

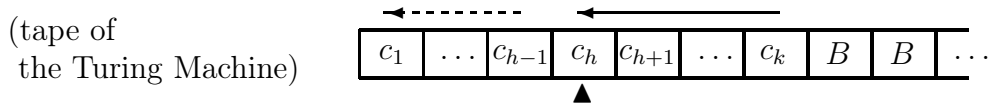
then we may encode the symbol a by the string aa and, similarly, b by ab , $\#$ by ba , and Δ by bb . Obviously, a Post Machine that uses that encoding, has to execute two dequeue statements (or two enqueue statements) for simulating a single dequeue statement (or a single enqueue statement, respectively) of the Post Machine P' .

Now we will prove the equivalence between Turing Machines and Post Machines, that is, we will prove that: (1) for any Turing Machine there exists a Post Machine which accepts the same language, and (2) vice versa.

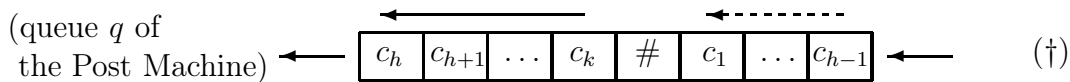
Without loss of generality, we will assume that:

- (i) the tape alphabet of the Turing Machine is $\Gamma = \{a, b, B\}$, where B is the blank symbol which is not printable,
- (ii) the Turing Machine cannot make a left move when its tape head scans the leftmost cell (recall that the position of the tape head can be tested by simulating a tape with two tracks), and
- (iii) the Post Machine has symbols a, b , and $\#$ only.

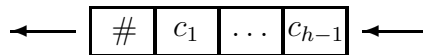
Proof of Point (1). Let us consider a tape configuration of the Turing Machine of the form:



where the rightmost cell with a non-blank symbol is c_k and the scanned cell (marked by \blacktriangle) is c_h , with $1 \leq h \leq k$. That tape configuration is represented by the following queue q of the Post Machine:



Thus, (i) the tape head of the Turing Machine scans the *leftmost cell* of the tape iff $\#$ is the rightmost element of the queue of the Post Machine (that is, the cells c_1, \dots, c_{h-1} are absent), and (ii) the tape head of the Turing Machine scans the *leftmost blank symbol* B iff $\#$ is the leftmost element of the queue of the Post Machine (that is, the cells c_h, c_{h+1}, \dots, c_k are absent). In Case (ii) if the tape of the Turing Machine has the non-blank symbols in its leftmost cells c_1, \dots, c_{h-1} , then the queue of the Post Machine is of the form:



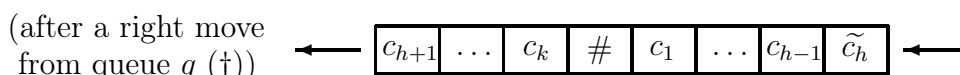
(Here we slightly depart from the representation of the tape of the Turing Machine used in [25], but no problem arises because we assumed that when the tape head of the Turing Machine scans the leftmost cell, it cannot make a left move.) In particular, if the tape of the Turing Machine has blank symbols only and the tape head scans the leftmost blank symbol B , then the queue of the Post Machine is of the form:



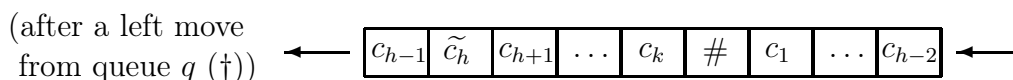
Recall that, since the blank symbol B is not printable, the tape head of the Turing Machine scans the leftmost cell with the blank symbol B , only when the tape has blank symbols only.

We have that, before and after the simulation of a move of the Turing Machine, the queue q of the Post Machine is not empty and it has at least the element $\#$.

Starting from the queue q on page 45 (see queue (\dagger)), the move of the Turing Machine that writes \tilde{c}_h in the place of c_h (here and in what follows, when no confusion arises, we identify a cell with the symbol written in that cell) and goes to the right, transforms q into the following queue:



Analogously, the move of the Turing Machine which writes \tilde{c}_h in the place of c_h and goes to the left, transforms the queue q on page 45 (see queue (\dagger)) into the following queue:



Now we show how the Post Machine may simulate via fragments of flowcharts both the right move and the left move of the Turing Machine. Thus, by suitably composing those fragments, according to the definition of the transition function of the Turing Machine, we get the simulation of the Turing Machine by a Post Machine.

In Figure 17 on page 47 we show some left and right moves of the Turing Machine and the corresponding queue configurations of the Post Machine that simulates the Turing Machine. This figure will help the reader to understand the statements of the simulating Post Machine. Here is the simulation of the right move and the left move.

SIMULATION OF THE RIGHT MOVE. The right move of the Turing Machine that writes \tilde{c} before moving, is simulated by the Post Machine by performing the following operations:

- (i) dequeue;
- (ii) if the dequeued element is $\#$ then insert the element $\#$ into the queue from its left end (this insertion is performed when the queue, which may or may not be empty, has no element $\#$);
- (iii) enqueue of \tilde{c} .

Thus, formally, we have that the right move is simulated by:

$$\begin{aligned} & dequeue(q)[\eta, \alpha, \alpha, \chi]; \\ \chi: & enqueue(q, \#); \quad enqueue(q, \#); \\ \omega: & dequeue(q)[\eta_1, \alpha_1, \beta_1, \alpha]; \quad \alpha_1: enqueue(q, a); \quad goto \omega; \quad \beta_1: enqueue(q, b); \quad goto \omega; \\ \alpha: & enqueue(q, \tilde{c}); \end{aligned}$$

Note that there is no need for the statements with label η or η_1 because the control never goes to those statements.

SIMULATION OF THE LEFT MOVE. The left move of the Turing Machine that writes \tilde{c} before moving, is simulated by the Post Machine by performing the following operations:

- (i) dequeue;
- (ii) if the dequeued element is $\#$ then insert the element $\#$ into the queue from its left end;
- (iii) ℓ -enqueue the element \tilde{c} ; r -dequeue one element, call it e ; ℓ -enqueue the element e (\ddagger)

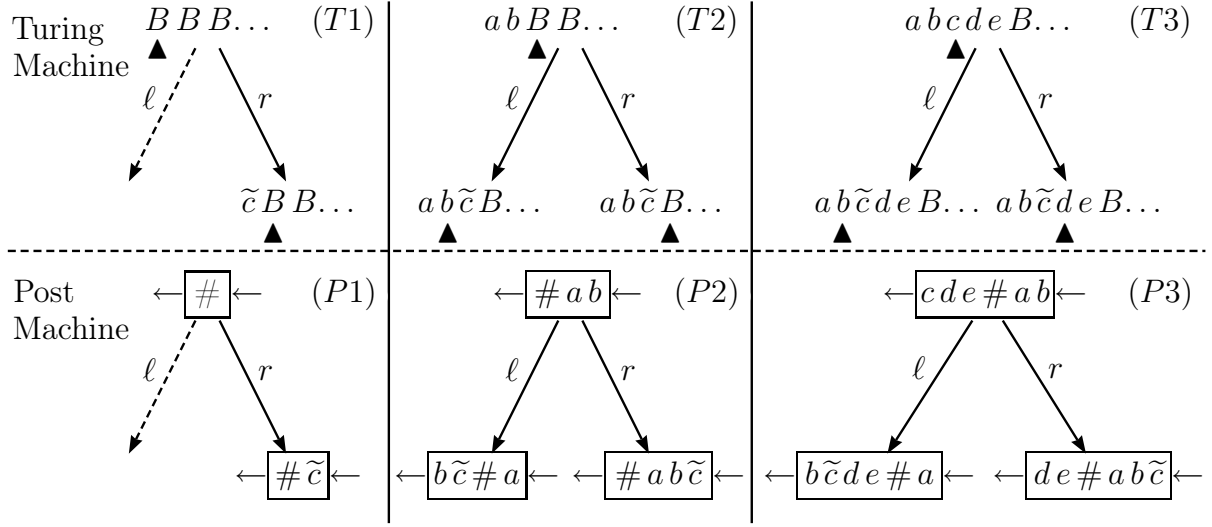


Fig. 17. Simulation of the Turing Machine by the Post Machine. Arcs with labels l and r denote the moves to the left and to the right, respectively. The \blacktriangle symbol shows the scanned cell. The symbol \tilde{c} is written in the scanned cell before moving to the left or to the right. The dashed arcs are not followed. The moves of the Turing Machine from the configurations (Ti) 's (for $i = 1, 2, 3$) are simulated by the moves of the Post Machine from the corresponding configurations (Pi) 's.

where $l\text{-enqueue}(q, e)$ denotes the insertion of the element e to the left of the queue q , and $r\text{-dequeue}(q)[\eta, \alpha, \beta, \chi]$ denotes the extraction of the rightmost element from the queue q . The labels η , α , β , and χ are the labels of the statements which are performed next if the queue q is empty, or the extracted element from the right is a , or b , or $\#$, respectively. We will define the $l\text{-enqueue}$ and the $r\text{-dequeue}$ operations below.

The operations of Point (iii) (see $(\dagger\dagger)$ above) are performed when in the queue there is the element $\#$ and the element e is either a or b . Note that e cannot be $\#$ because the left move is not allowed when the tape head of the Turing Machine scans the leftmost cell and only in that case the rightmost element of the queue is $\#$.

Thus, formally, we have that the left move is simulated by:

$$\begin{aligned}
 & dequeue(q)[\eta, \alpha, \alpha, \chi]; \\
 & \chi: enqueue(q, \#); \quad enqueue(q, \#); \\
 & \omega: dequeue(q)[\eta_1, \alpha_1, \beta_1, \alpha]; \quad \alpha_1: enqueue(q, a); \quad goto \omega; \quad \beta_1: enqueue(q, b); \quad goto \omega; \\
 & \alpha: l\text{-enqueue}(q, \tilde{c}); \\
 & \quad r\text{-dequeue}(q)[\eta_1, \alpha_1, \beta_1, \chi_1]; \\
 & \alpha_1: l\text{-enqueue}(q, a); \quad goto \varphi; \\
 & \beta_1: l\text{-enqueue}(q, b); \quad goto \varphi;
 \end{aligned}$$

where the next statement to be executed has label φ . Note that there is no need for the statements with label η or η_1 or χ_1 .

Here is the definition of the $l\text{-enqueue}(q, e)$ operation. We assume that in the queue there is the element $\#$ and the element e is either a or b . We realize the $l\text{-enqueue}(q, e)$ operation as follows: we insert a second element $\#$ in the queue, then we insert the

element e , and finally, we make a complete rotation of the elements of the queue, but we do not reinsert to the right the second element $\#$. Thus, we have that:

$$\begin{aligned} \ell\text{-enqueue}(q, e) = & \\ & \lambda: \text{enqueue}(q, \#); \text{enqueue}(q, e); \\ & \omega: \text{dequeue}(q)[\eta, \alpha, \beta, \chi]; \quad \alpha: \text{enqueue}(q, a); \text{goto } \omega; \quad \beta: \text{enqueue}(q, b); \text{goto } \omega; \\ & \chi: \text{enqueue}(q, \#); \\ & \omega_1: \text{dequeue}(q)[\eta_1, \alpha_1, \beta_1, \chi_1]; \quad \alpha_1: \text{enqueue}(q, a); \text{goto } \omega_1; \quad \beta_1: \text{enqueue}(q, b); \text{goto } \omega_1; \end{aligned}$$

where χ_1 is the label of the statement to be executed next. Note that there is no need for the statements with label η or η_1 . Note also that at label χ_1 the second element $\#$ that was inserted in the queue q by the statement at label λ , is deleted, and the element e occurs as the first (leftmost) element of the queue q , as desired.

Here is the definition of the $r\text{-dequeue}(q)[\eta, \alpha, \beta, \chi]$ operation. We assume that in the queue there is the element $\#$ which is not in the rightmost position (recall that the left move cannot take place when the tape head scans the leftmost cell), and the dequeued element is a or b . We realize the $r\text{-dequeue}(q)[\eta, \alpha, \beta, \chi]$ operation as follows. We insert a second element $\#$ in the queue and then we make a complete rotation of the elements of the queue, reinserting to the right neither the second element $\#$ nor the element which was in the queue *to the left* of that second element $\#$. This requires a delay of the reinsertion for checking whether or not the second element $\#$ follows the present element. Thus, we have that (see also Figure 18):

$$\begin{aligned} r\text{-dequeue}(q)[\eta, \alpha, \beta, \chi] = & \\ & \lambda: \text{enqueue}(q, \#); \\ & \omega_0: \text{dequeue}(q)[\eta_0, \alpha_0, \beta_0, \chi_0]; \\ & \alpha_0: \text{enqueue}(q, a); \text{goto } \omega_0; \quad \beta_0: \text{enqueue}(q, b); \text{goto } \omega_0; \quad \chi_0: \text{enqueue}(q, \#); \\ & \quad \text{dequeue}(q)[\eta_1, \alpha_1, \beta_1, \chi]; \\ & \alpha_1: \text{dequeue}(q)[\alpha_\eta, \alpha_a, \alpha_b, \alpha]; \\ & \alpha_a: \text{enqueue}(q, a); \text{goto } \alpha_1; \quad \alpha_b: \text{enqueue}(q, a); \text{goto } \beta_1; \\ & \beta_1: \text{dequeue}(q)[\beta_\eta, \beta_a, \beta_b, \beta]; \\ & \beta_a: \text{enqueue}(q, b); \text{goto } \alpha_1; \quad \beta_b: \text{enqueue}(q, b); \text{goto } \beta_1; \end{aligned}$$

Note that there is no need for statements with label η or η_0 or η_1 or α_η or β_η or χ (recall that: (i) we have assumed that the given queue q is not empty, (ii) at label λ we have inserted in the queue q a second element $\#$, and (iii) the first element $\#$ is not in the rightmost position).

Proof of Point (2). The proof that for any Post Machine there exists a Turing Machine, say M , which accepts the same language, follows from the fact that the queue of the Post Machine can be represented by the leftmost part of the tape of the Turing Machine M and the operations on the queue can be simulated by operations on the tape. We leave to the reader the details of the construction of the Turing Machine M .

Now let us introduce the following definition.

We say that a function f from N to N is *computable* by a Post Machine P iff starting with the string $a^n b a^{f(n)}$ in the queue, P eventually performs an *accept* statement.

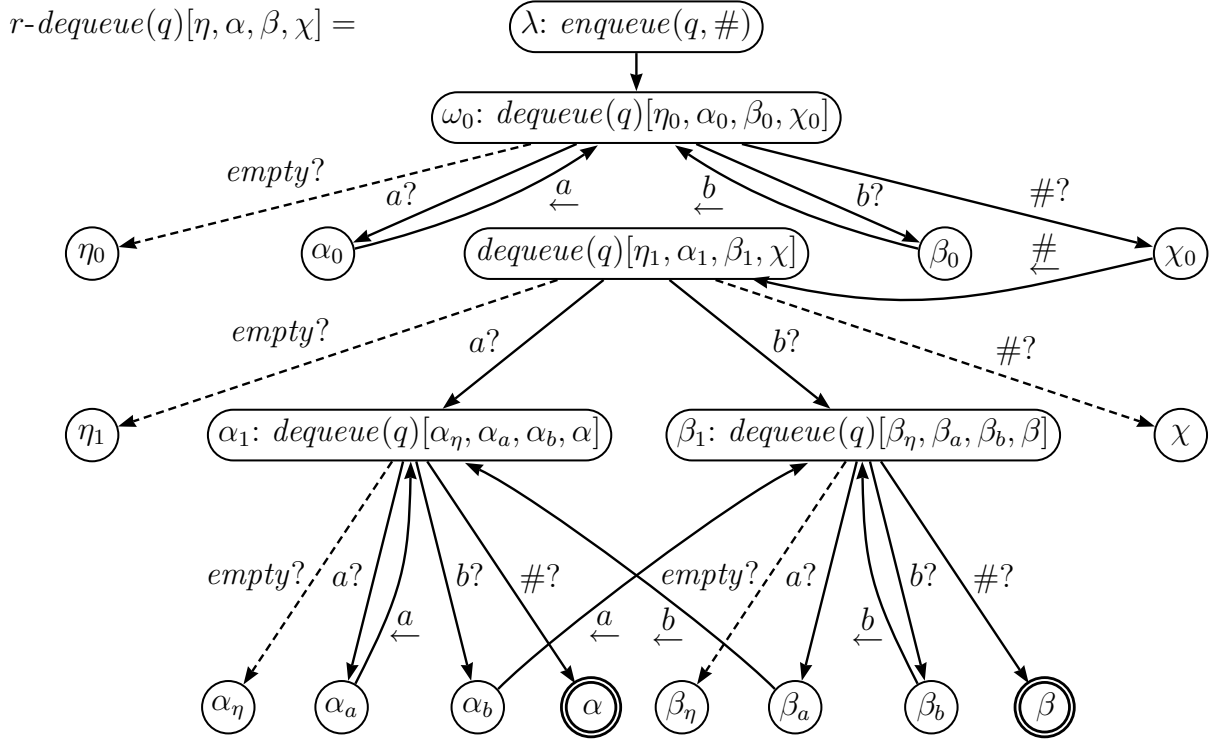


Fig. 18. Diagram that illustrates the statements for the $r\text{-dequeue}(q)[\eta, \alpha, \beta, \chi]$ operation. We have written \xleftarrow{a} , \xleftarrow{b} , and $\xleftarrow{\#}$, instead of $enqueue(q, a)$, $enqueue(q, b)$, and $enqueue(q, \#)$, respectively. The dashed arcs (those with label *empty?*) are not followed because the queue q is never empty. There is no need for statements with label η or η_0 or η_1 or α_η or β_η or χ .

Thus, as a consequence of the equivalence between Turing Machines and Post Machines which preserves the accepted language, we have that a function f from N to N is Turing computable iff f is computable by a Post Machine.