z} \to \{\top, \bot\}$)
- Le double d'un nombre n? ($f : \mathbb{Z} \to \mathbb{Z}$)
- Étant donné un entier naturel, est-il premier? ($f : \mathbb{N} \to \{\top, \bot\}$)
- Étant donnée une liste d'entiers, est-elle triée? ($f : \mathbb{Z}^* \to \{\top, \bot\}$)
- Quel est le minimum d'une liste d'entiers? ($f : \mathbb{Z}^* \to \mathbb{Z}$)
- Y a-t-il un chemin entre $a \in G$ et $b \in G$ dans un graphe G?
- Étant donné un programme écrit en python, ce programme-va-t'il se terminer ou faire une boucle infinie?

Calculabilité et Complexité: Une définition

Definition

Un algorithme est une suite finie et non ambiguë d'instructions et d'opérations.

Dans notre cas on suppose qu'un algorithme prend une entrée $a \in A$ (potentiellement inutilisée) et sort une sortie $b \in B$.

Remarque

Tous les algorithmes (qui nous intéressent) calculent donc une fonction et on dit donc que cette fonction est *calculable*. Mais, on verra que toutes les fonctions ne sont pas calculables par un algorithme... et on dit donc que ce sont des fonctions *non-calculables*.

Calculabilité et Complexité: Une définition

La question principale de la calculabilité est: étant donné un problème et donc une fonction $f : A \to B$, est-ce que f est calculable?

Remarques:

- f est calculable veut dire qu'il existe un algorithme qui calcule f: si on lance l'algorithme sur tout $a \in A$, on obtient $f(a)$.
- En calculabilité, on ne se préoccupe pas du *temps* ni de l'*espace mémoire* utilisé par l'algorithme pour résoudre le problème. Si f est calculable, alors on peut le prouver en exhibant un algorithme, peu importe ses performances.
- On verra des notions moins forte que calculable: semi-décidable, co-semi-décidable...

Calculabilité et Complexité: Une définition

La question principale de la complexité est: étant donné un problème p calculable et donc une fonction $f : A \to B$, quelle est la difficulté de p, c'est-à-dire quelles sont les performances du meilleur algorithme qui peut calculer f?

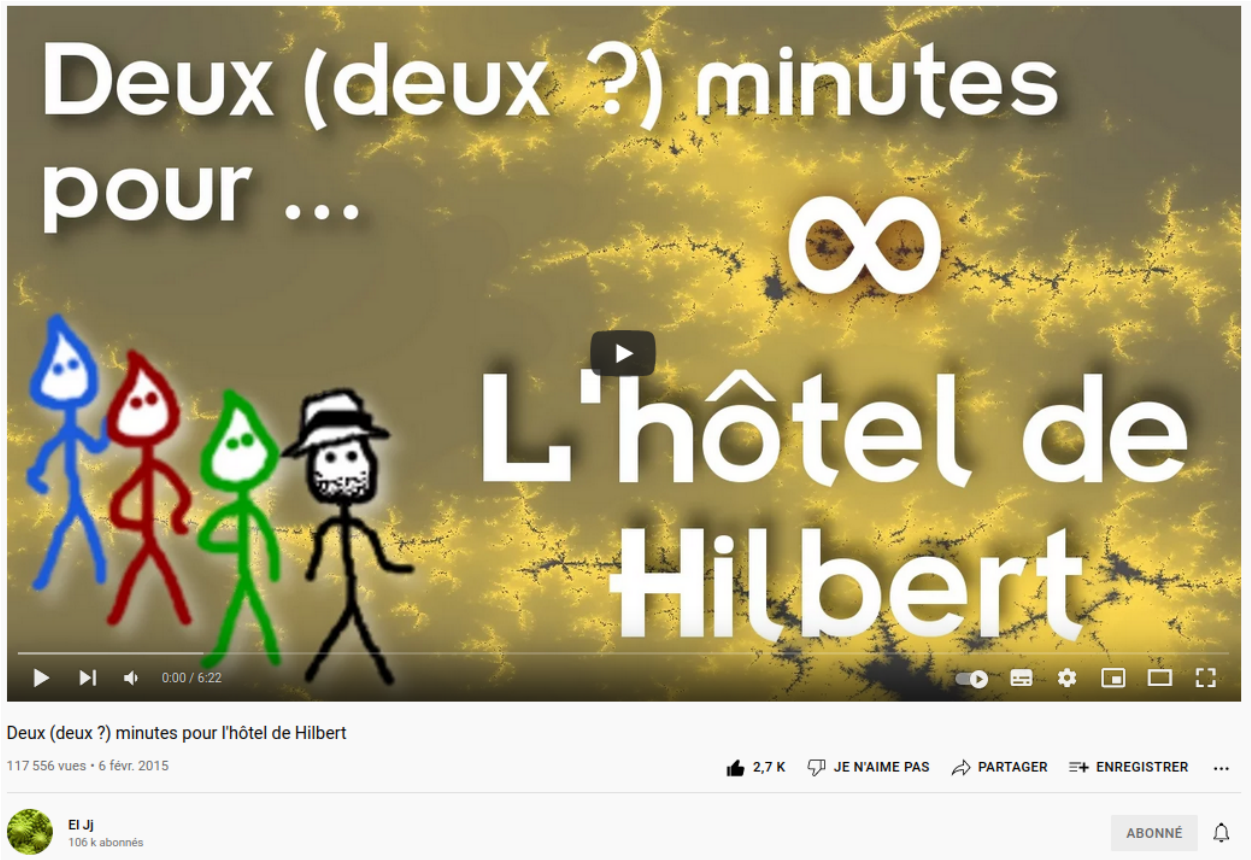
Remarques:

- On parle de performance en *temps* (nombre d'étapes de l'algorithme) et en *espace mémoire* utilisé par l'algorithme pour résoudre le problème en fonction de la *taille de l'entrée*.
- On prouvera donc que le problème à une difficulté bornée (par exemple un temps linéaire ou une mémoire polynomial, etc.) en exhibant un algorithme et en prouvant ses performances.
- Il est très difficile de prouver qu'un problème est réellement difficile. Mais on pourra prouver que tel problème est au moins aussi dur que tel autre (on parle de réduction).

Table of Contents

L'argument de la cardinalité

Vidéo conseillée pour se rafraîchir la mémoire sur les différentes cardinalités:



https://youtu.be/N_cDA6tF-40

L'argument de la cardinalité

On prouvera formellement tous ces résultats dans le TD1.
En résumé:

- On dit que deux ensembles (infinis ou finis) sont de même cardinalité (taille) s'il existe une bijection entre les deux.
- Il existe plusieurs cardinalités infinies différentes:
  - $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, Σ^* (l'ensemble des mots finis sur un alphabet fini), l'ensemble des algorithmes (écrit en pseudo-langage ou dans le langage de programmation que vous voulez) ont tous la même cardinalité: ils sont *dénombrables*.
  - $\mathbb{R}, [0,1]$, l'ensemble des fonctions $f : \mathbb{N} \to \mathbb{N}$ ou $f : \mathbb{N} \to \{0,1\}$, l'ensemble des problèmes informatiques sont tous de même cardinalité: ils sont *indénombrables*.
  - Grace à l'argument de la diagonale de Cantor, on peut prouver que $|\mathbb{N}| < |\mathbb{R}|$: il y a beaucoup plus de problèmes que d'algorithmes et donc il existe des fonctions non-calculables.
  - Pire: la très grande majorité des fonctions sont non-calculables.

L'argument de la cardinalité

- Ce simple argument prouve que la très grande majorité des fonctions sont non-calculables.
- Mais ça ne nous dit pas qu'il y a des fonctions *intéressantes* qui sont non-calculables!
- À ce stade, on pourrait même penser qu'aucune fonction qu'on pourrait décrire formellement n'est non-calculable.
- C'est pourtant faux: on verra qu'il y a des fonctions très simples et relativement naturel qui sont non-calculables.

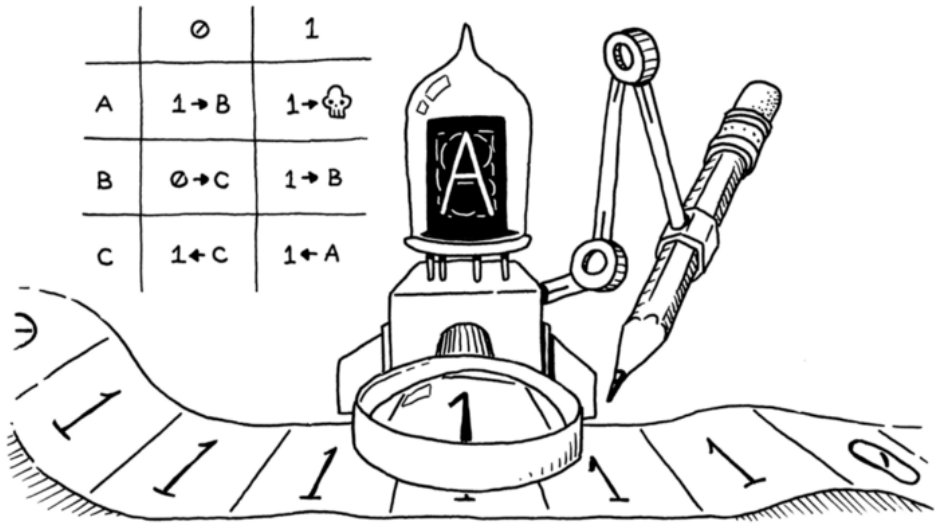
Table of Contents

Machine de Turing: À quoi ça sert?

Alan Turing (1912-1954)

- Pour montrer qu'une fonction n'est pas calculable, il faut d'abord définir l'ensemble des algorithmes (car cela définit l'ensemble de ce qui est calculable).
- Mais il est difficile de décrire rigoureusement tout ce qu'un algorithme peut-être. Imaginez devoir définir mathématiquement votre langage de programmation préféré dans ses moindres détails...
- L'intérêt des machines de Turing proposé par Alan Turing en 1936 est qu'elles définissent les algorithmes de façon intuitive et simple!

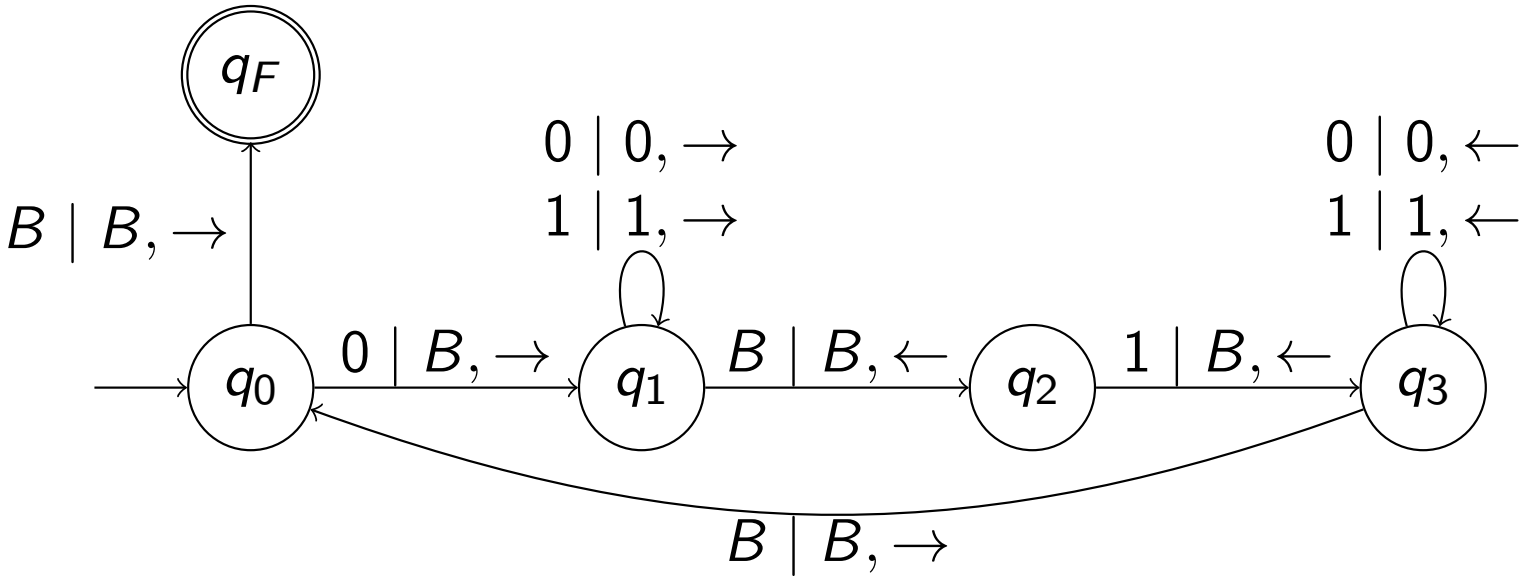
Machine de Turing: À quoi ça sert?

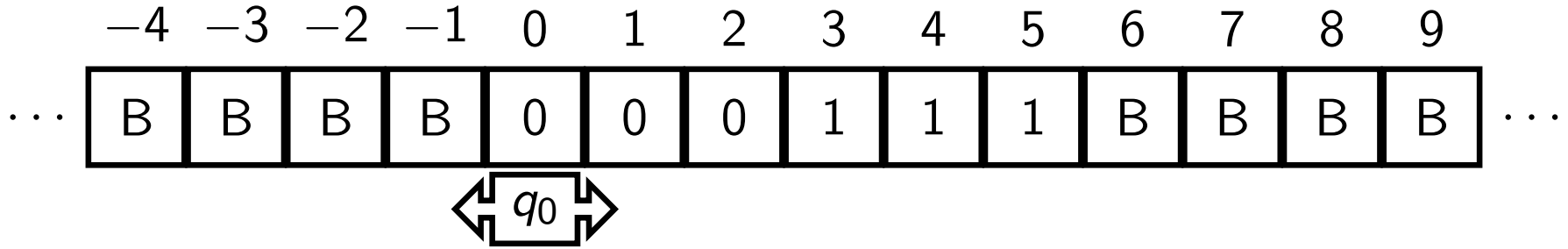


L'idée d'Alan Turing est inspirée du calculateur humain devant sa feuille:

- feuilles découpées en cases : ruban ;
- crayon posé sur une case : tête de lecture/écriture, déplacement ;
- l'opérateur dispose d'une mémoire finie (son cerveau) : états.

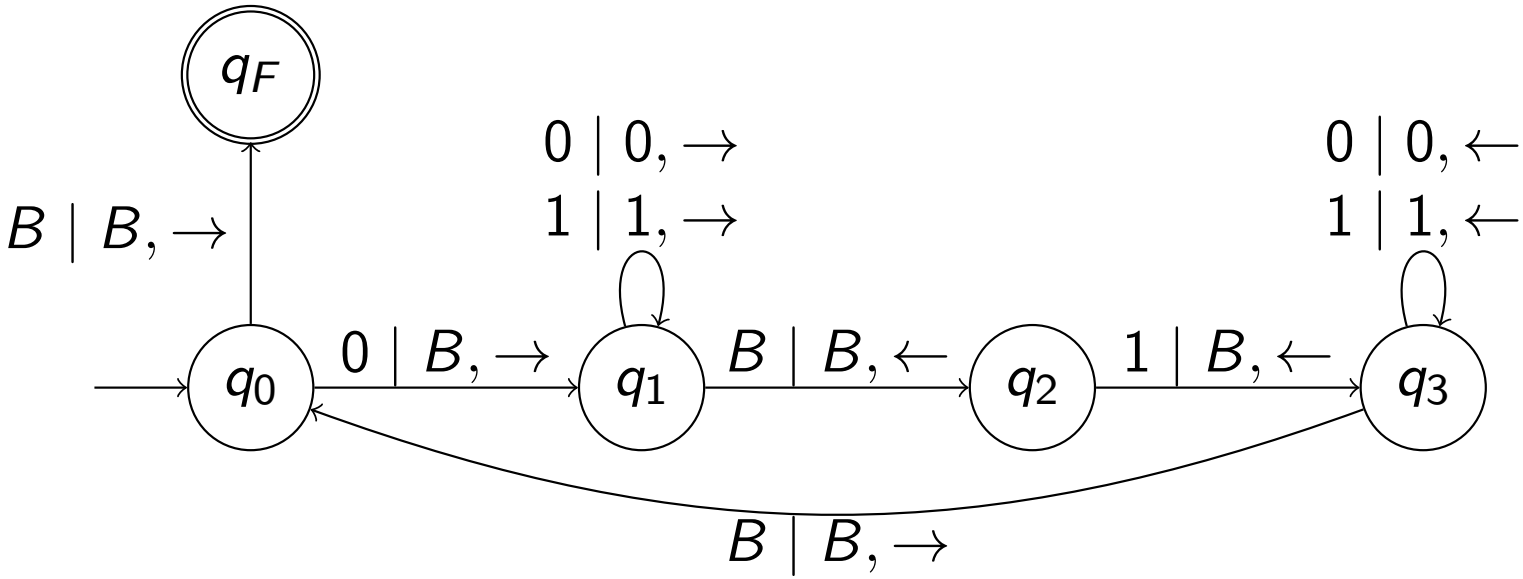
Exemple de machine de Turing

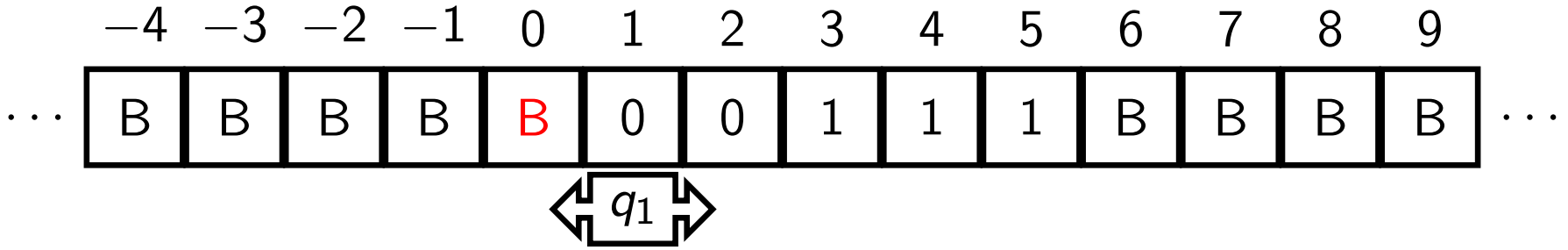




Étape:1

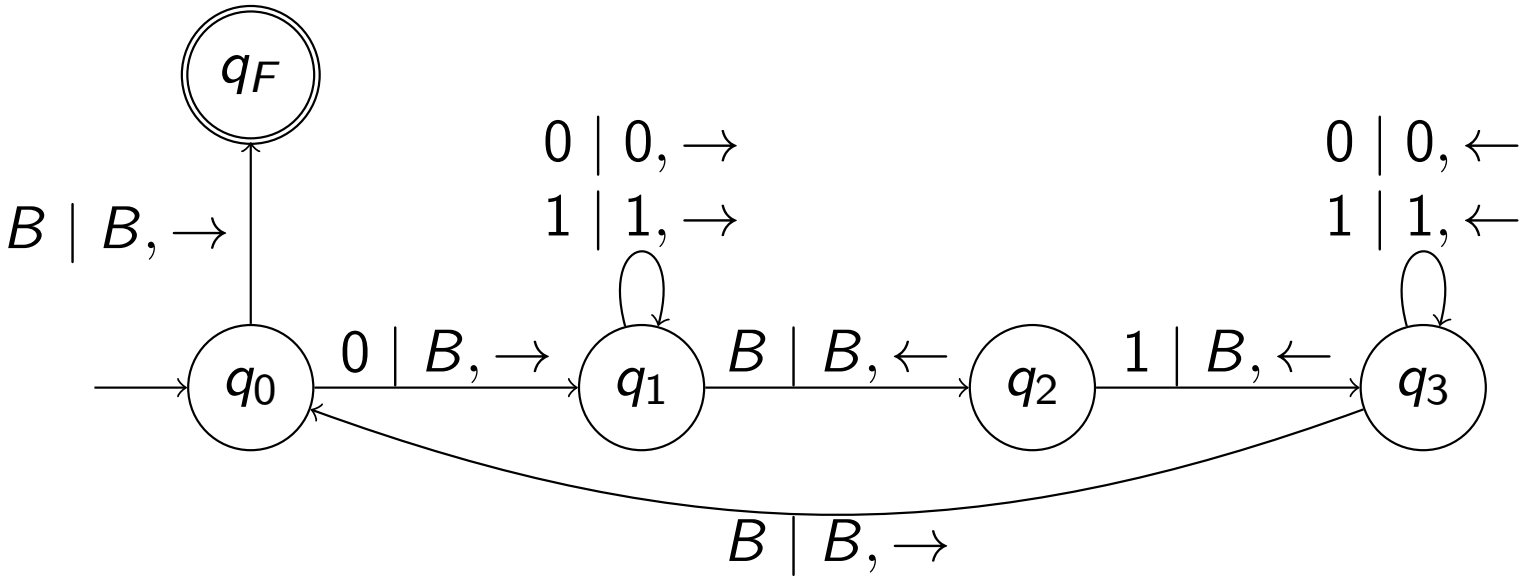
Exemple de machine de Turing

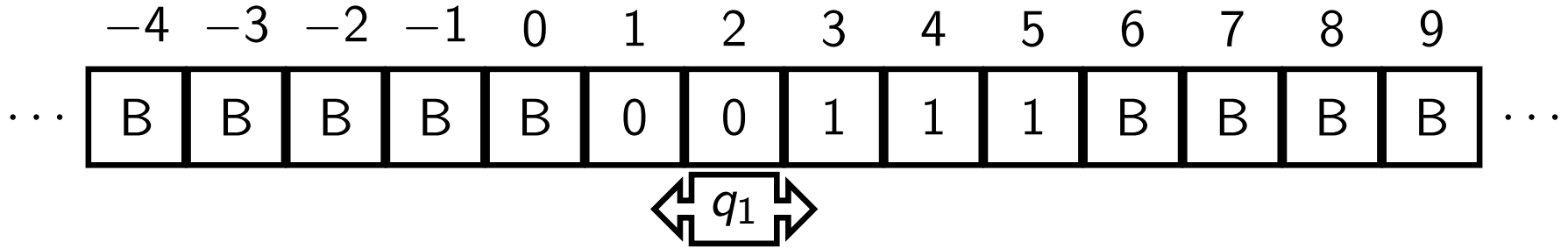




Étape:2

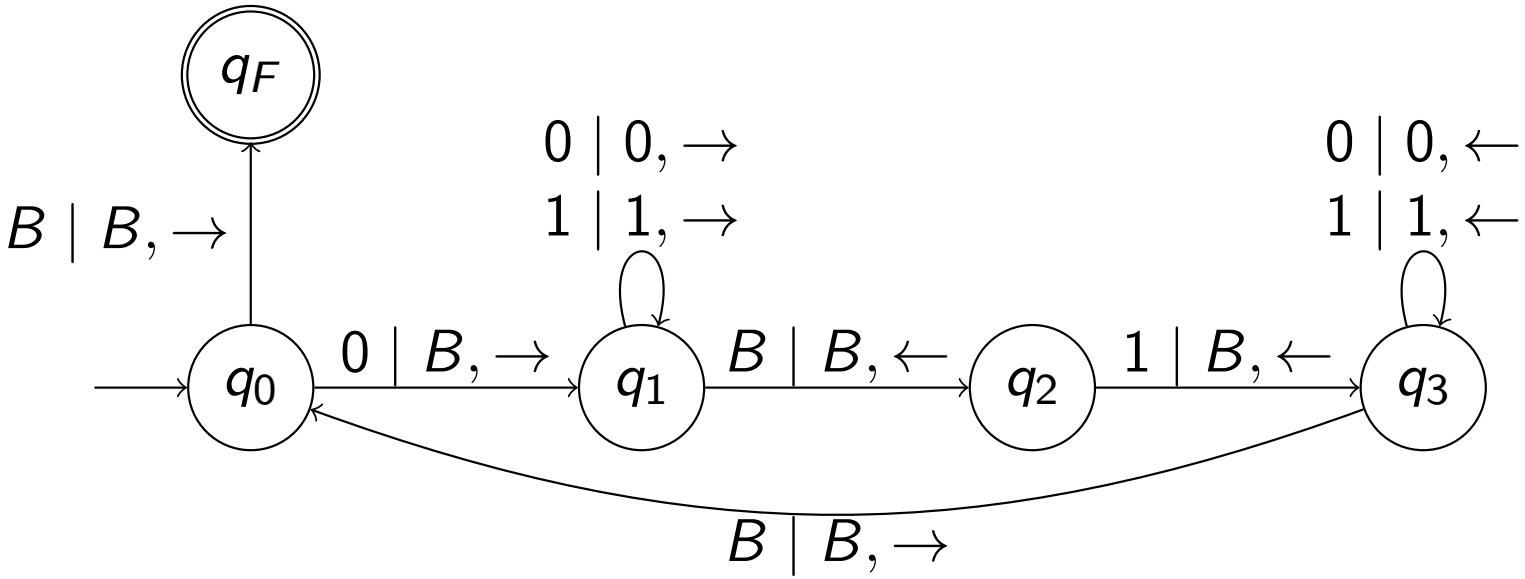
Exemple de machine de Turing

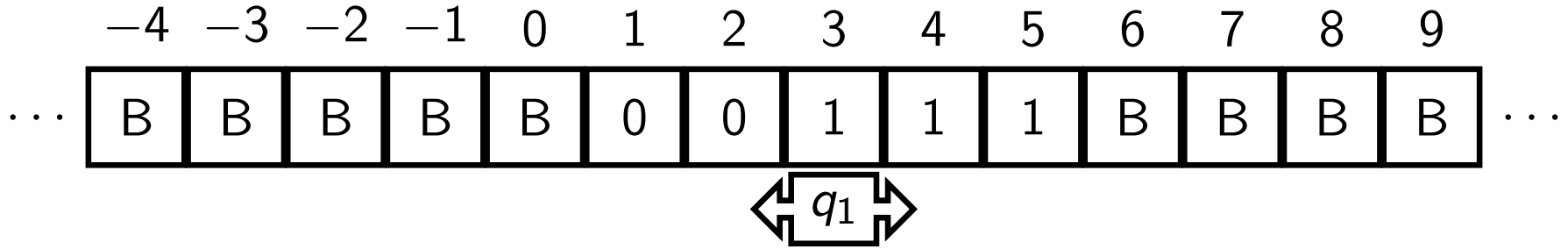




Étape:3

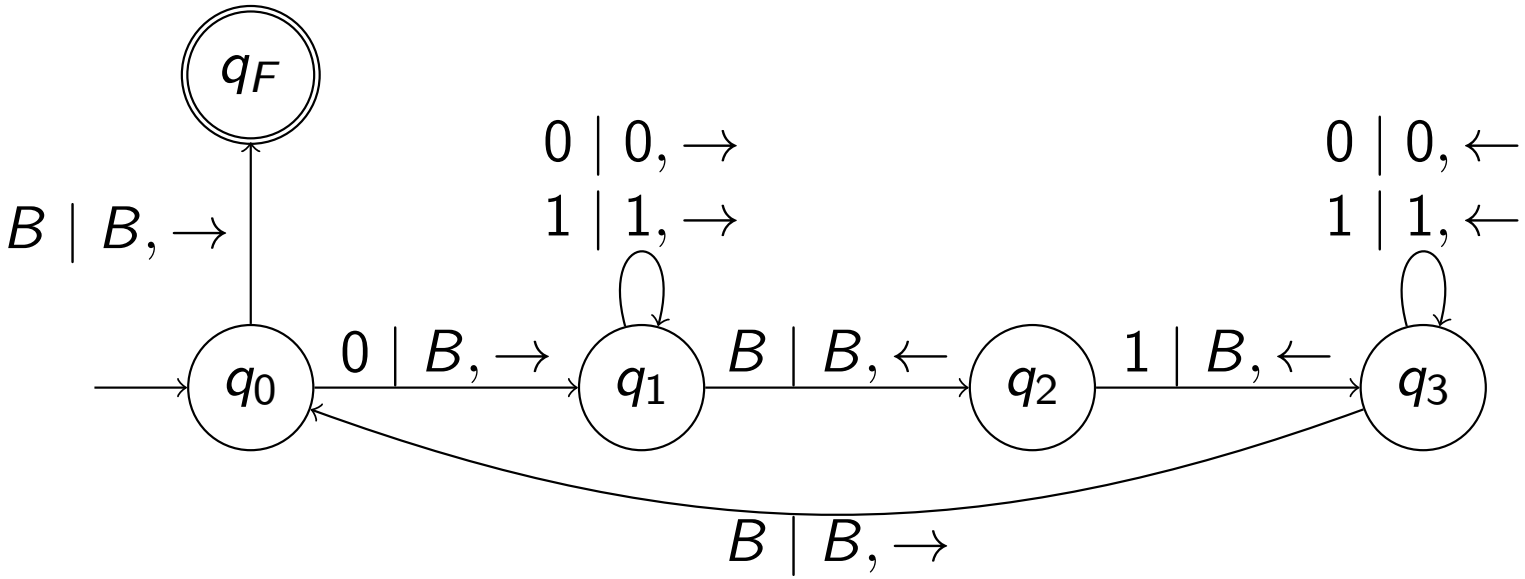
Exemple de machine de Turing

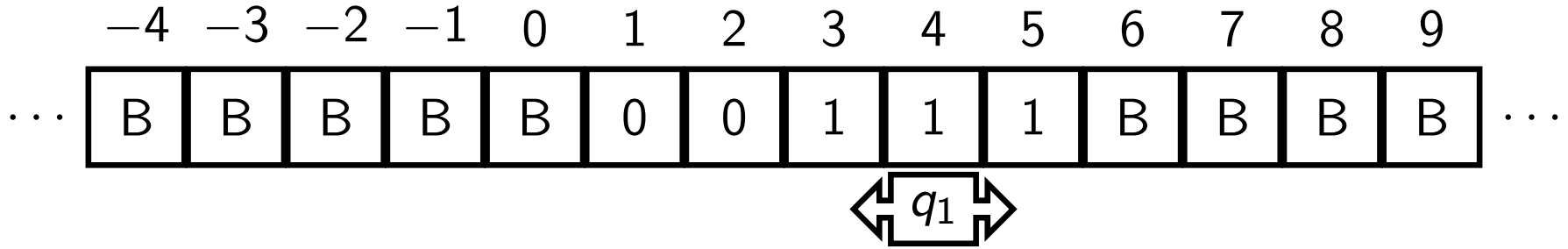




Étape:4

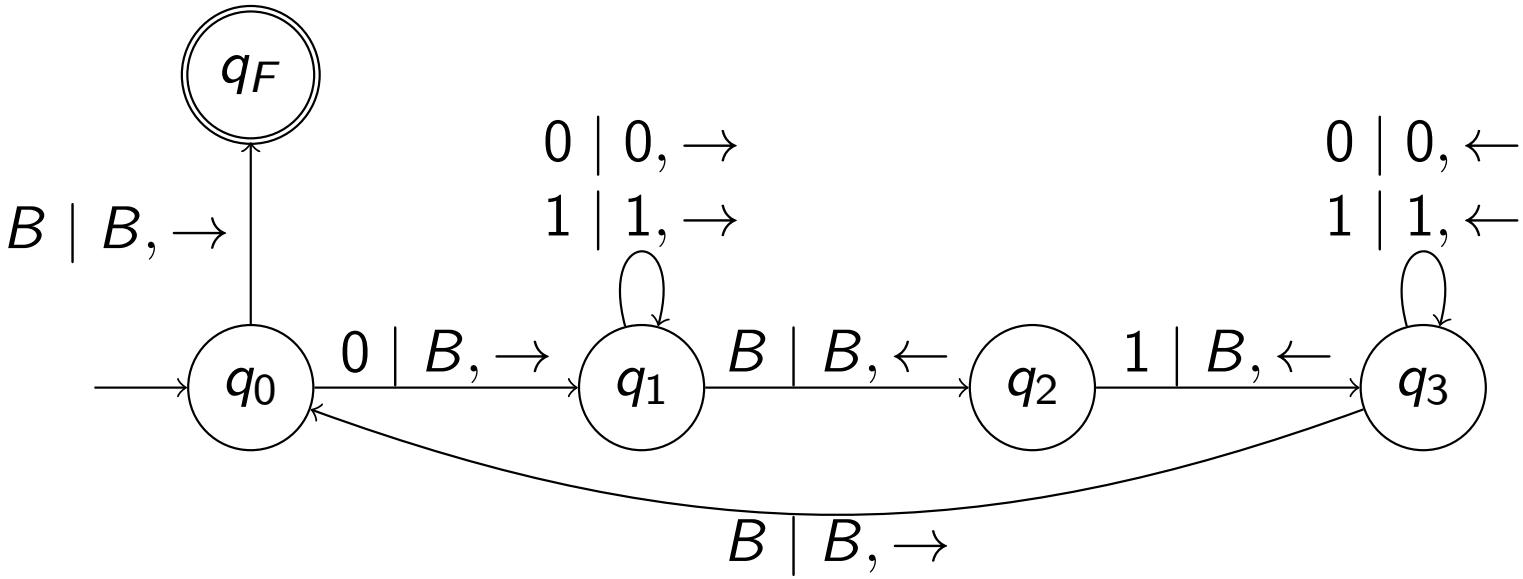
Exemple de machine de Turing

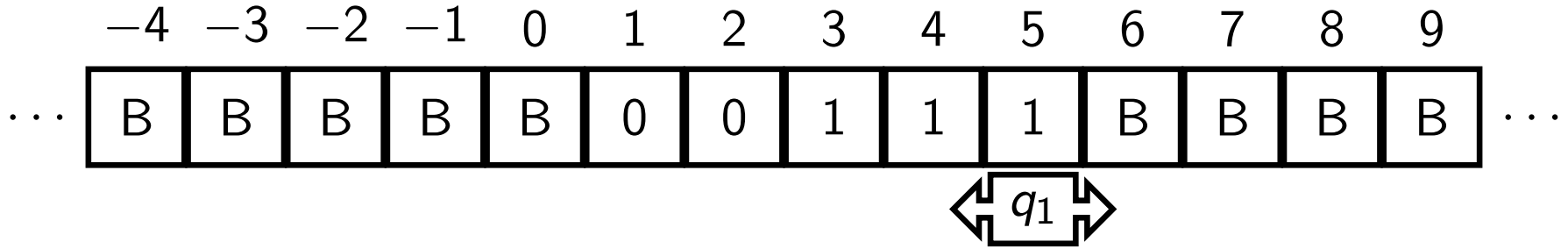




Étape:5

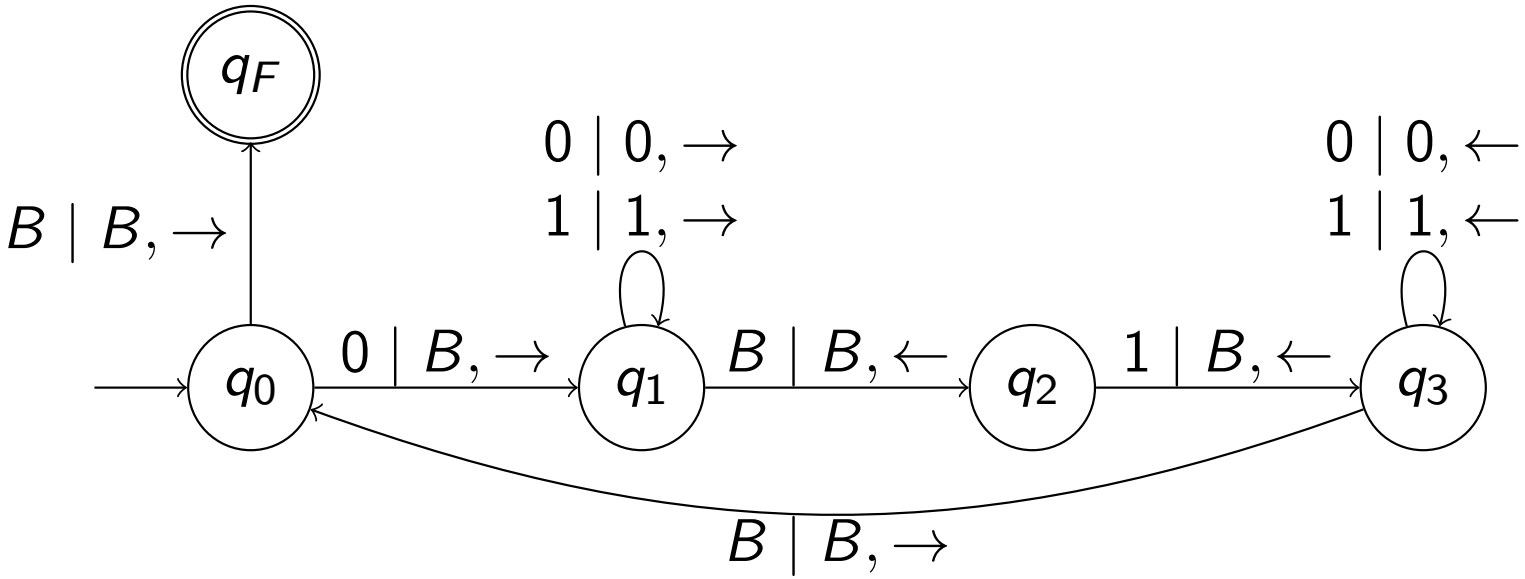
Exemple de machine de Turing

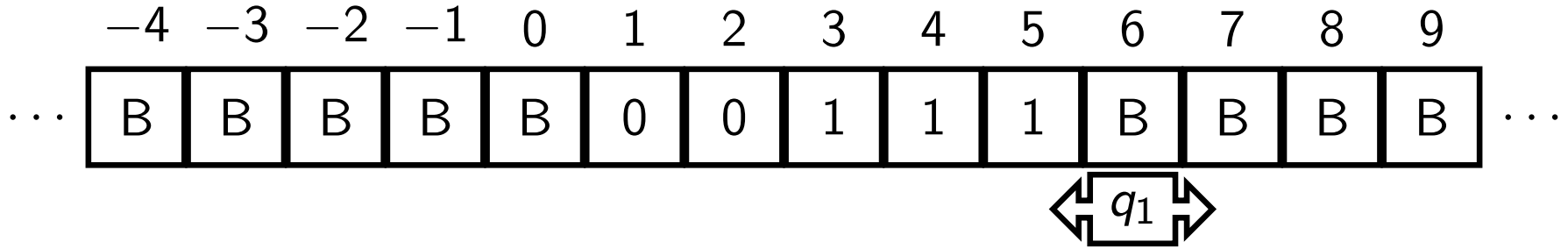




Étape:6

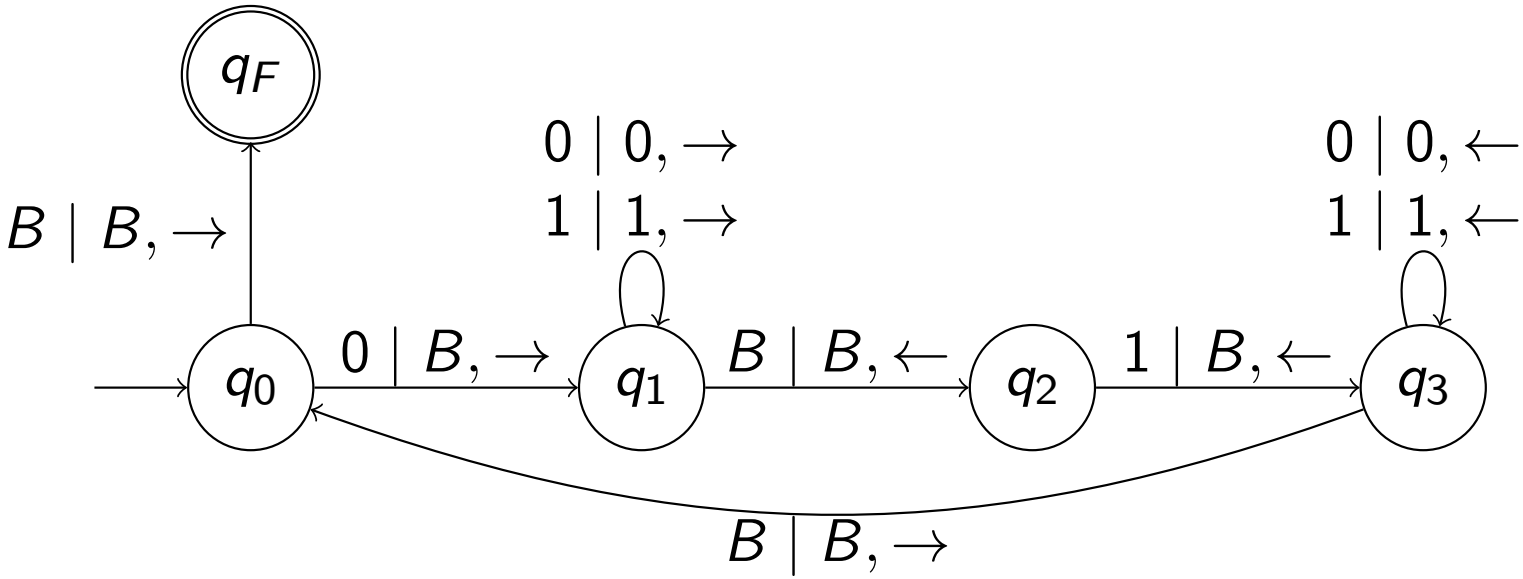
Exemple de machine de Turing

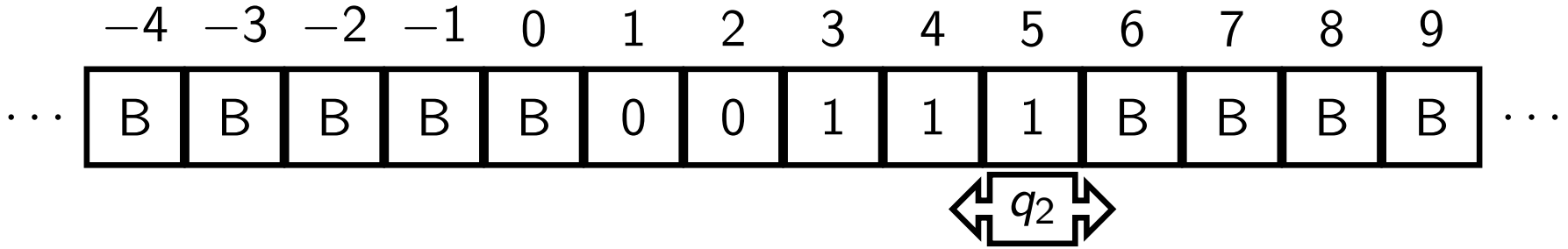




Étape:7

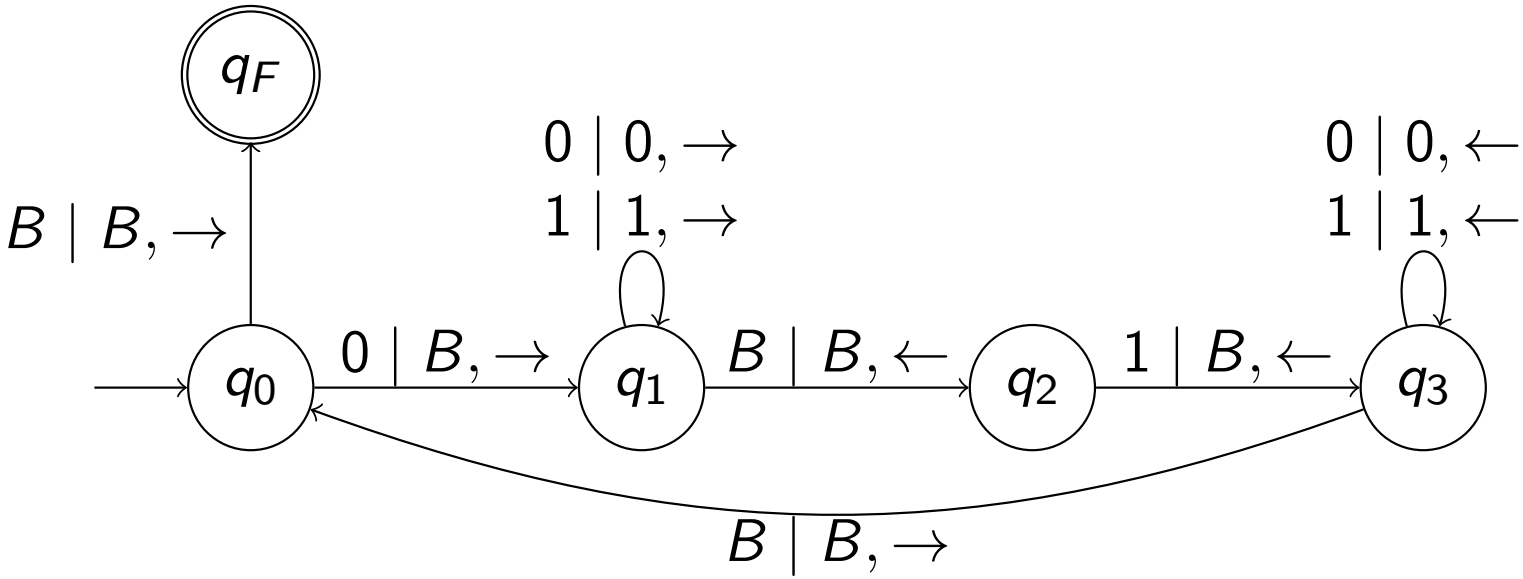
Exemple de machine de Turing

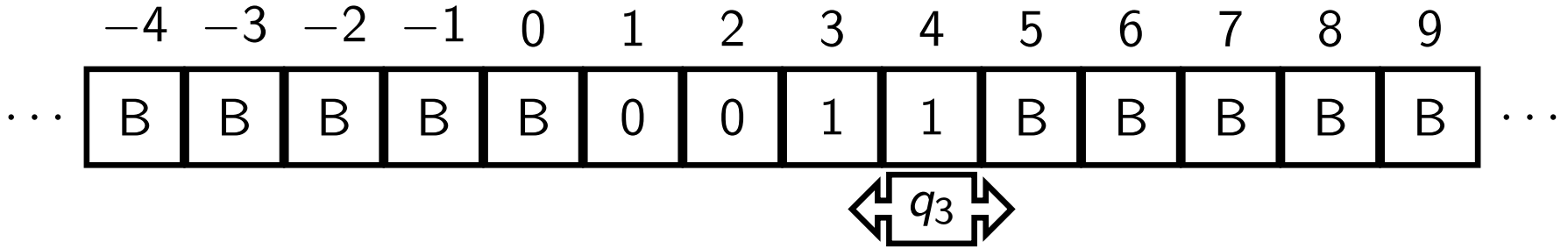




Étape:8

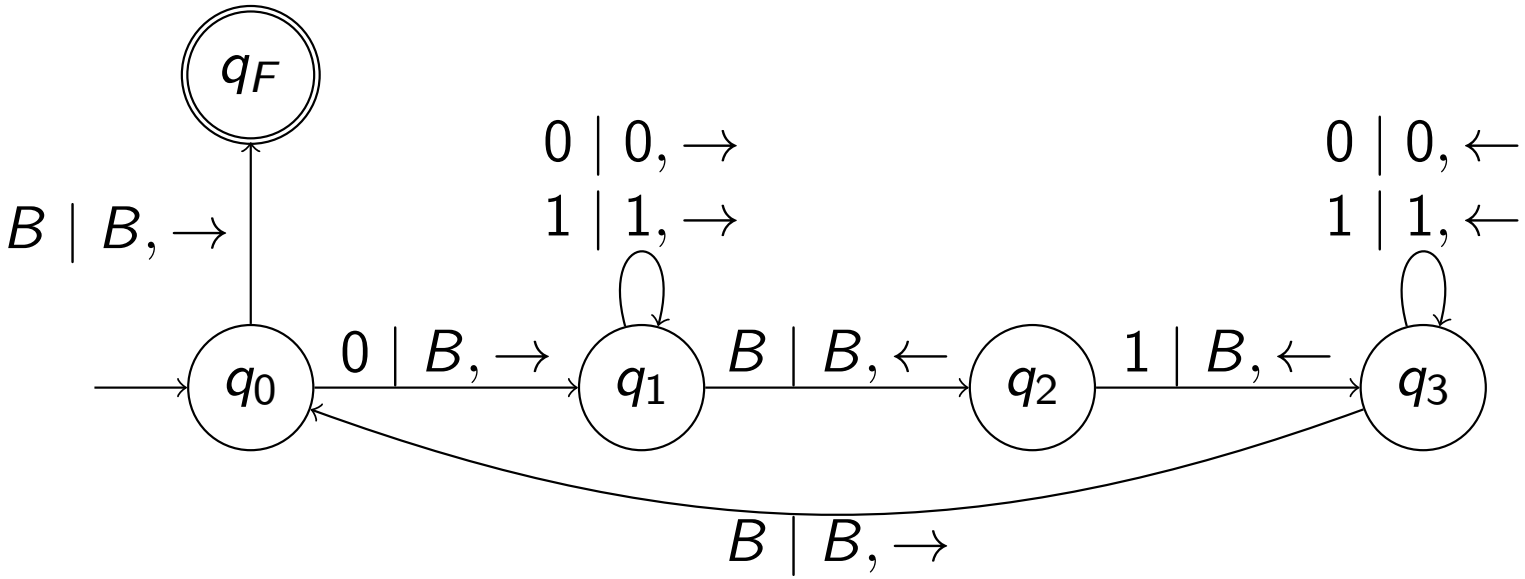
Exemple de machine de Turing

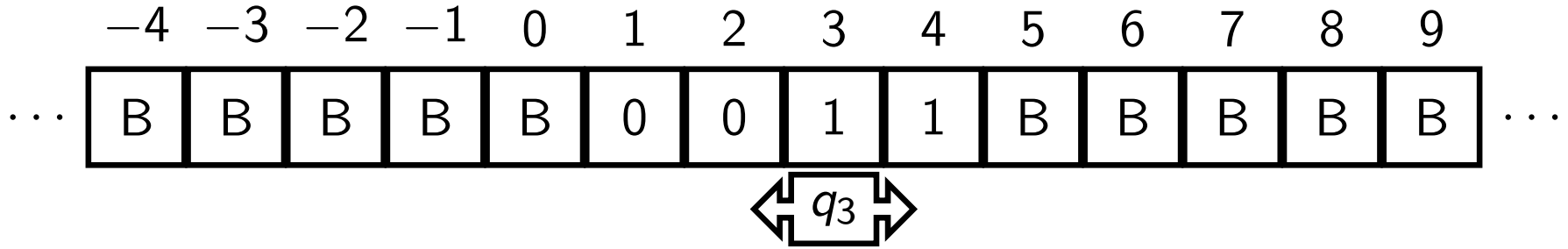




Étape:9

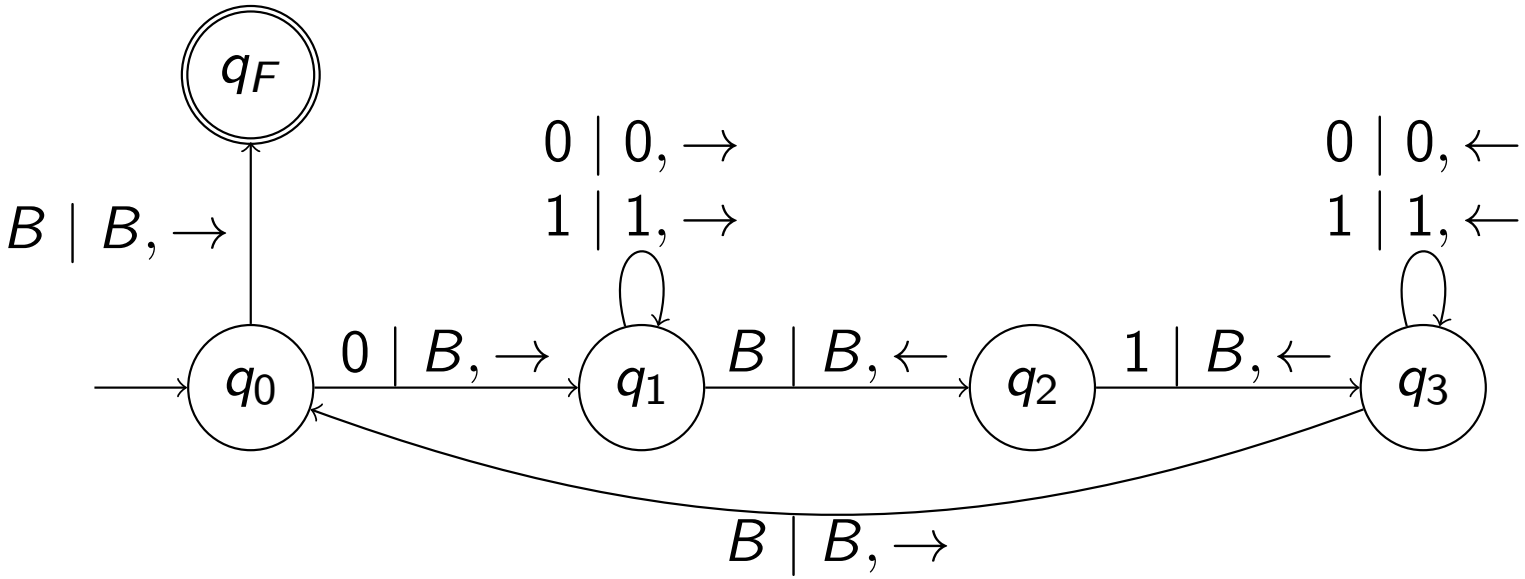
Exemple de machine de Turing

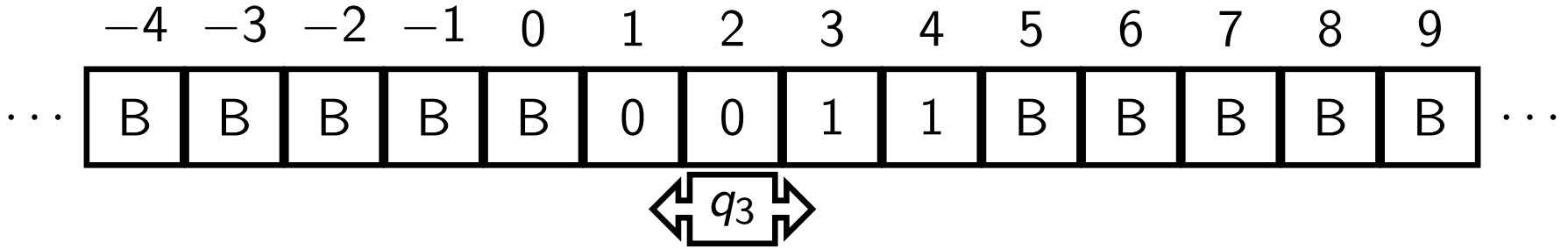




Étape:10

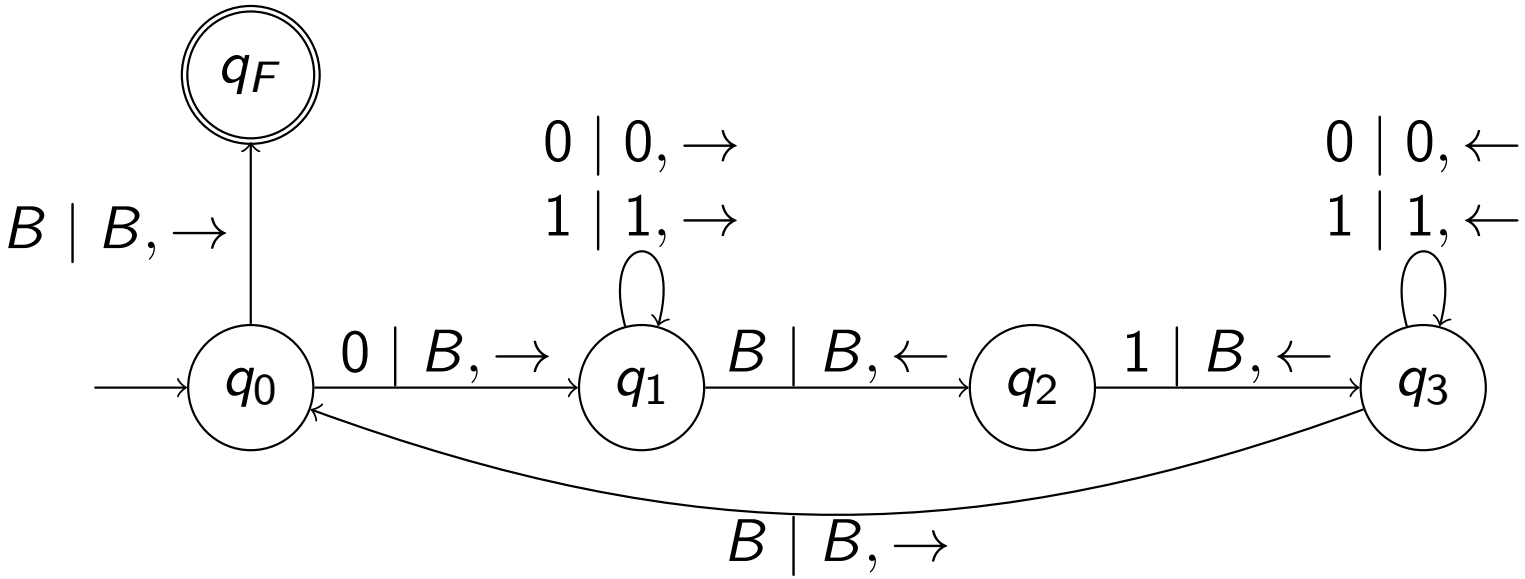
Exemple de machine de Turing

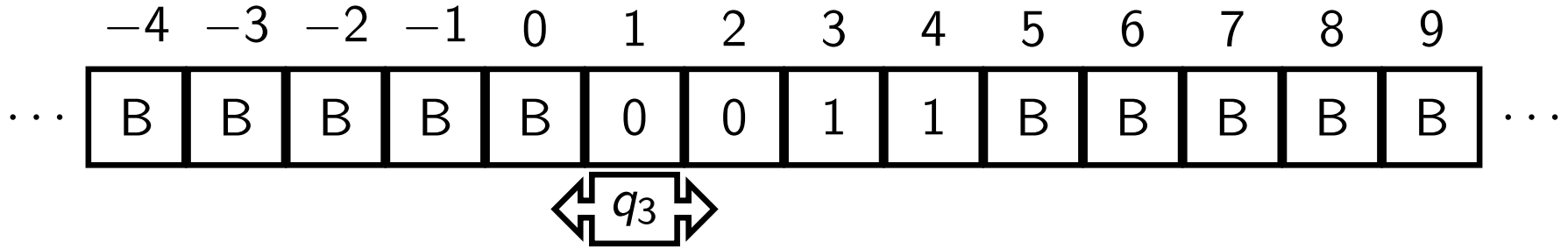




Étape:11

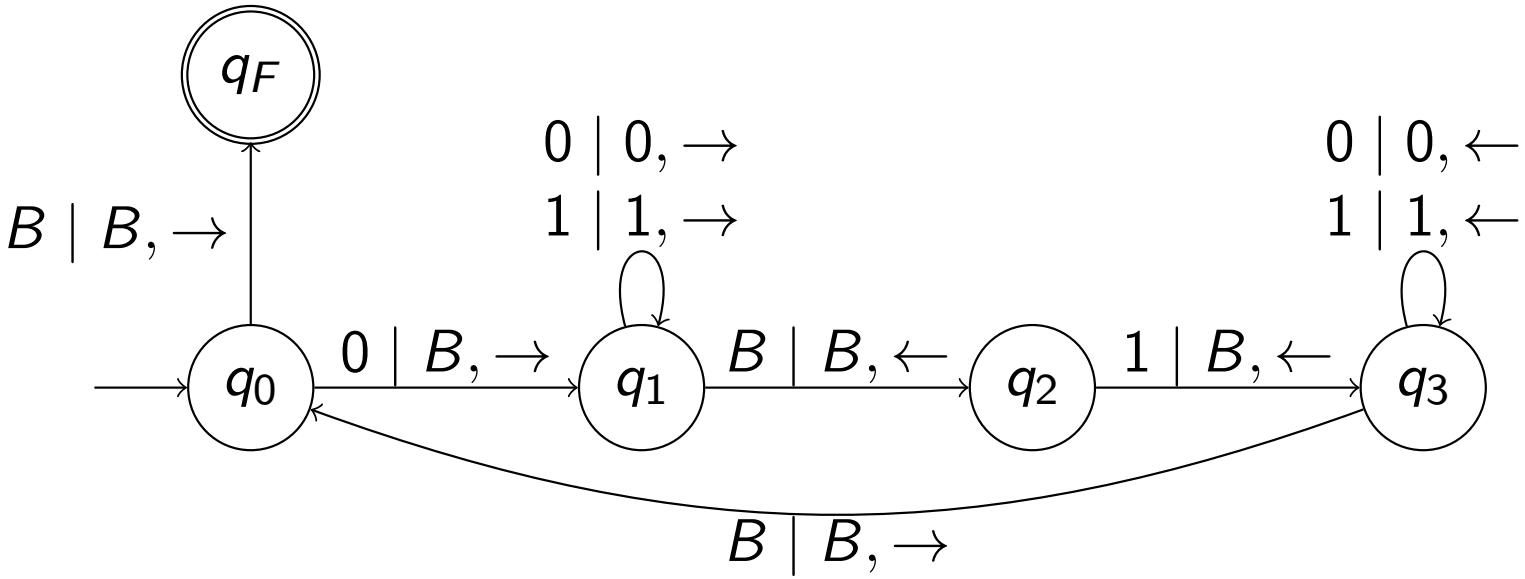
Exemple de machine de Turing

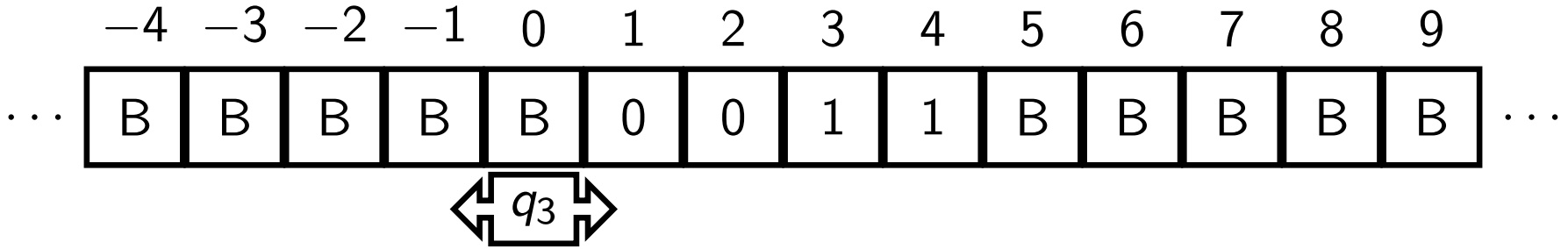




Étape:12

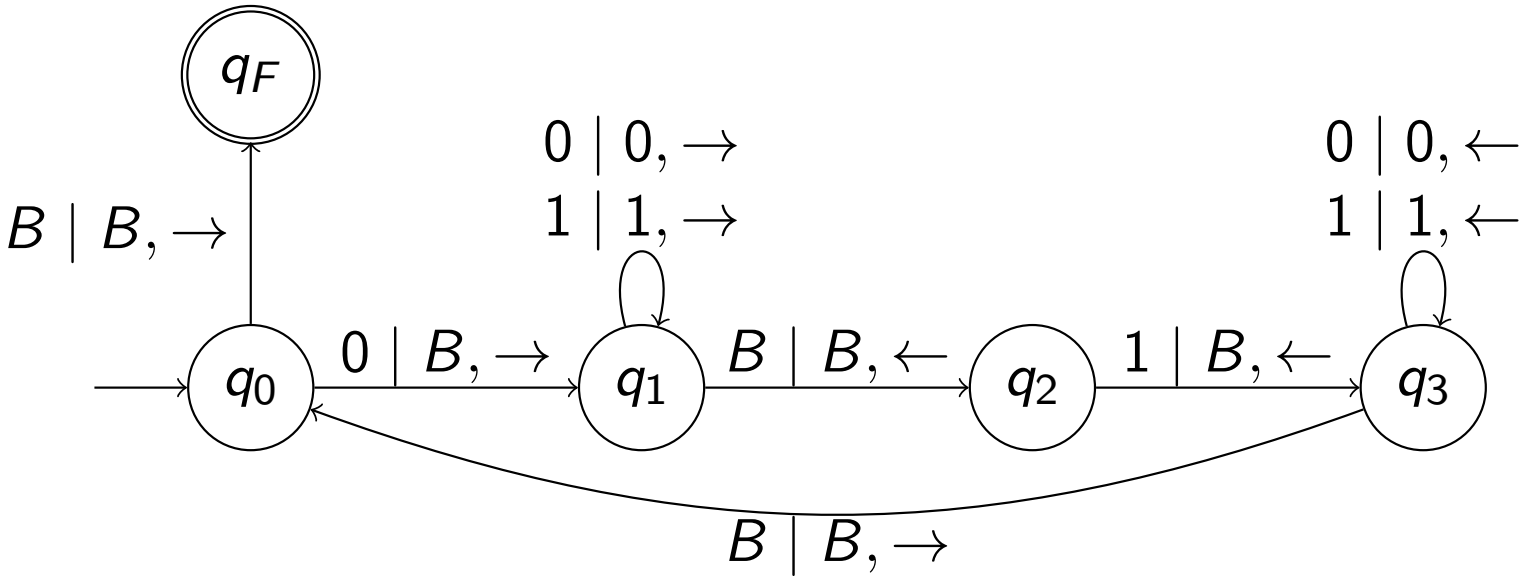
Exemple de machine de Turing

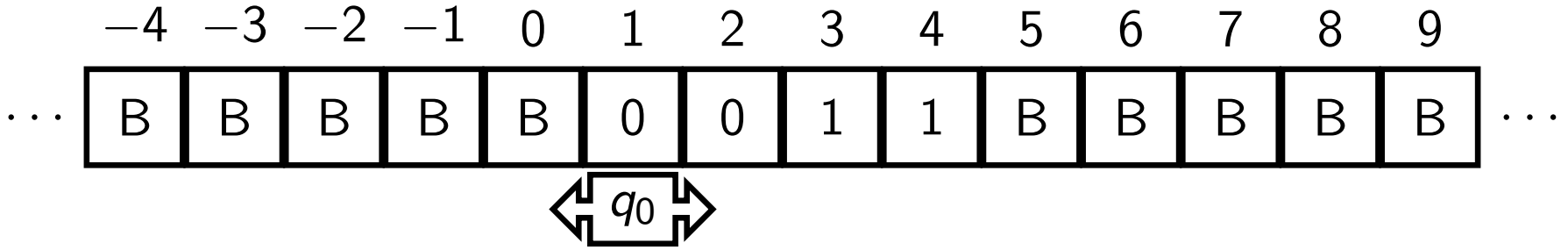




Étape:13

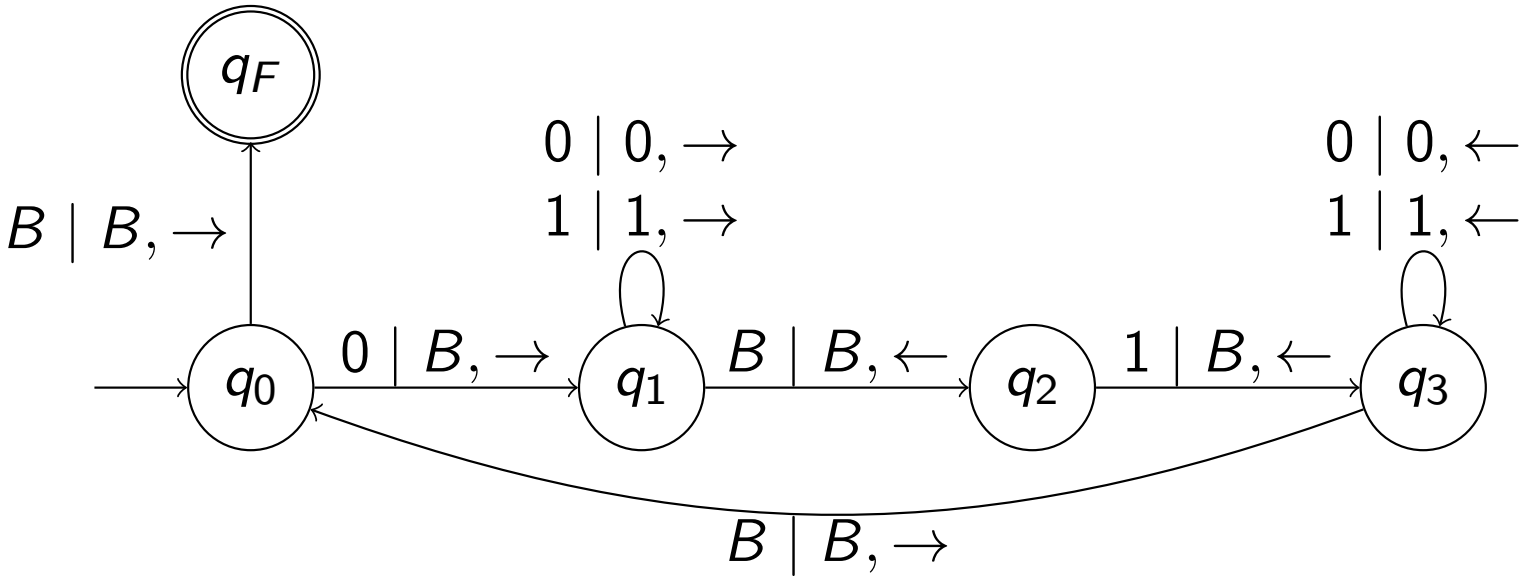
Exemple de machine de Turing

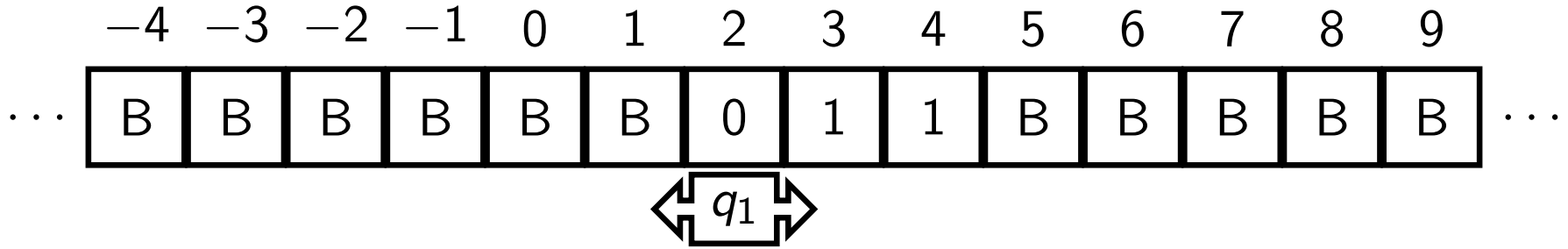




Étape:14

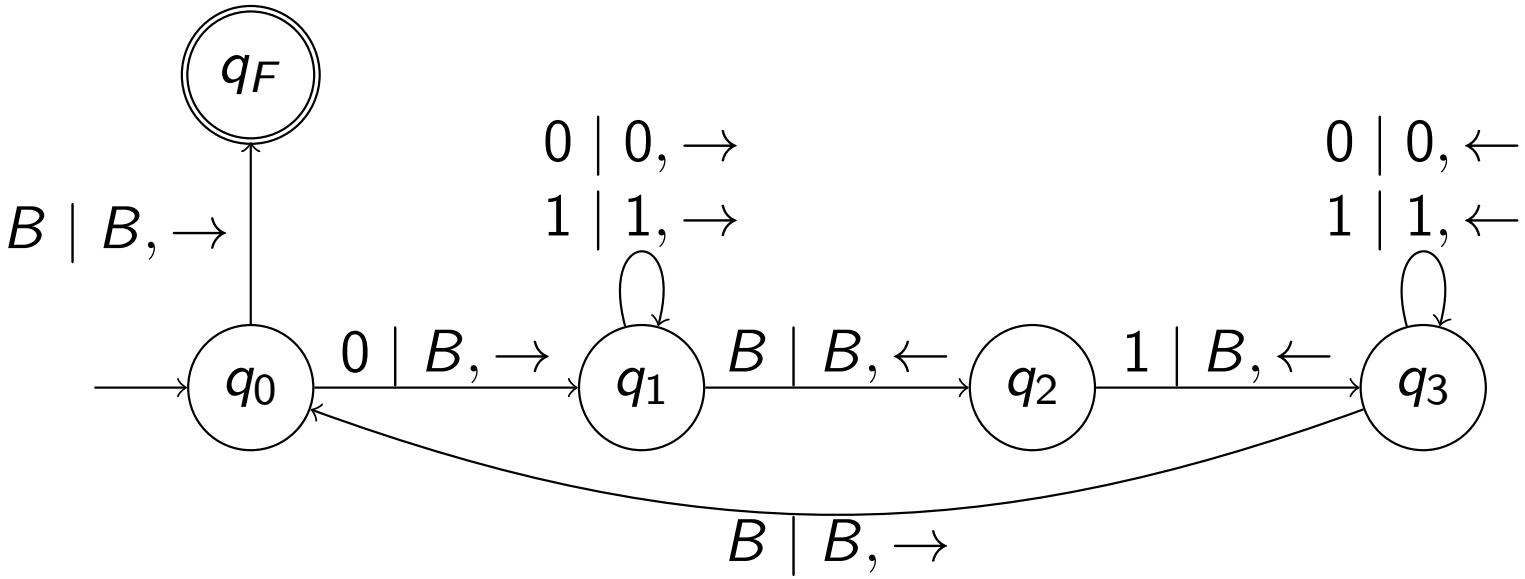
Exemple de machine de Turing

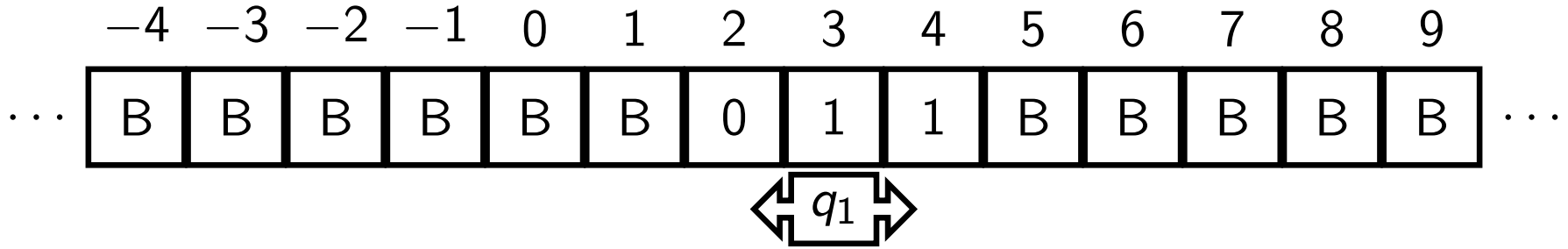




Étape:15

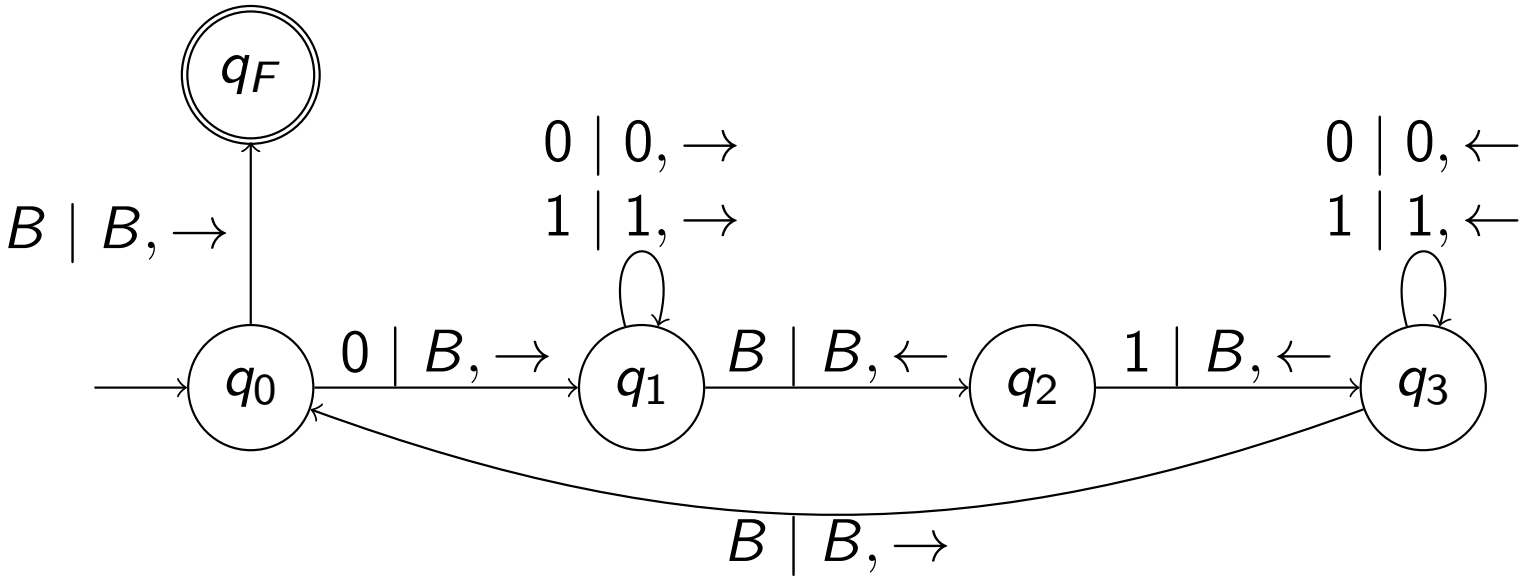
Exemple de machine de Turing

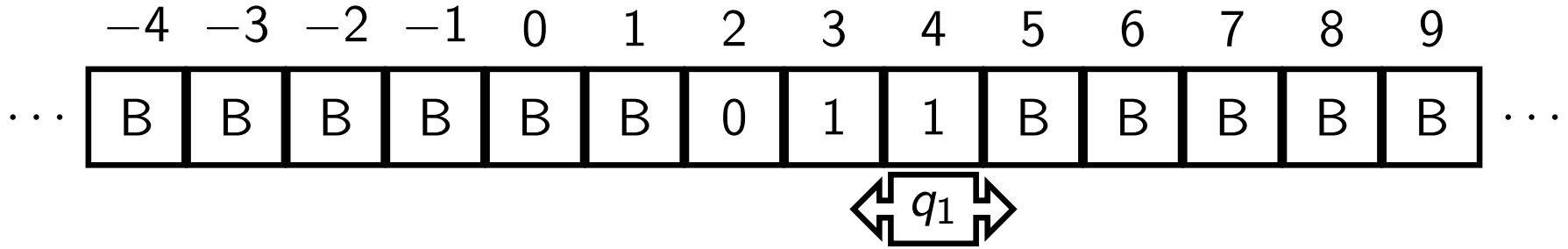




Étape:16

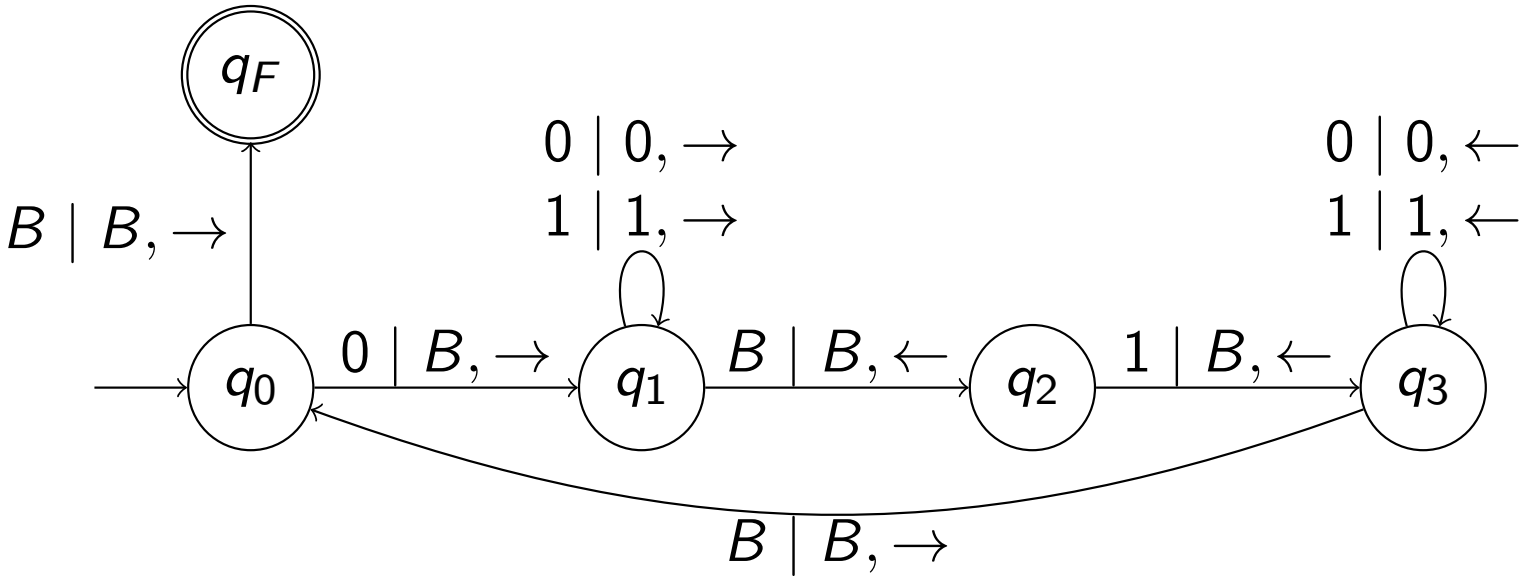
Exemple de machine de Turing

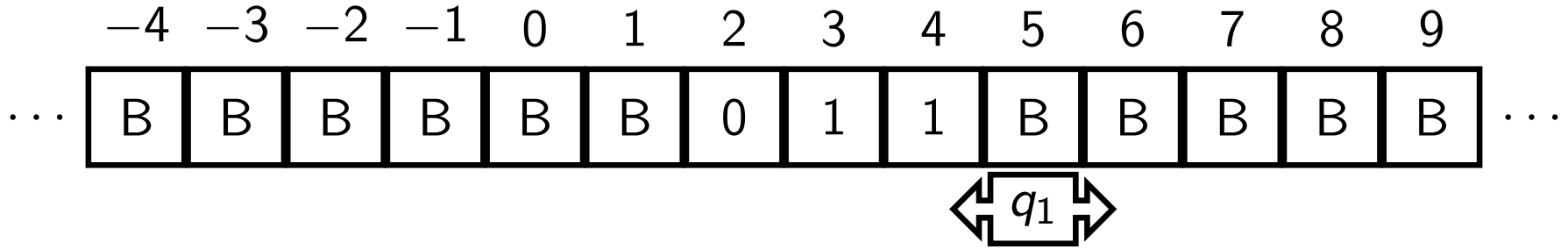




Étape:17

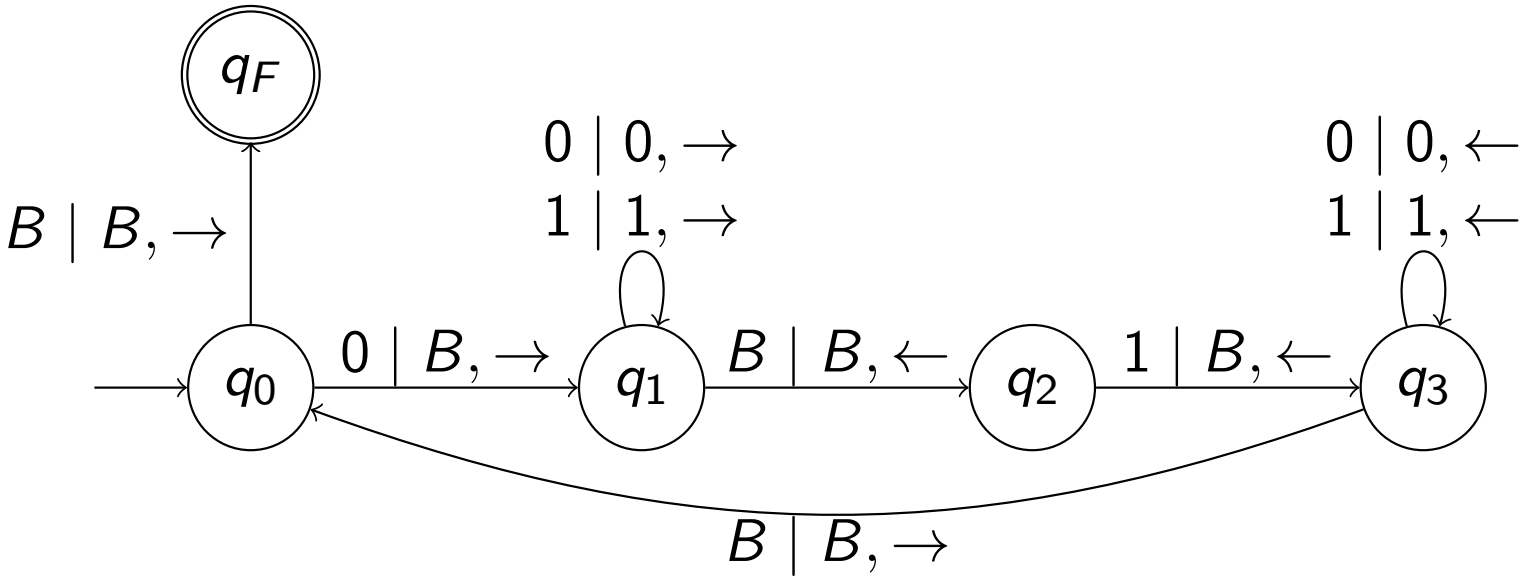
Exemple de machine de Turing

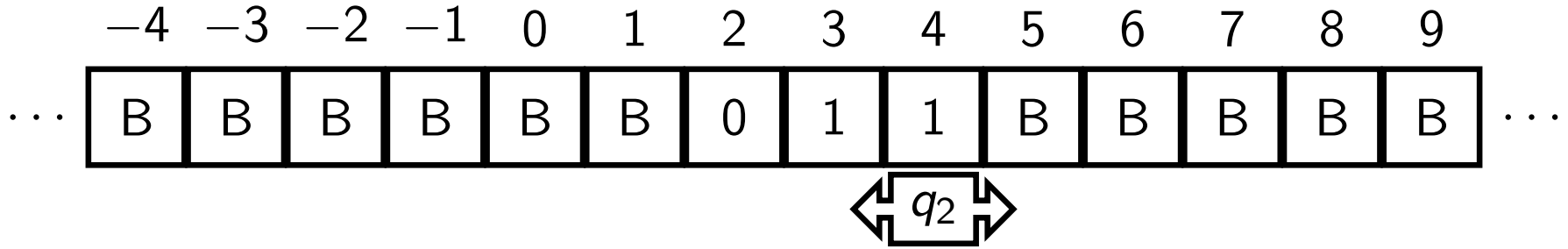




Étape:18

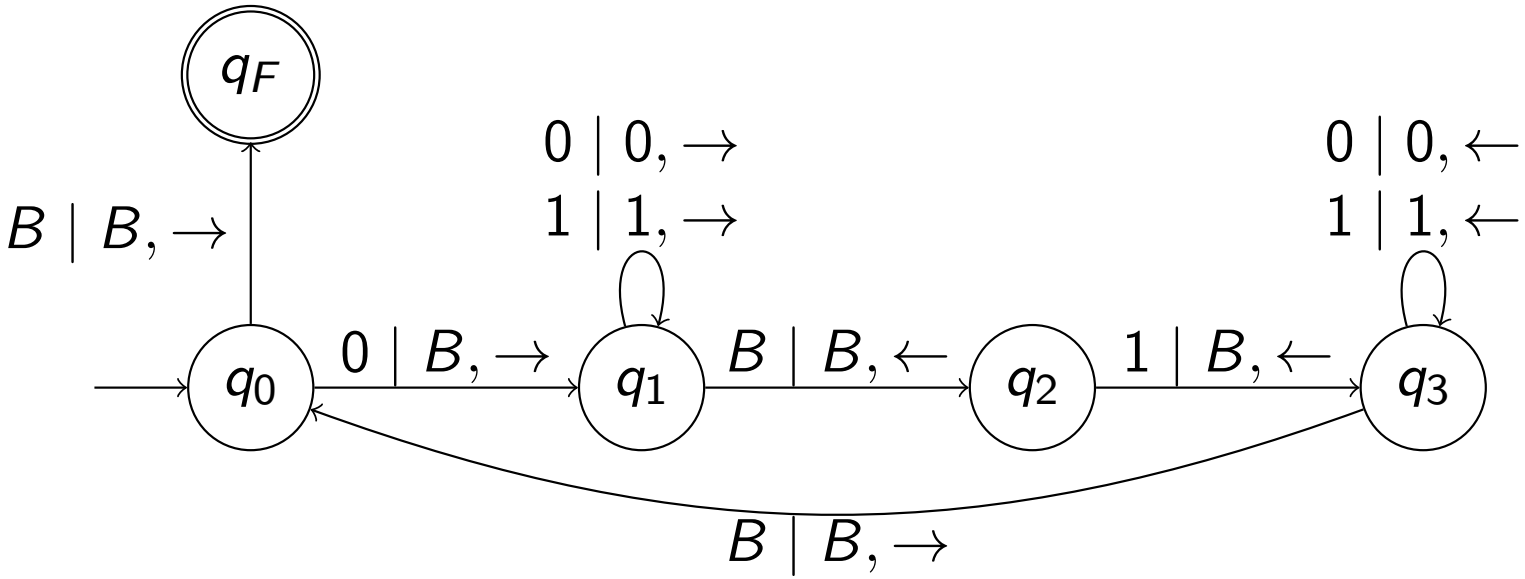
Exemple de machine de Turing

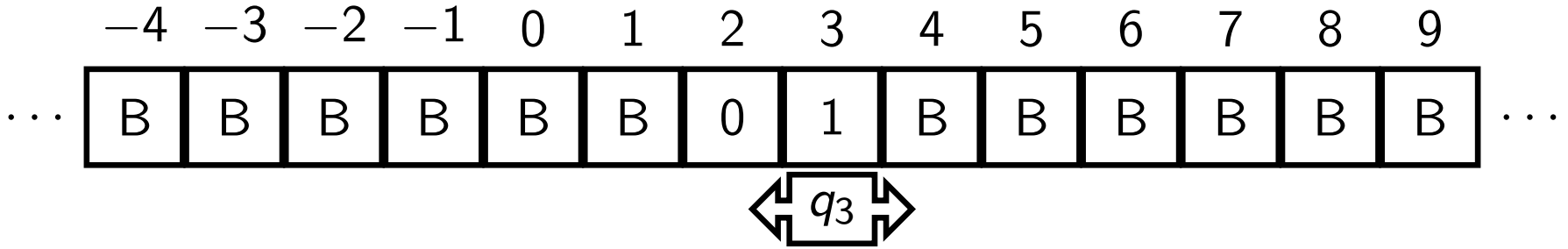




Étape:19

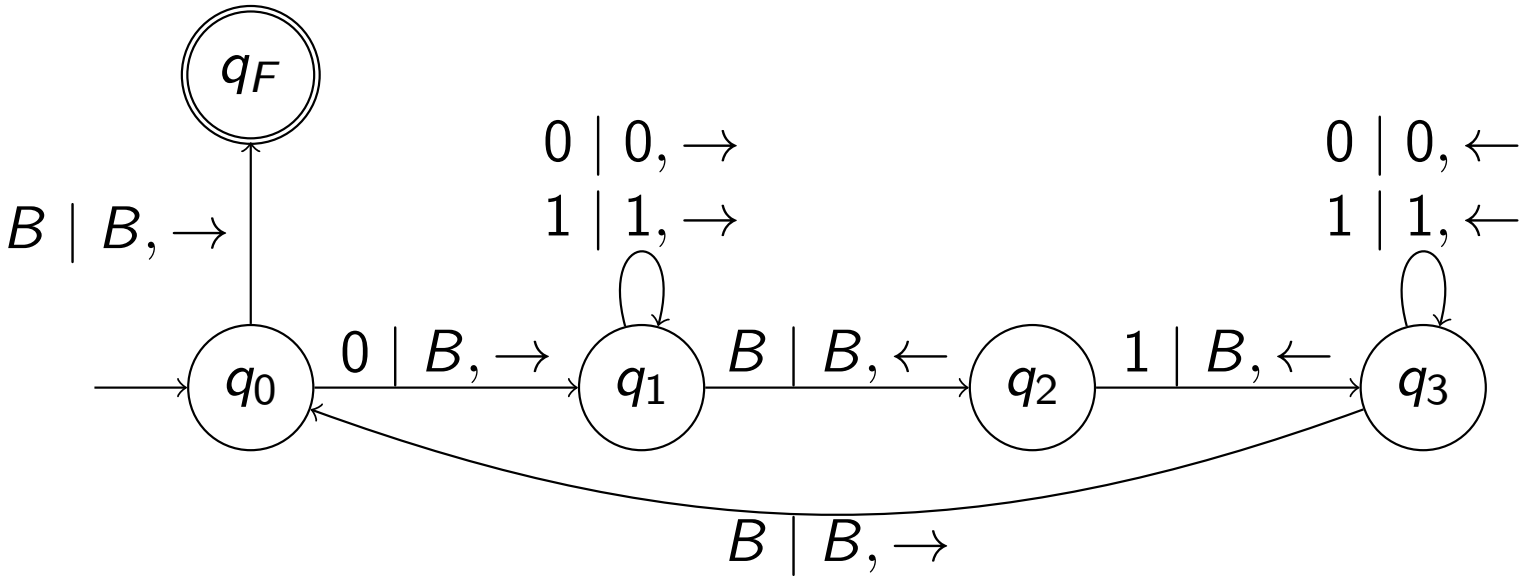
Exemple de machine de Turing

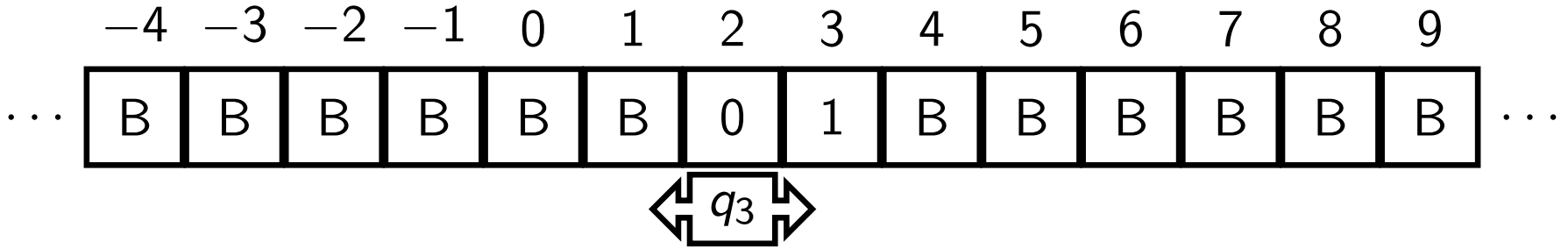




Étape:20

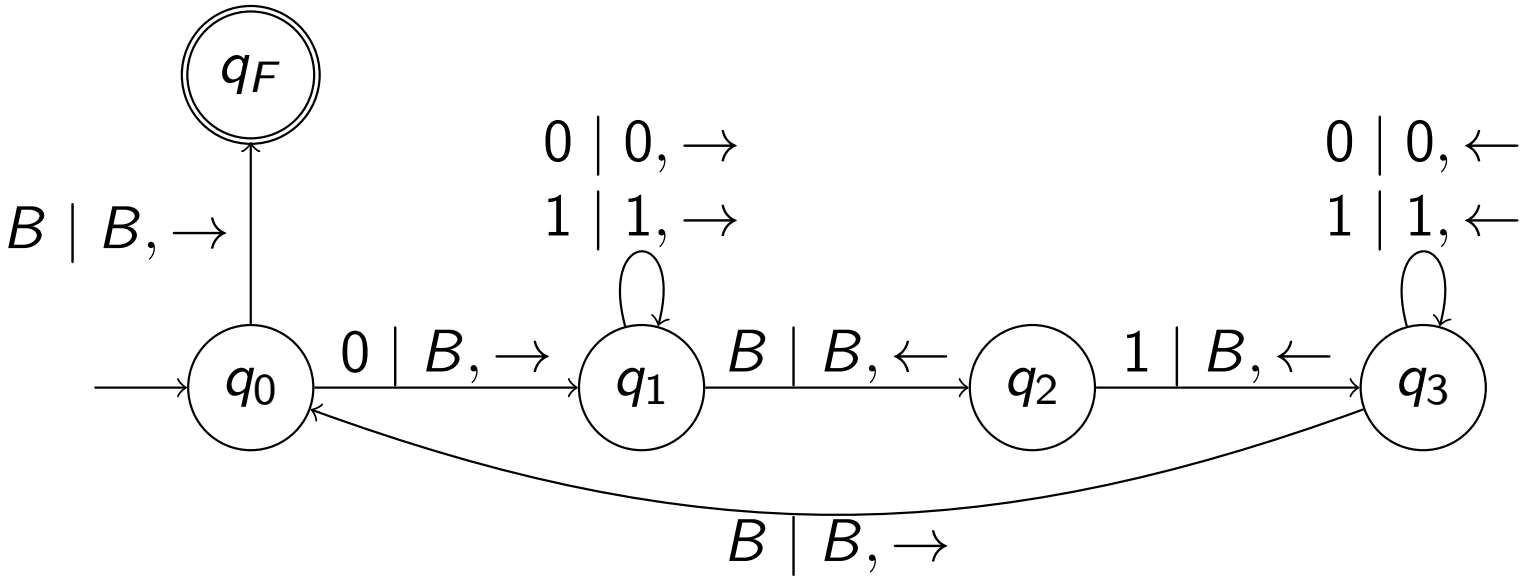
Exemple de machine de Turing

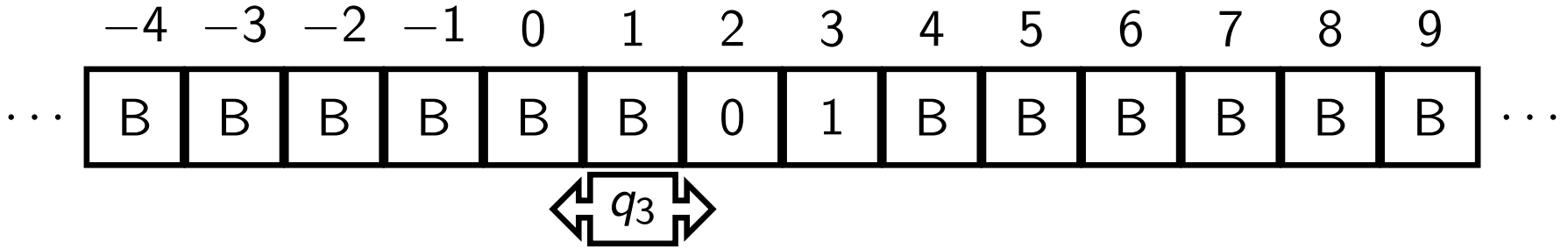




Étape:21

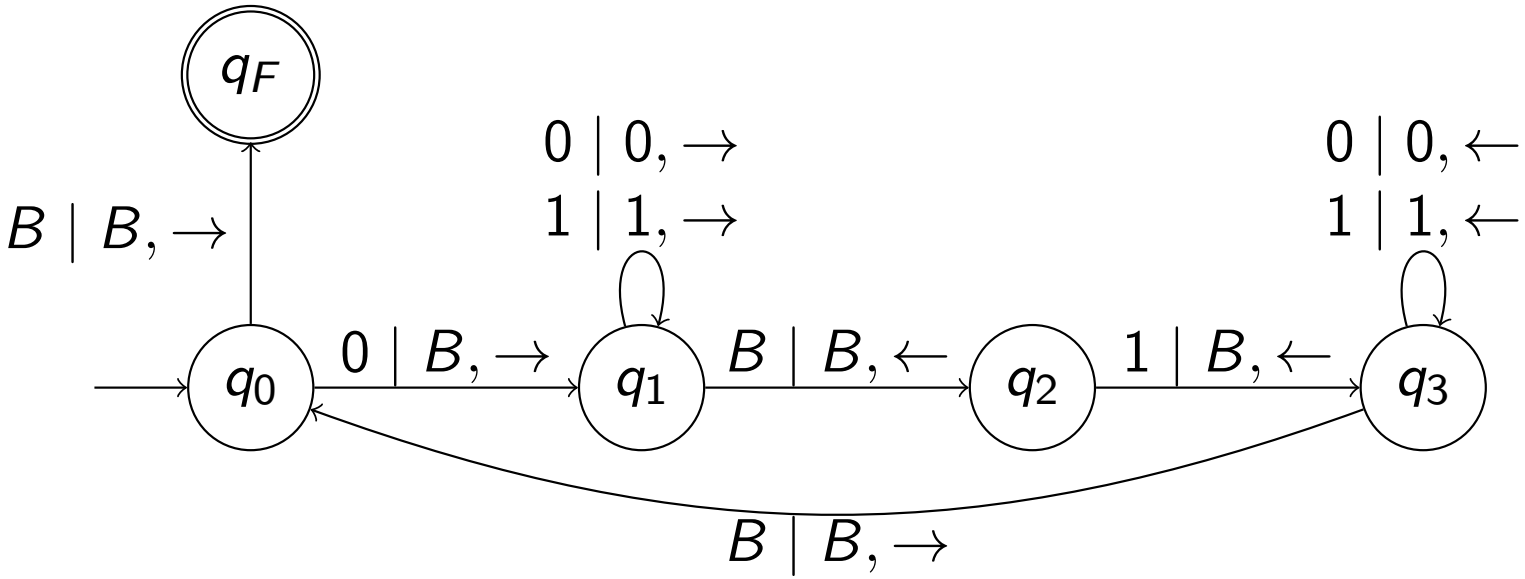
Exemple de machine de Turing

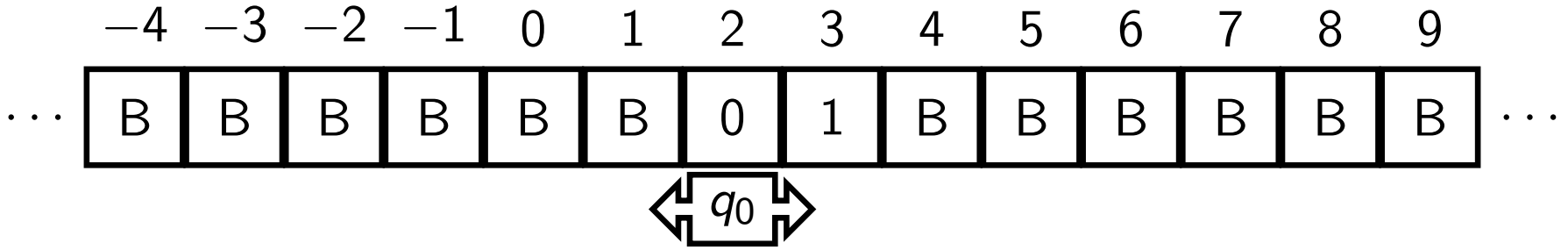




Étape:22

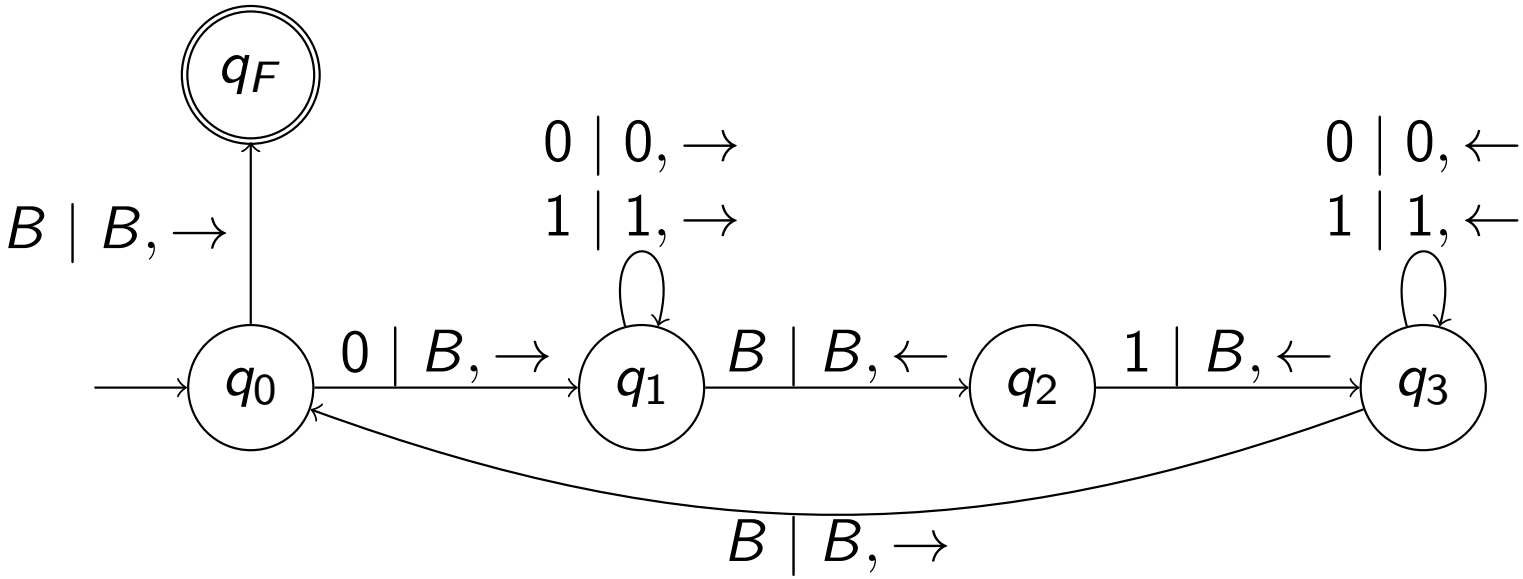
Exemple de machine de Turing

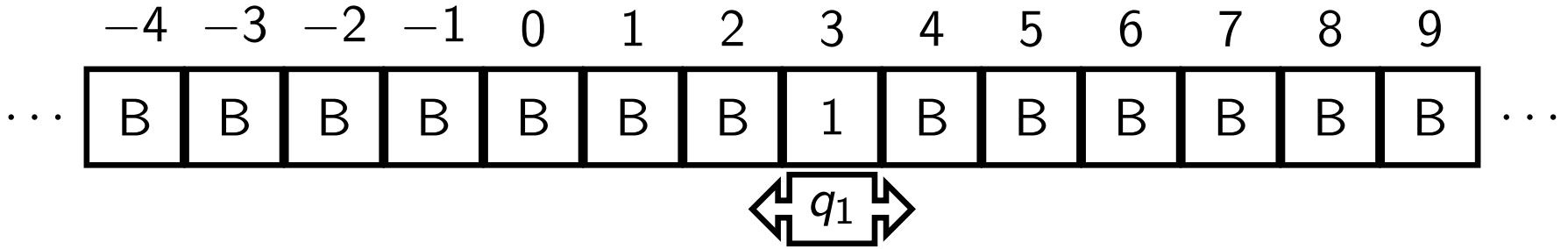




Étape:23

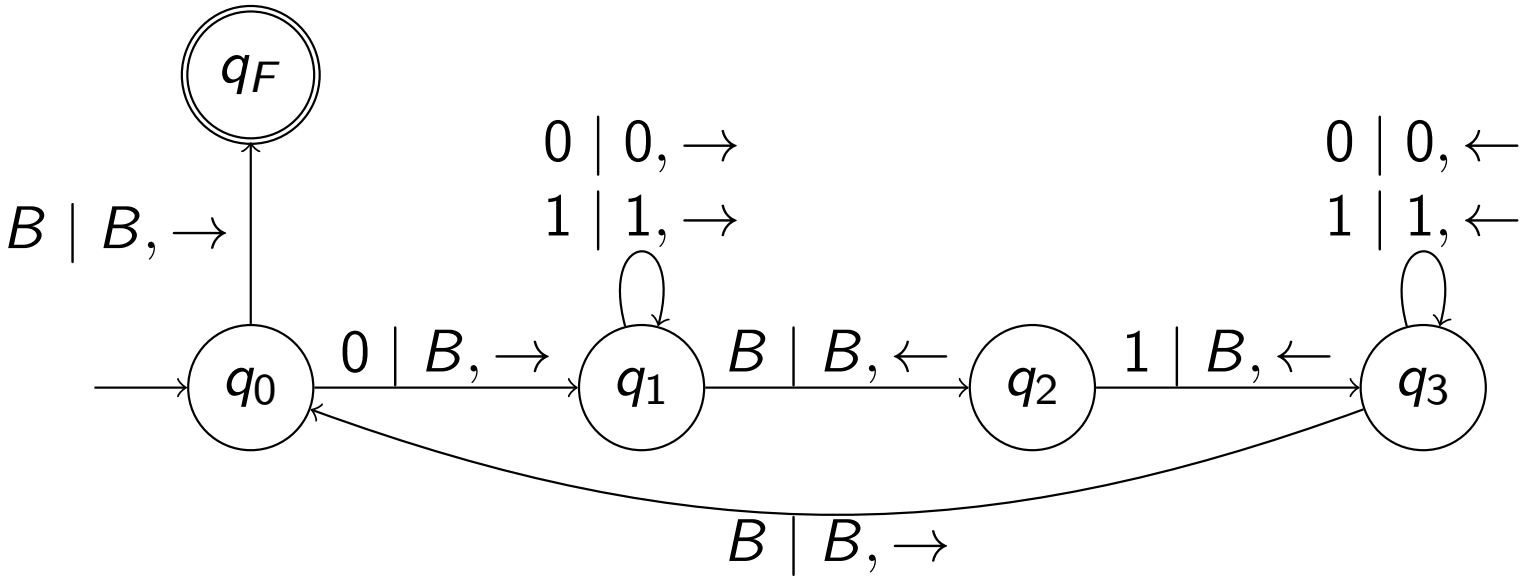
Exemple de machine de Turing

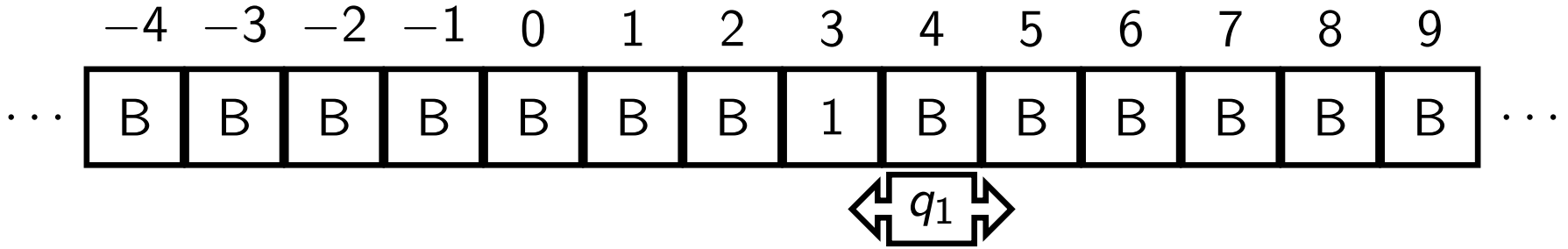




Étape:24

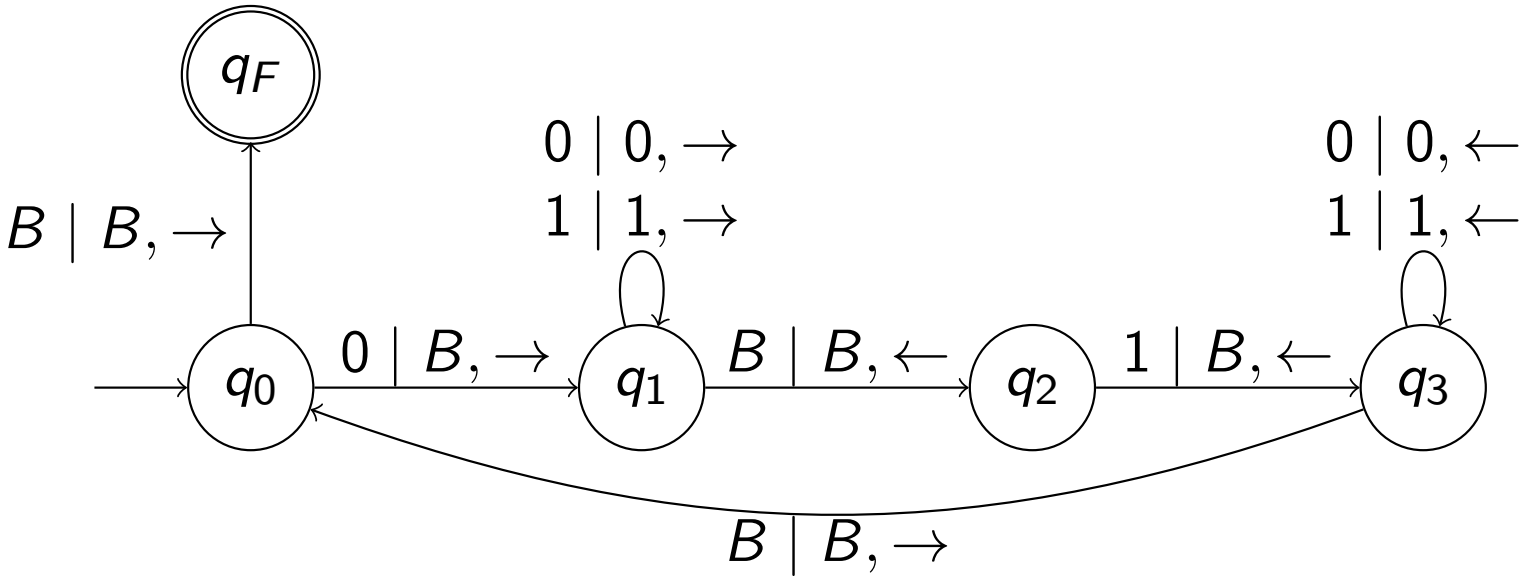
Exemple de machine de Turing

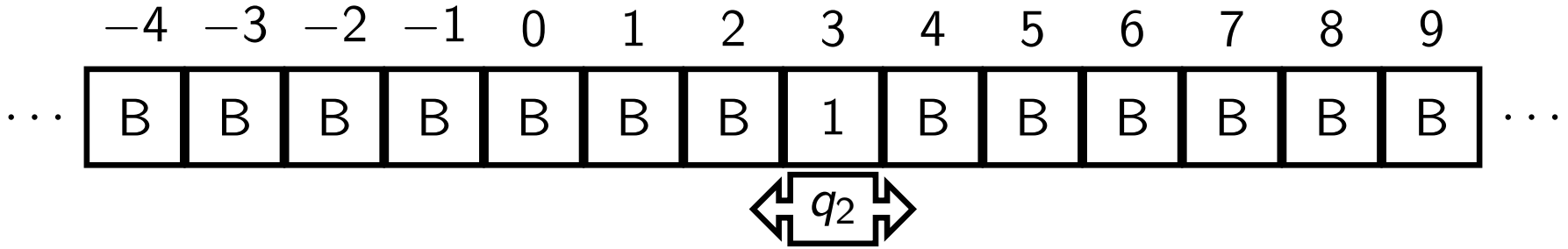




Étape:25

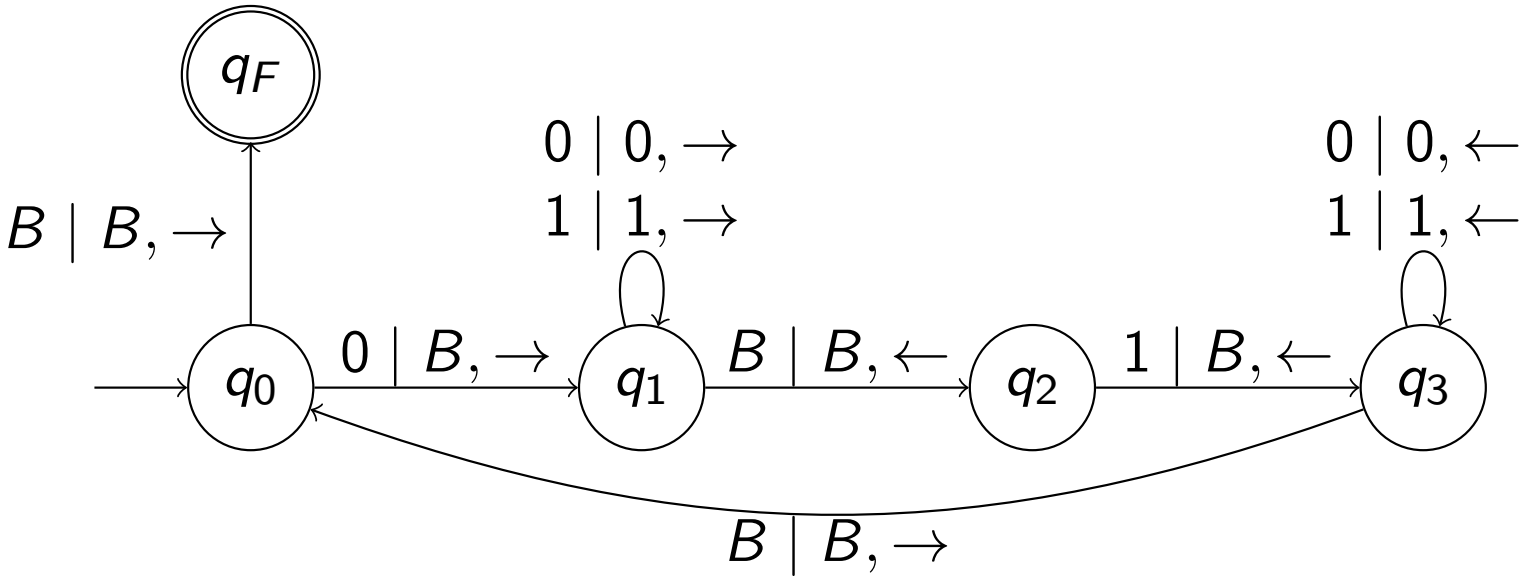
Exemple de machine de Turing

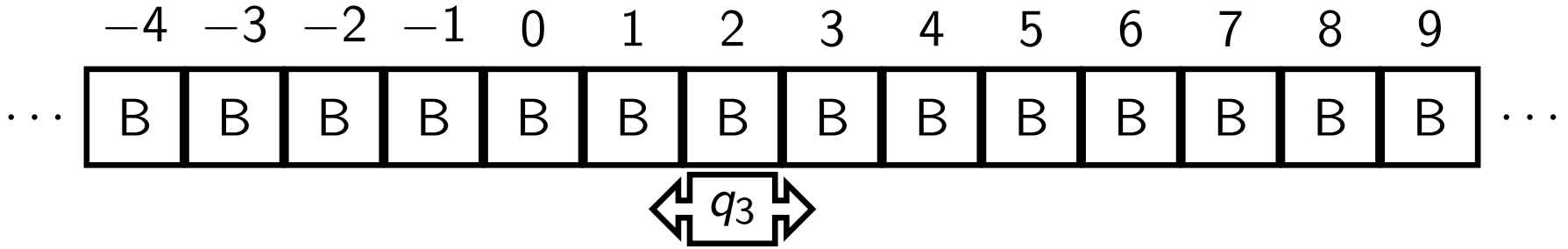




Étape:26

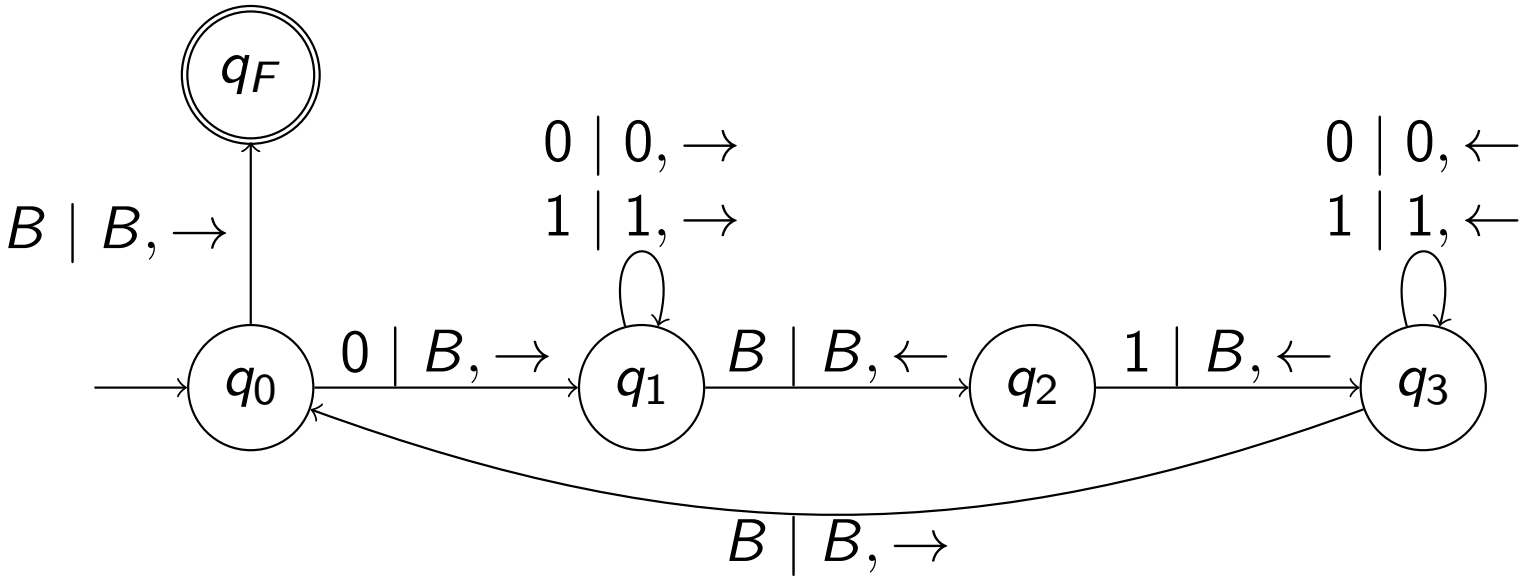
Exemple de machine de Turing

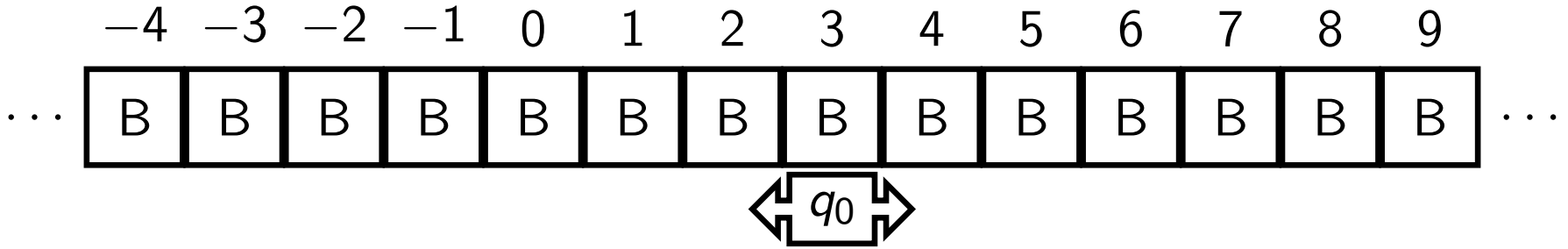




Étape:27

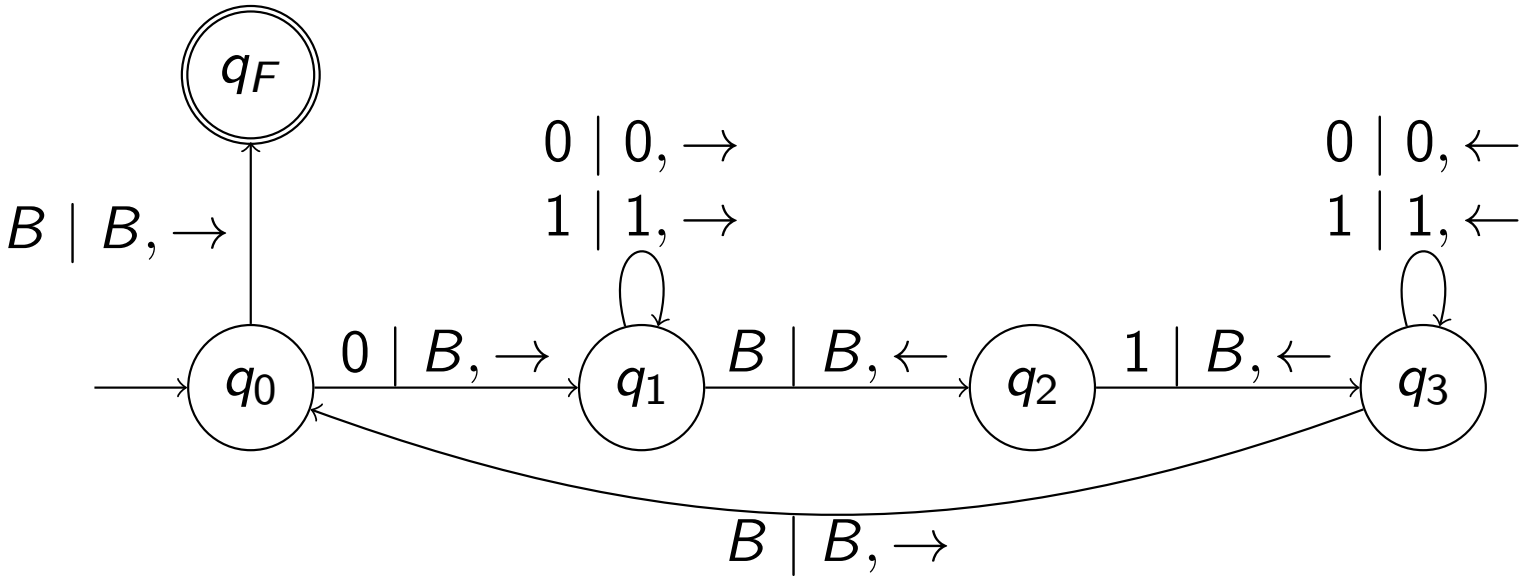
Exemple de machine de Turing

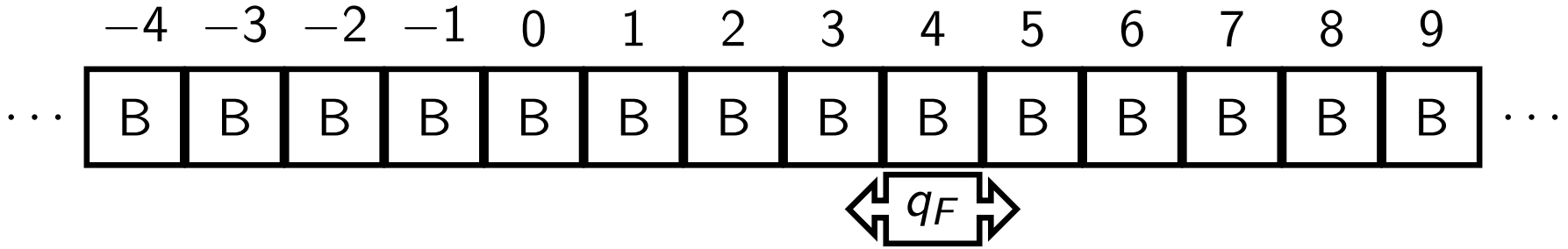




Étape:28

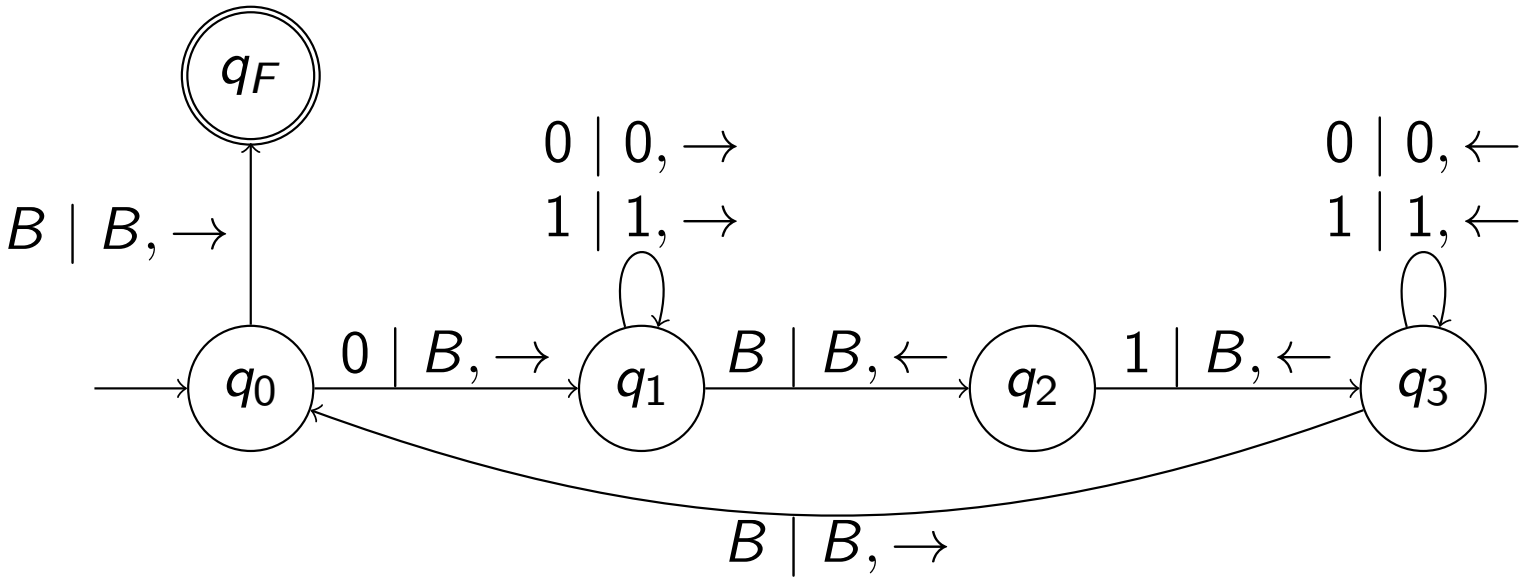
Exemple de machine de Turing

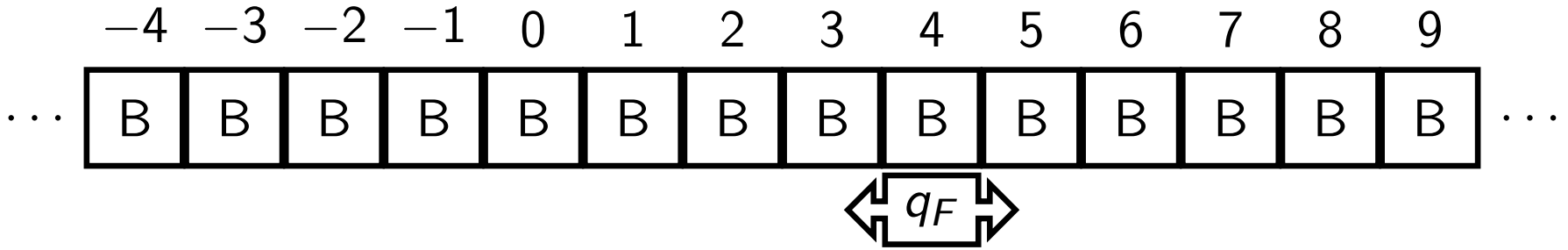




Étape:29

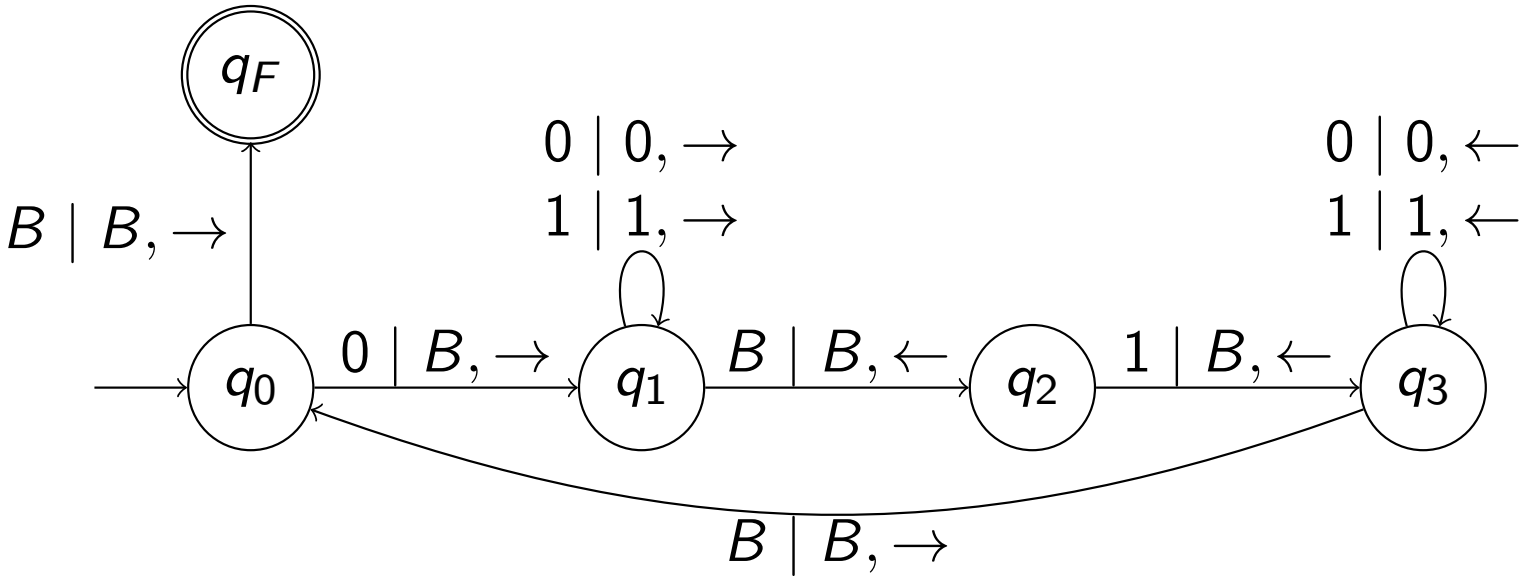
Exemple de machine de Turing

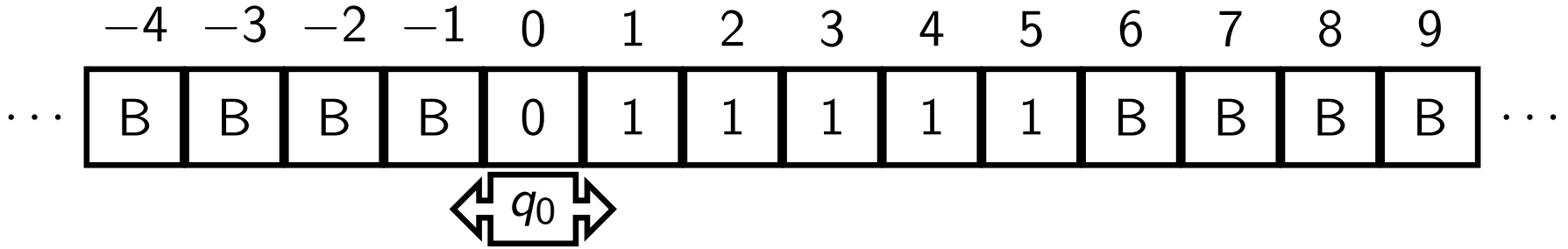




La machine M accepte le mot 000111: elle renvoie vrai.
Considérons la fonction f définie à partir de M, $f(000111) = \top$.

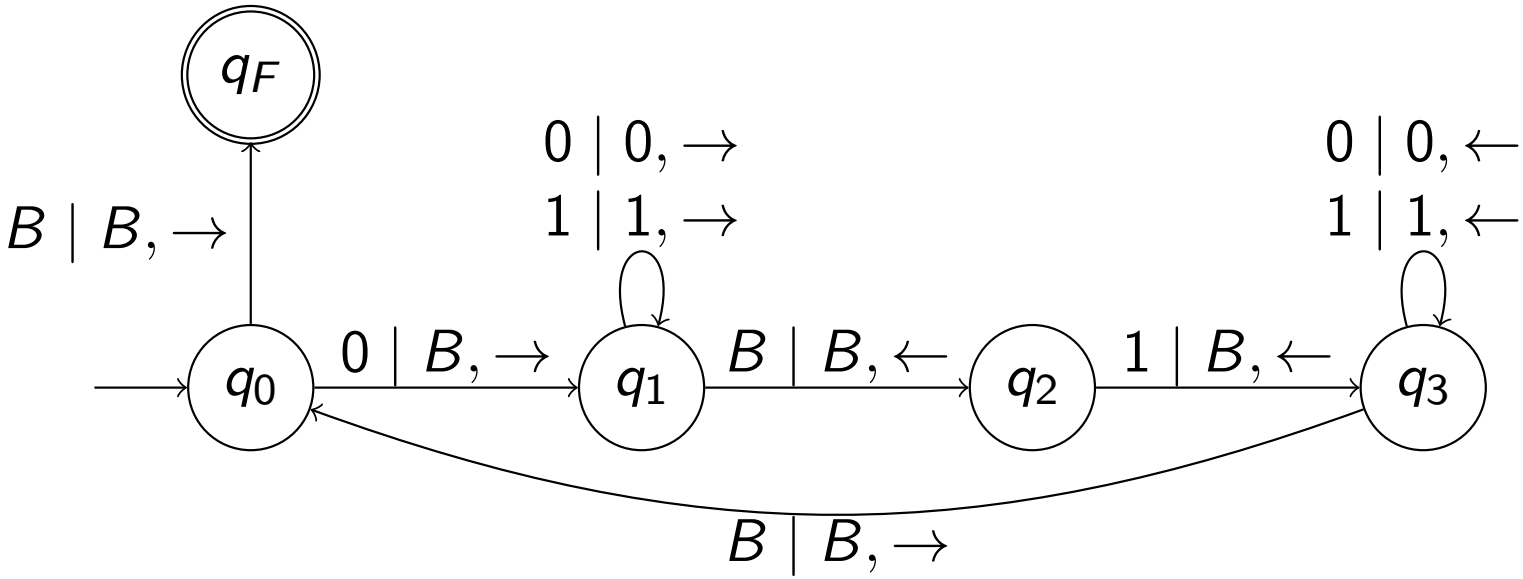
Exemple de machine de Turing

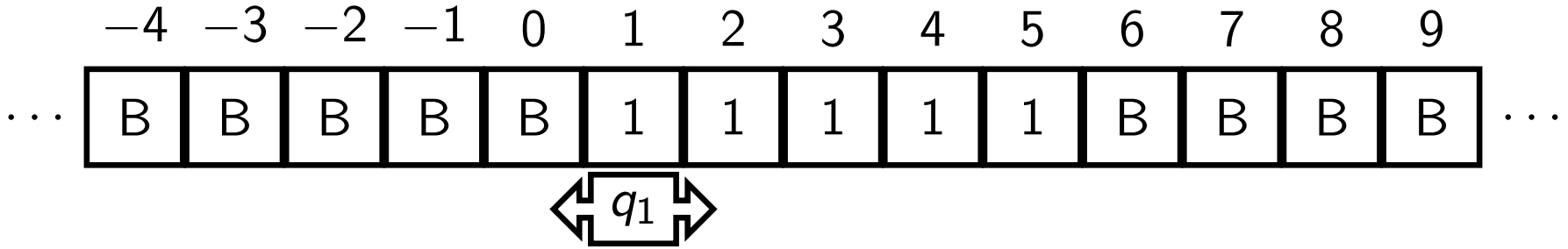




Étape:1

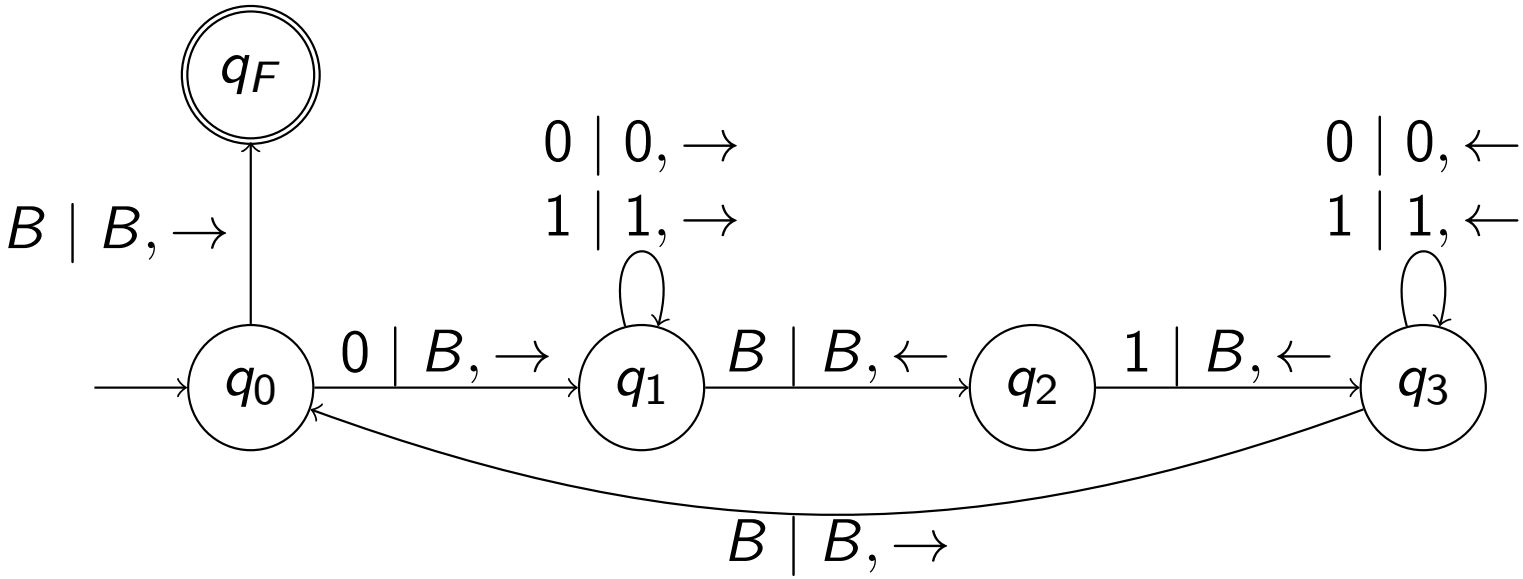
Exemple de machine de Turing

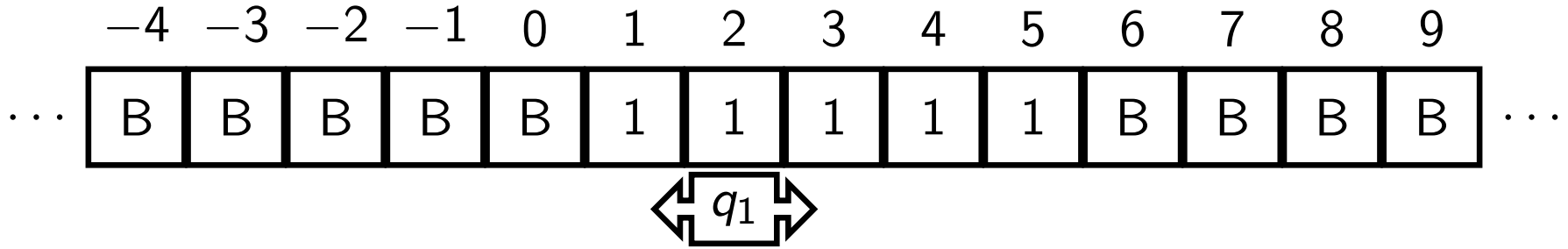




Étape:2

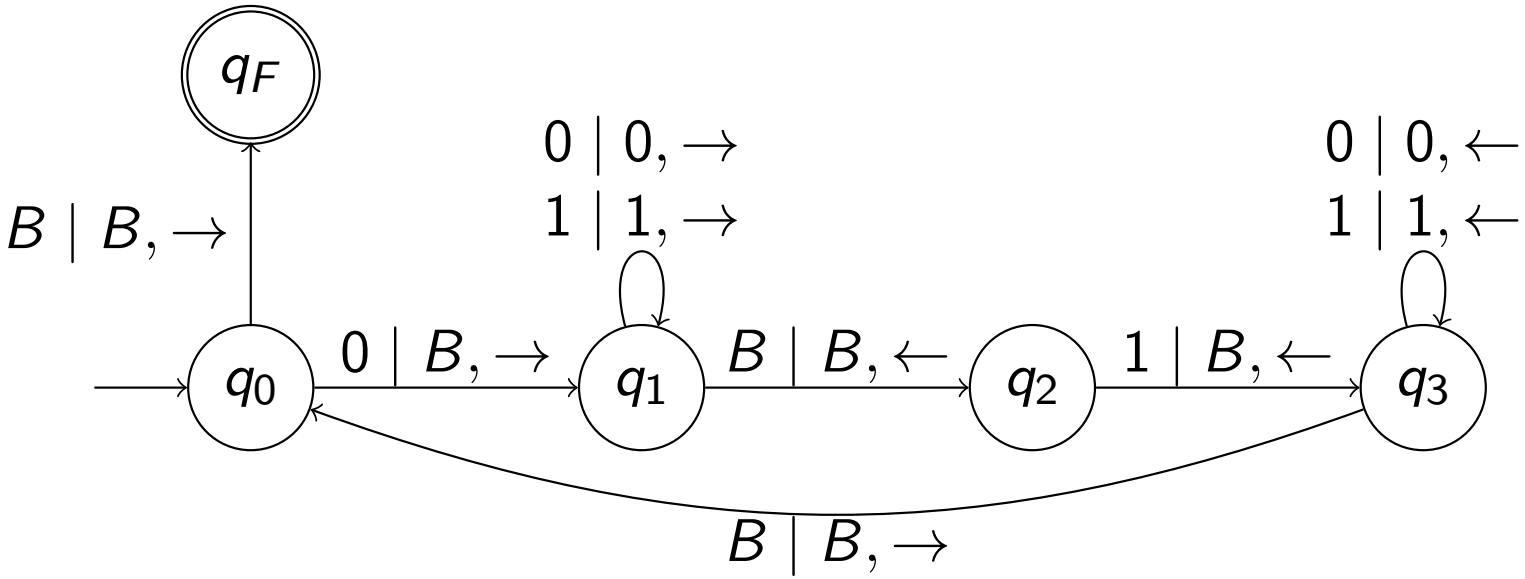
Exemple de machine de Turing

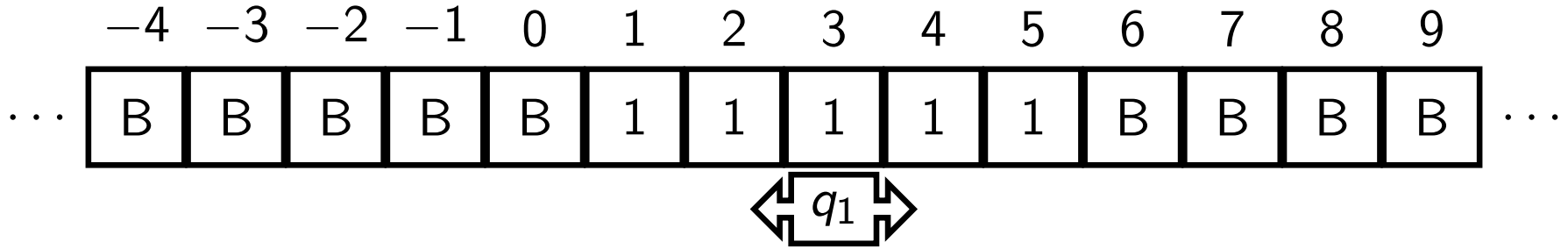




Étape:3

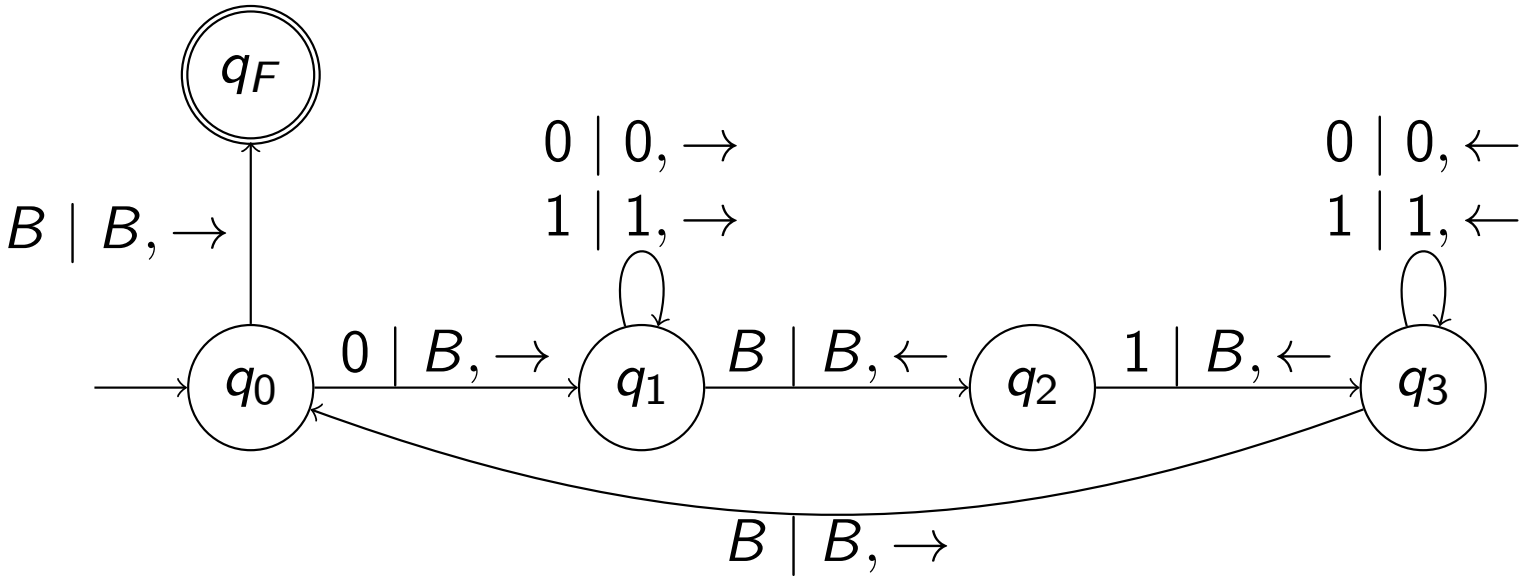
Exemple de machine de Turing

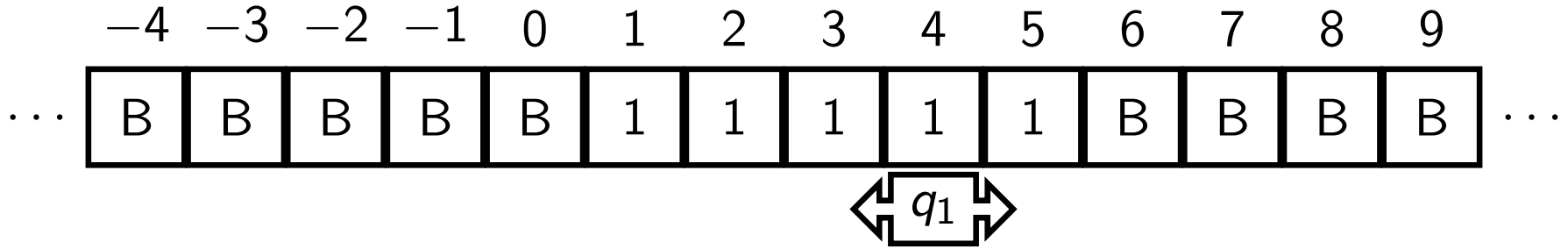




Étape:4

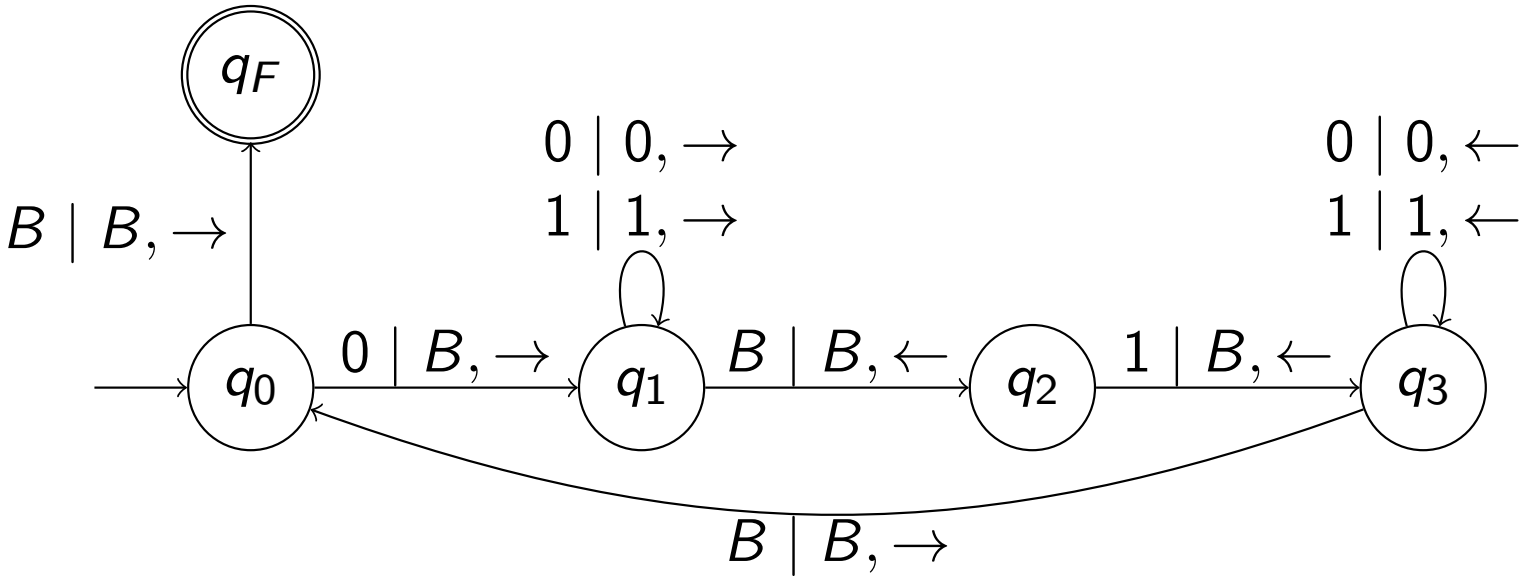
Exemple de machine de Turing

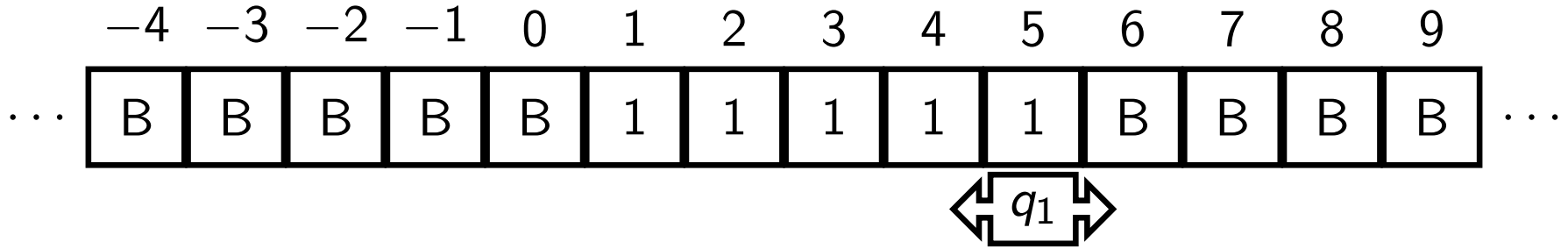




Étape:5

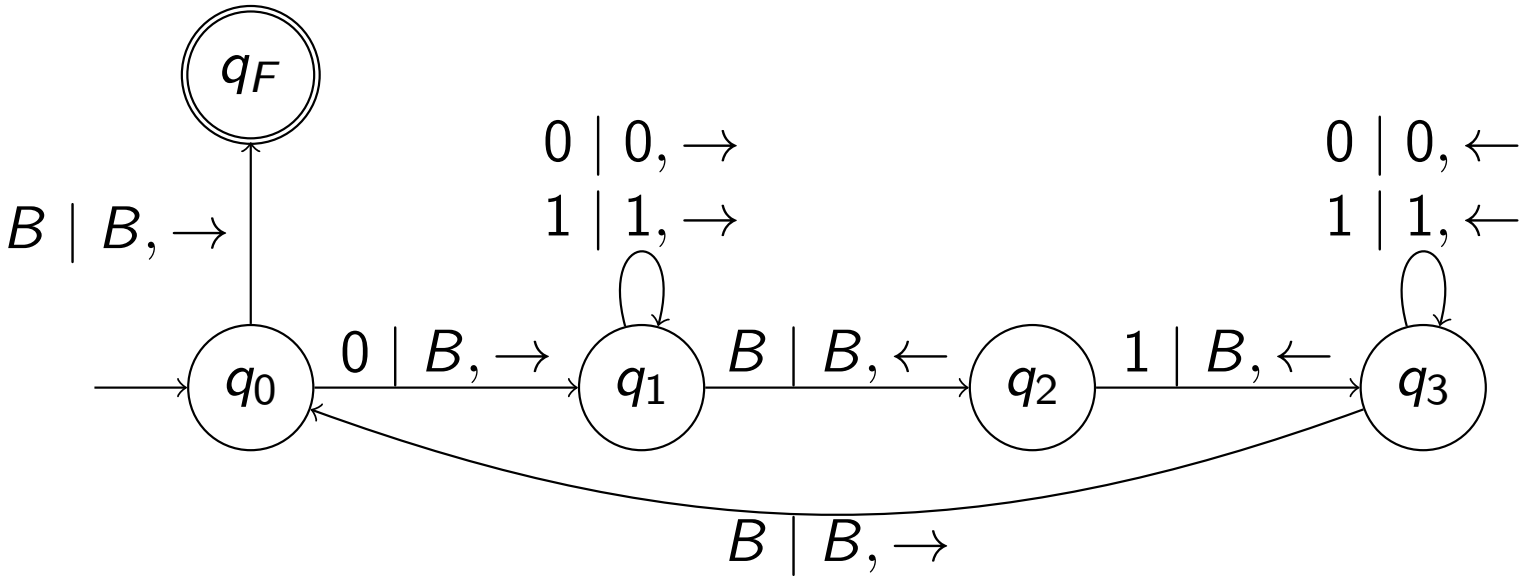
Exemple de machine de Turing

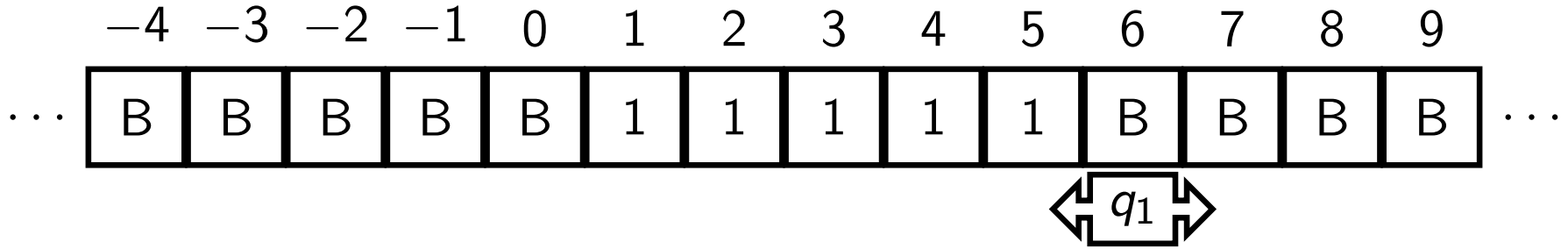




Étape:6

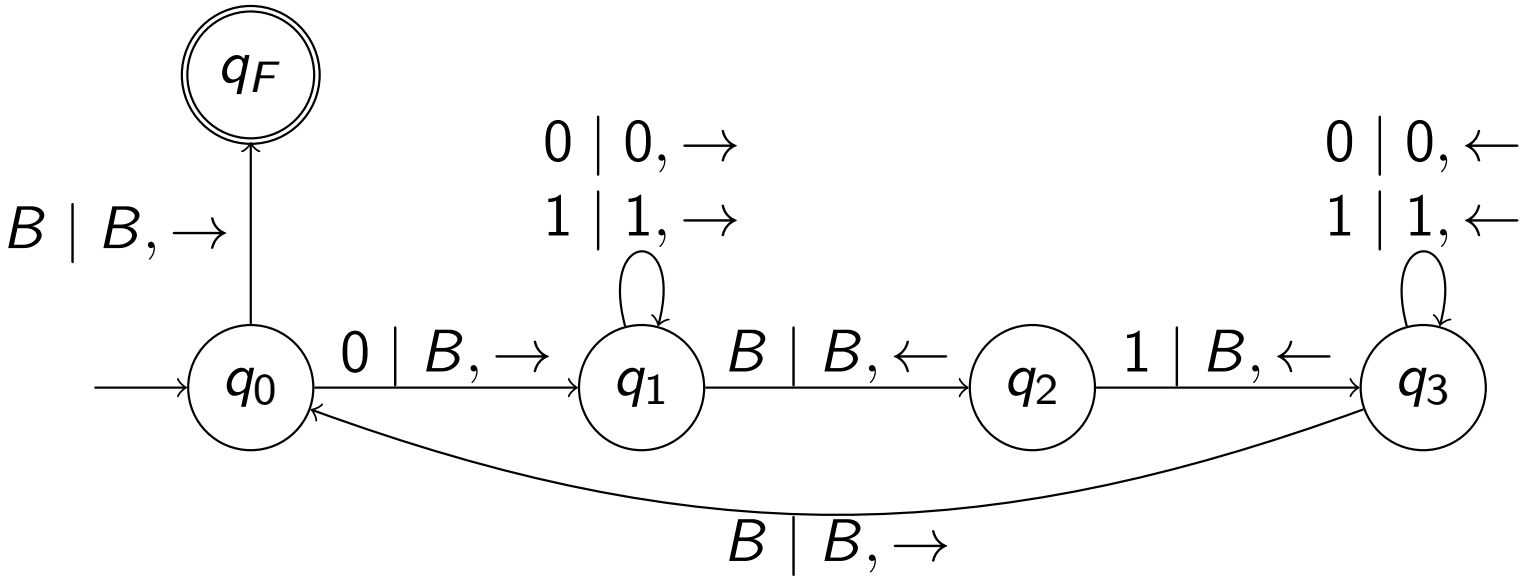
Exemple de machine de Turing

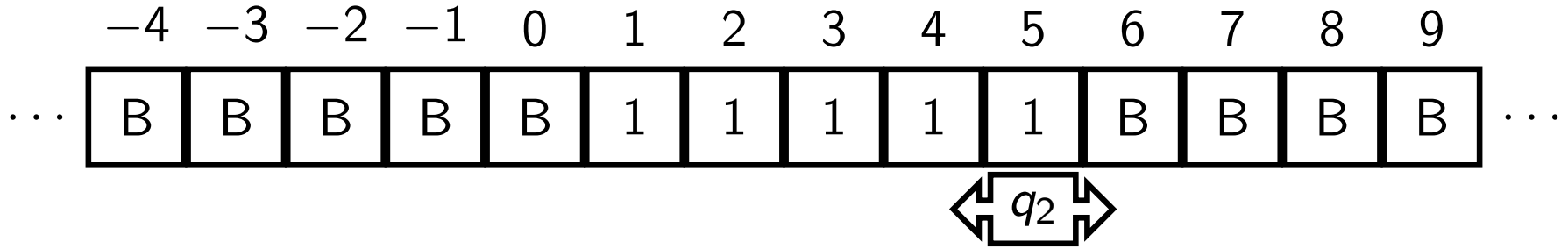




Étape:7

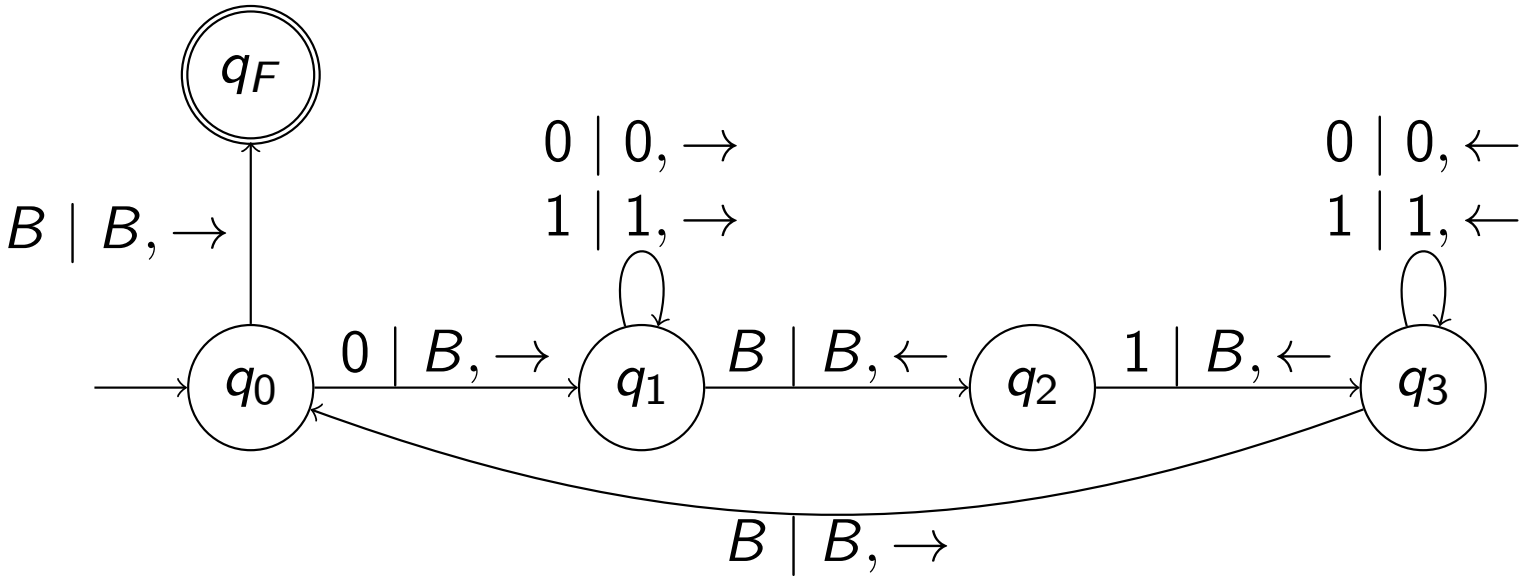
Exemple de machine de Turing

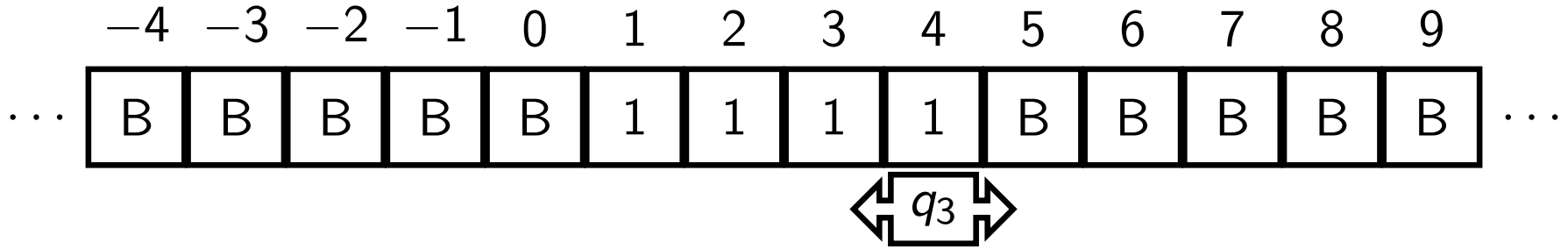




Étape:8

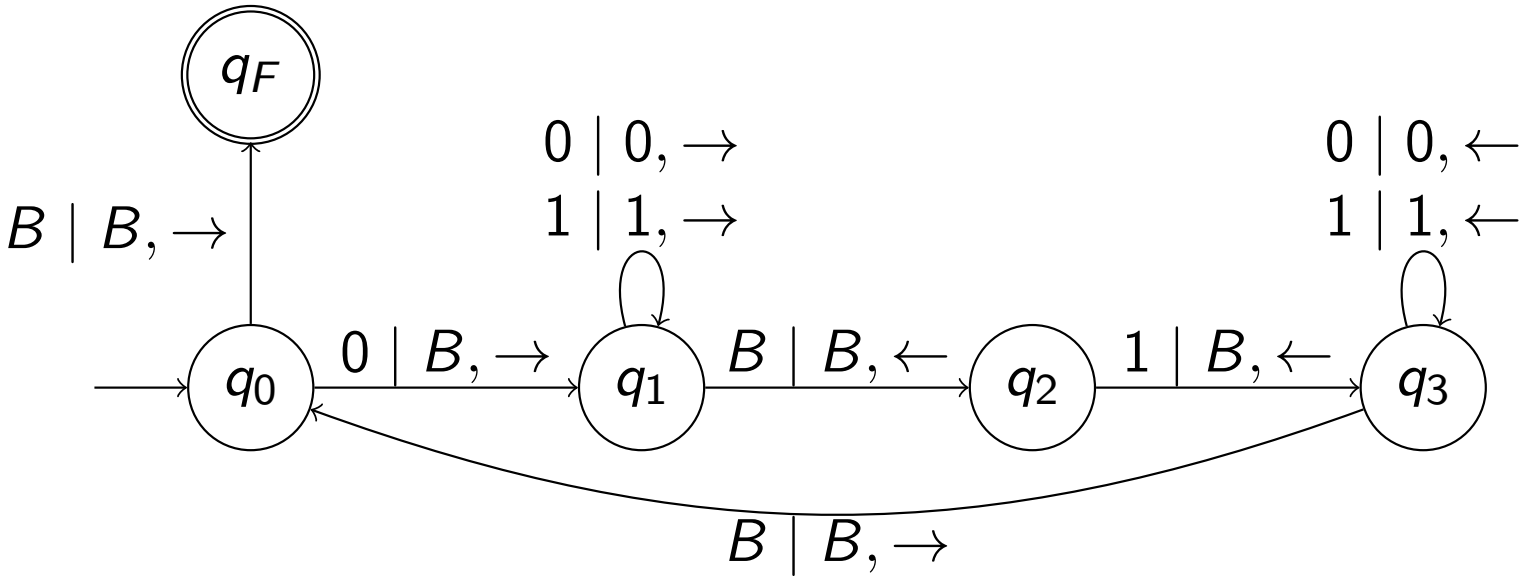
Exemple de machine de Turing

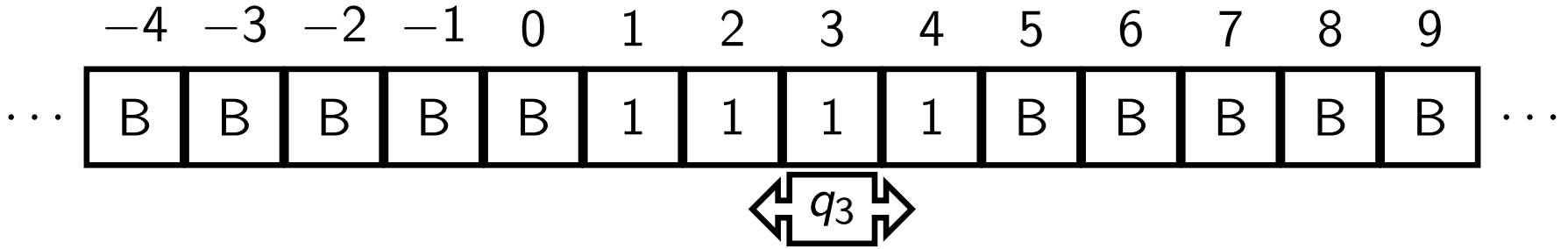




Étape:9

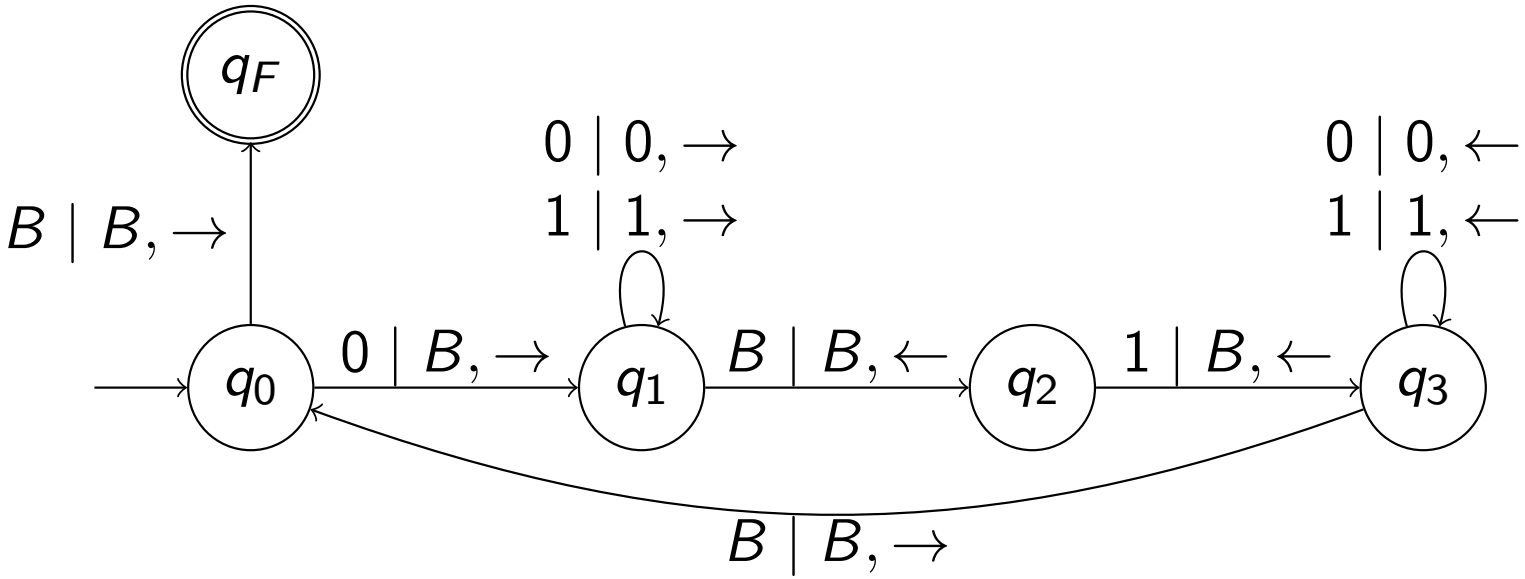
Exemple de machine de Turing

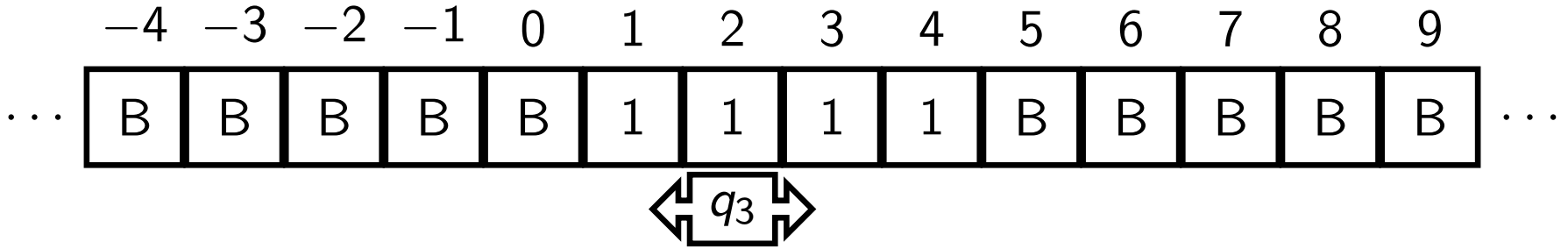




Étape:10

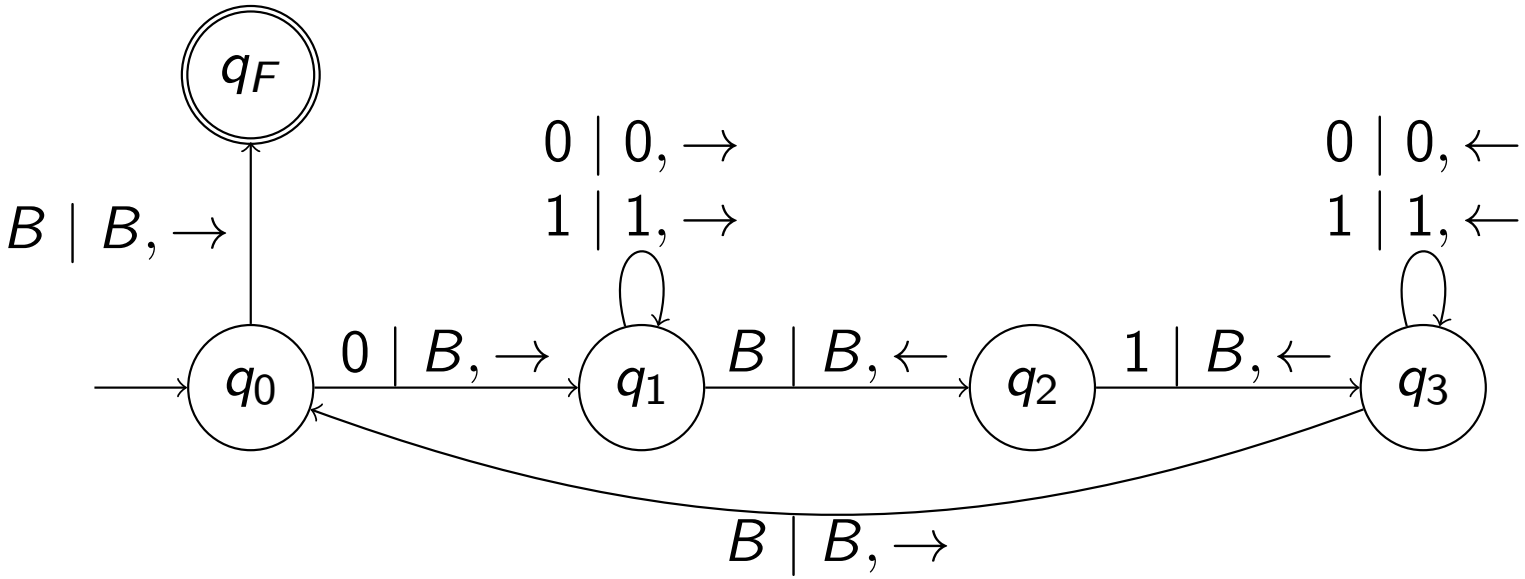
Exemple de machine de Turing

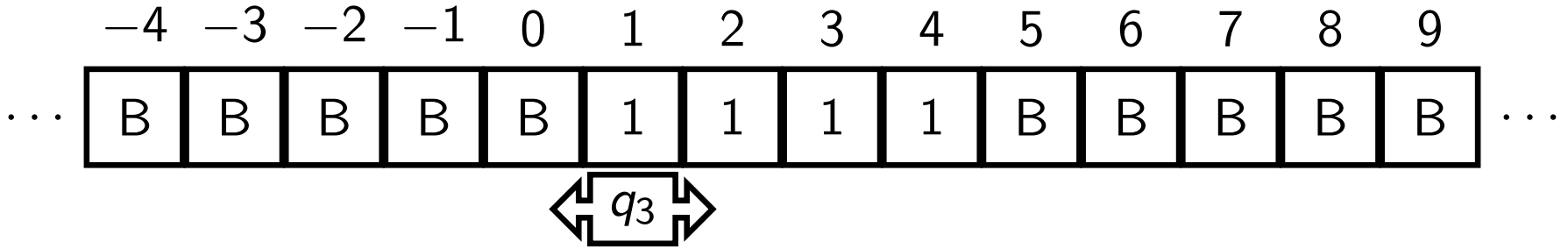




Étape:11

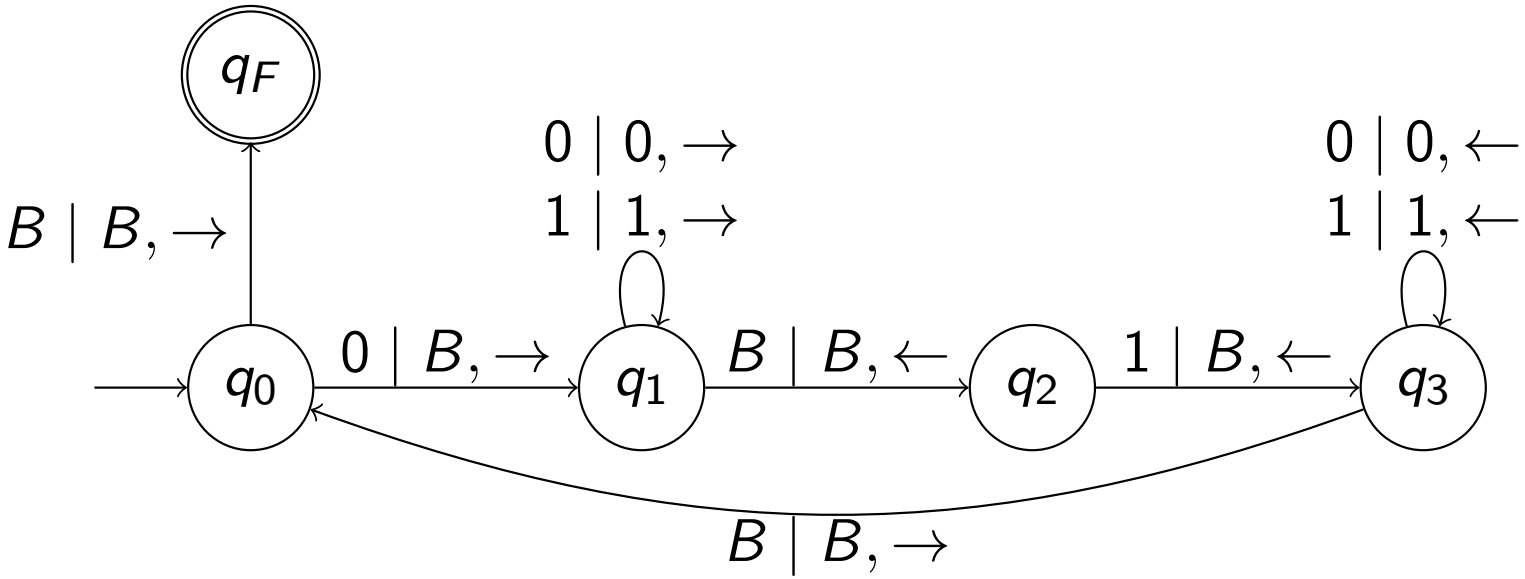
Exemple de machine de Turing

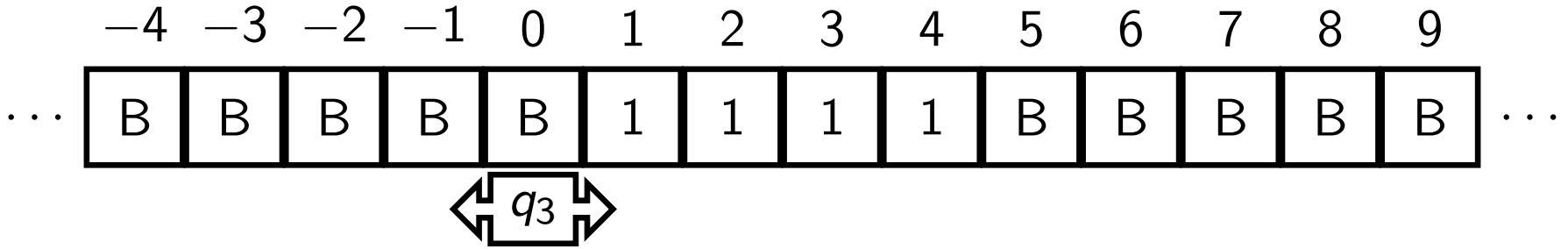




Étape:12

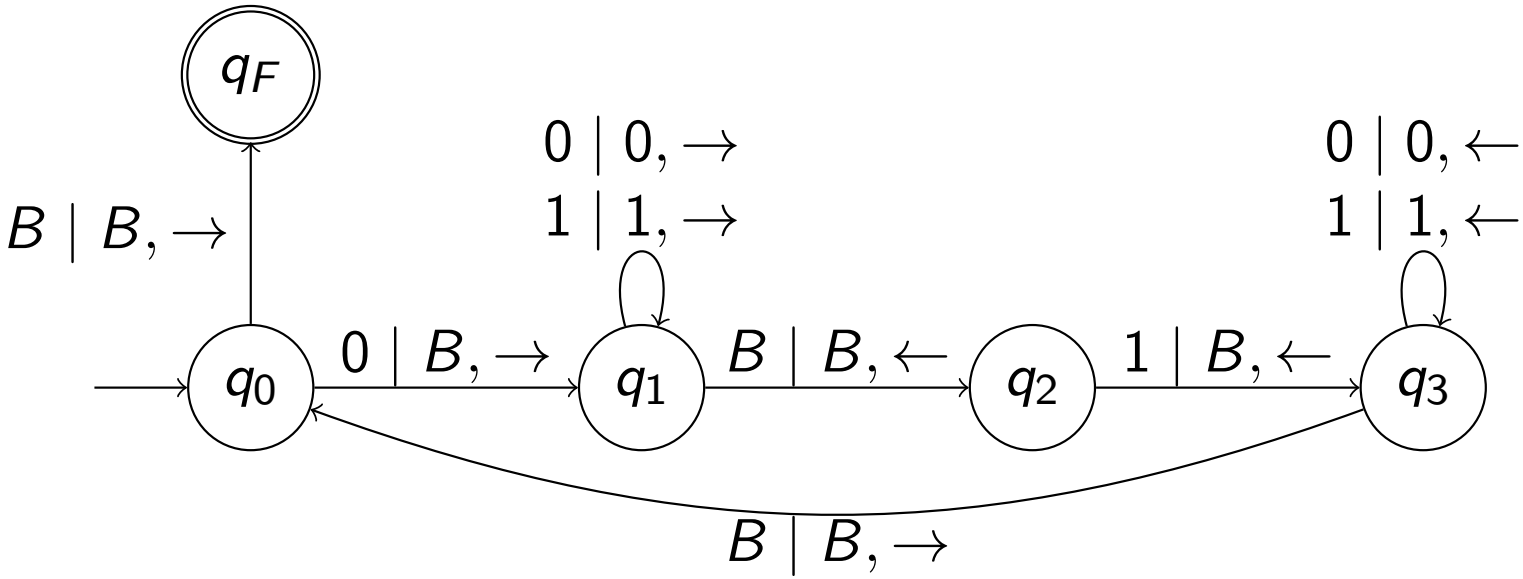
Exemple de machine de Turing

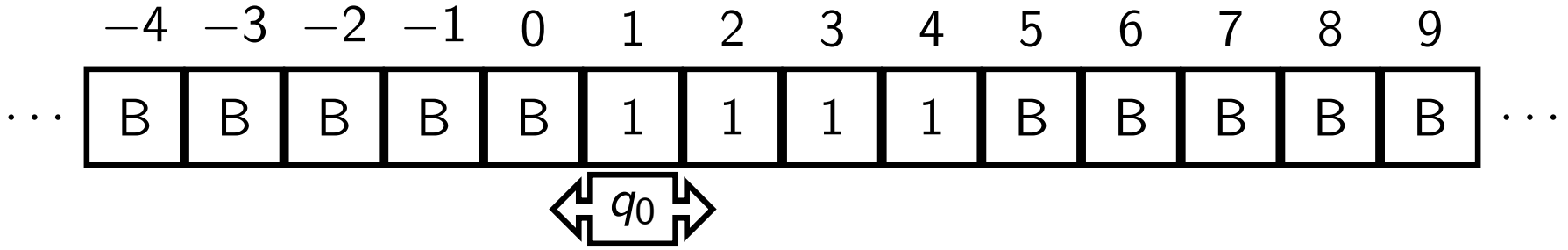




Étape:13

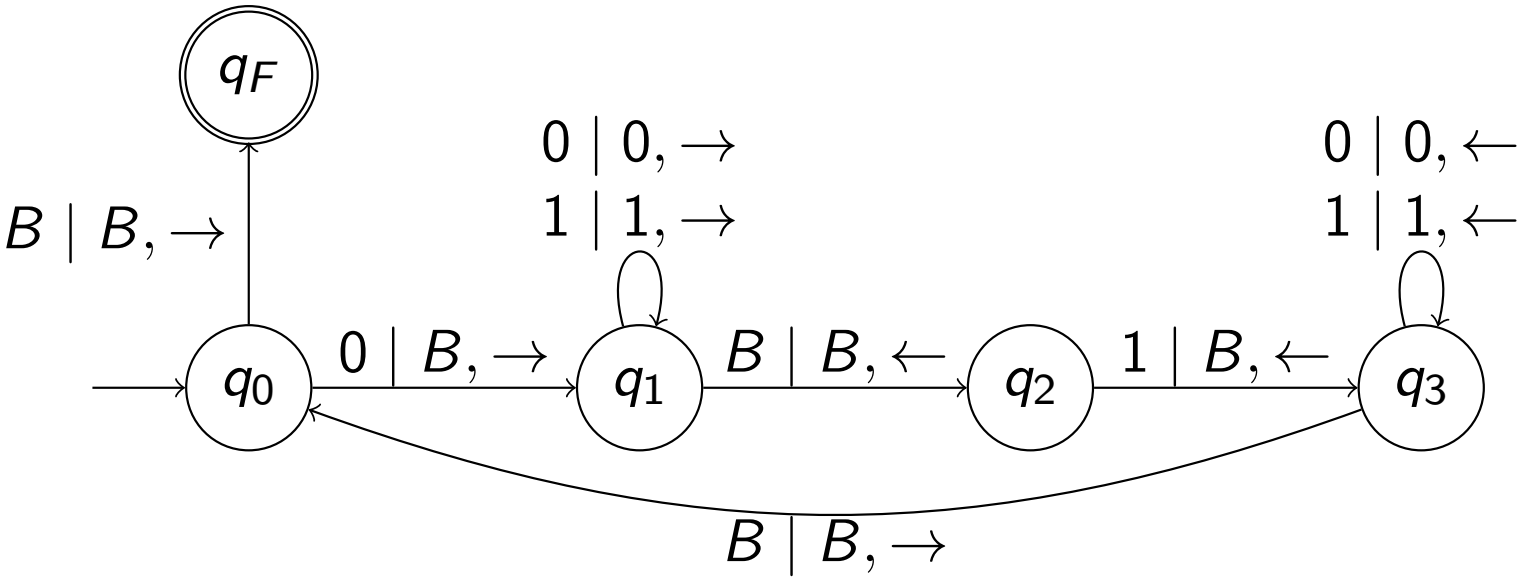
Exemple de machine de Turing

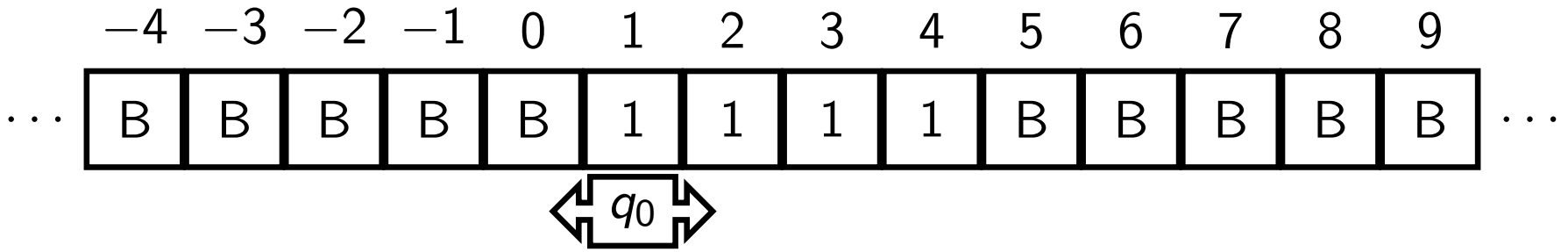




Étape:14

Exemple de machine de Turing





La machine M n'accepte pas le mot 011111: elle renvoie faux.
Considérons la fonction f définie à partir de M, $f(011111) = \perp$.

Machine de Turing: définition formelle

Definition

Une **machine de Turing** (**MT**) déterministe est un 7-uplet $M = (Q, \Gamma, \Sigma, \delta, q_0, B, q_F)$ où

- Q est un ensemble fini : les **états**,
- Γ est un ensemble fini : l'**alphabet de ruban**,
- $\Sigma \subset \Gamma$ est l'**alphabet d'entrée**,
- $\delta : (Q \setminus \{q_F\}) \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la **fonction de transition** décrite ci-après,
- $q_0 \in Q$ est l'**état initial**,
- $B \in \Gamma \setminus \Sigma$ est le **symbole blanc**,
- $q_F \in Q$ est l'**état final**.

Machine de Turing: définition formelle

Definition

- Γ contient tous les symboles qui peuvent apparaître sur le ruban et $\Sigma \subset \Gamma$ car l'entrée est initialement écrite sur le ruban.
- La fonction de transition δ est une application partielle de l'ensemble $(Q \setminus \{q_F\}) \times \Gamma$ dans l'ensemble $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$.
- Une transition $\delta(q, a) = (p, b, \leftarrow)$ signifie que dans l'état q et en lisant le symbole de ruban a, la machine passe dans l'état p, remplace a par b sur le ruban, et déplace la tête de lecture/écriture d'une cellule sur la gauche. Une application partielle peut être indéfinie pour certains arguments, auquel cas la machine n'a pas de mouvement suivant et s'arrête.
- En particulier, il n'y a pas de transition depuis l'état final q_F.

Machine de Turing: définition formelle

Definition

Initialement, le mot d'entrée est écrit sur le ruban et toutes les autres cellules contiennent le symbole blanc B. La machine est dans l'état q_0, et la tête de lecture/écriture est positionnée sur la lettre la plus à gauche de l'entrée. Il y a trois possibilités :

- **acceptation** si au cours des transitions la machine entre dans l'état final q_F (et donc s'arrête),
- **rejet** si au cours des transitions la machine s'arrête dans un état non final (s'il n'y a pas de mouvement suivant à réaliser),
- **rejet** si la machine ne s'arrête jamais.

Machine de Turing: définition formelle

Remarque:

La machine de Turing telle qu'on vient de la définir ne permet de calculer que des *problèmes de décision* (c'est-à-dire, où la sortie n'est que vrai ou faux).
Mais:

- On peut facilement définir des variations de cette machine de Turing qui permet de calculer d'autres types de fonctions (en écrivant la réponse sur le ruban par exemple).
- Pour toute fonction $f : A \to B$, on peut trouver une fonction de décision équivalente d'un point de vue calculatoire. On peut prendre par exemple, la fonction $g : A \times B \to \{\top, \bot\}$ définie comme $g(x, y) = \top$ ssi $f(x) = y$.

Table of Contents

Langages

Definition

Le langage **reconnu** (ou **accepté**) par la MT M de fonction $f_M : \Sigma^* \to \{\bot, \top\}$ est $L(M) = \{w \in \Sigma^* \mid f_M(w) = \top\}$.

Definition

- Un langage est **décidable**, s'il est reconnu par une machine de Turing qui s'arrête sur toutes les entrées.
- Un langage est **semi-décidable** s'il est reconnu par une machine de Turing.
- Un langage est **co-semi-décidable** si son complément est semi-décidable.

Langages

Theorem

Un langage est décidable ssi il est semi-décidable et co-semi-décidable.

Preuve au TD 3

Remarque

Dans la littérature,

- **décidable** se dit **récursif**,
- **semi-décidable** se dit **récursivement énumérable**,
- **co-semi-décidable** se dit **co-récursivement énumérable**.

Remarque

Le terme **semi-décidable** vient du fait que la machine de Turing qui semi-décide s'arrête pour toute entrée qui appartient au langage (on a à coup sûr la réponse si le mot appartient au langage, car la machine atteindra un état final acceptant), mais ne s'arrête pas obligatoirement sur les entrées qui n'appartiennent pas au langage (si le mot n'appartient pas au langage, la machine peut ne pas s'arrêter). Il y a donc une asymétrie entre les mots *dans* et *en dehors* du langage. En lançant une machine sur un mot dont on se demande s'il appartient au langage, on ne sait pas si la machine va s'arrêter et donner une réponse, mais on sait que si le mot est *dans* le langage alors la machine finira par nous donner la réponse en l'acceptant au bout d'un temps fini (mais a priori inconnu).