Idempotent Turing Machines

Keisuke Nakano
Research Institute of Electrical Communication, Tohoku University, Sendai, Japan

Abstract

A function f is said to be idempotent if $f(f(x)) = f(x)$ holds whenever $f(x)$ is defined. This paper presents a computation model for idempotent functions, called an idempotent Turing machine. The computation model is necessarily and sufficiently expressive in the sense that not only does it always compute an idempotent function but also every idempotent computable function can be computed by an idempotent Turing machine. Furthermore, a few typical properties of the computation model such as robustness and universality are shown. Our computation model is expected to be a basis of special-purpose (or domain-specific) programming languages in which only but all idempotent computable functions can be defined.

2012 ACM Subject Classification Theory of computation → Turing machines

Keywords and phrases Turing machines, Idempotent functions, Computable functions, Computation model

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.79

Funding This work was partially supported by JSPS KAKENHI Grant Numbers JP21K11744, JP18H04093 and JP17H06099.

Acknowledgements I am grateful to Mirai Ikebuchi for careful proofreading of earlier drafts of this manuscript. I also want to thank anonymous reviewers for their helpful comments.

1 Introduction

A function f whose domain and codomain are equal is said to be *idempotent* if $f(f(x)) = f(x)$ holds whenever $f(x)$ is defined. The idempotence of functions plays an essential role in a wide area of computer science. For example, some program optimization and parallelization do work only when core functions are idempotent; bidirectional transformation is well-behaved only when backward (putback) functions must be idempotent [4, 5, 15]. Moreover, a string sanitizer that removes or escapes potentially dangerous characters to prevent cross-site scripting attacks must be idempotent. Non-idempotent sanitizers are known to make the server vulnerable against double encoding attacks [16]. In such situations, we need to decide if a given function is idempotent or not. However, the idempotence of computable functions is undecidable in general.

To solve the problem, we may design a domain-specific language so that every function defined in the language either is always idempotent as far as it follows the syntax of the language or can be statically verified to be idempotent. Hooimeijer et al. [8] developed the BEK language for describing string sanitizers, which can be checked to perform their appropriate behaviour including idempotence. In this linguistic approach, programmers can define only idempotent functions. However, the restriction may be too strong to exclude idempotent functions that they want to define.

This paper gives a solution to this problem by a computation model, called an *idempotent Turing machine*, which exactly characterizes all idempotent computable functions. More specifically, the computation model is expressive enough for idempotent functions in the sense that every idempotent Turing machine computes an idempotent function and every idempotent computable function can be computed by an idempotent Turing machine. Because of the latter statement, we can claim that a language is sufficiently expressive for idempotent functions if it is capable of simulating any other idempotent Turing machines.

The present work follows along the lines of prior work of Axelsen and Glück [2] on *reversible Turing machines*, which are (locally) forward and backward deterministic Turing machines. They have shown that a reversible Turing machine is expressive enough for computing injective functions under *function semantics* under which the meaning of a Turing machine is specified by a function whose input and output correspond to strings on the tape at the initial and final configuration, respectively. We will adopt the function semantics to show the expressiveness of idempotent Turing machines in the present paper. In addition, two more desirable properties of idempotent Turing machines are shown under the function semantics, which have been shown for reversible Turing machines by Axelsen and Glück: the robustness under tape reduction and the existence of a universal machine. The author [14] has also followed along this line to introduce involutory Turing machines as a computation model for involution which is its own inverse.

Our contribution of the present paper is summarized as follows:

- An idempotent Turing machine is proposed as a particular form of a multitape Turing machine. Every idempotent Turing machine computes an idempotent function.
- An idempotent Turing machine is shown to be expressive enough to specify idempotent functions, i.e., every idempotent function is computed by an idempotent Turing machine.
- An idempotent Turing machine is shown to be robust under tape reduction, i.e., every multitape idempotent Turing machine can be simulated by a single-tape Turing machine.
- A universal idempotent Turing machine is shown to exist in terms of an appropriate redefinition of universality [2] i.e., there is an idempotent Turing machine which simulates any other idempotent Turing machine from the description of that machine.

After all of the above are presented, this paper concludes with the related work and a discussion on future work. Some proofs are provided in Appendix A.

2 Preliminaries

An *alphabet* is a finite set of symbols. The set of all strings over an alphabet Σ is denoted by Σ^*. For convenience, we regard a nested tuple of strings as a flattened one, e.g., $((w_1, w_2), w_3)$ and $(w_1, (w_2, w_3))$ may be identified with (w_1, w_2, w_3) for $w_1, w_2, w_3 \in \Sigma^*$.

For a (binary) relation $R \subseteq A \times B$, $a\ R\ b$ stands for $(a, b) \in R$. The identity relation $Id_A \subseteq A \times A$ is $\{(a, a) \mid a \in A\}$. The composition of two relations $R \subseteq A \times B$ and $S \subseteq B \times C$, denoted by $S \circ R$, is given as $\{(a, c) \mid \exists b \in B, a\ R\ b \wedge b\ S\ c\}$. For a relation $R \subseteq A \times B$ over two sets A and B, the *inverse relation* $R^{-1} \subseteq B \times A$ is defined by $\{(b, a) \mid a\ R\ b\}$. A relation $R \subseteq A \times A$ is said to be *symmetric* if $R^{-1} = R$. A relation $R \subseteq A \times B$ is said to be *functional* if $a\ R\ b_1$ and $a\ R\ b_2$ imply $b_1 = b_2$ for any $a \in A$ and $b_1, b_2 \in B$. A functional relation $R \subseteq A \times B$, written by $R : A \to B$, is simply called a (partial) *function* and $R(a)$ with $a \in A$ stands for $b \in B$ such that $a\ R\ b$ if exists. We may write $a \mapsto b$ for an element (a, b) in a functional relation. A function $R : A \to B$ is said to be *total* if $R(a) \in B$ is defined for any $a \in A$. A function $R : A \to B$ is said to be *injective* if R^{-1} is functional. For any injective function $R : A \to B$ it is easy to see that $R^{-1} \circ R \subseteq Id_A$ and $R \circ R^{-1} \subseteq Id_B$ hold. A function $R : A \to A$ is called *idempotent* if $R \circ R = R$ holds.

3 Turing Machines

The notion of Turing machines is one of the best-known computation model which can implement any computable functions. Many variants of Turing machines have been proposed in the literature in which a single tape or multiple tapes are used, tapes are one-ended or

doubly-infinite, and a head peeks only single cell or multiple adjacent cells. Since they are known to be equi-expressive [17], we basically follow the definition given by Axelsen and Glück [2]. In this section, we define a Turing machine and show its basic properties. We also present a reversible Turing machine, which plays an important role for our main results.

3.1 Syntax and Semantics of Turing machines

A Turing machine manipulates symbols on a doubly-infinite tape of cells according to an internal state and a fixed transition relation.

▶ **Definition 1** (k-tape Turing machine). A *k-tape Turing machine* T is a tuple $(Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ where Q is a finite set of *states*, Σ is a *tape alphabet* not containing the special blank symbol $\sqcup$, $q_{\mathsf{ini}} \in Q$ is the *initial state*, $q_{\mathsf{fin}} \in Q$ is the *final state*, and $\Delta = \Delta^{\mathsf{sym}} \uplus \Delta^{\leftrightarrow}$ is a ternary relation defining a set of *transition rules* where

$$\Delta^{\mathsf{sym}} \subseteq (Q \setminus \{q_{\mathsf{fin}}\}) \times (\Sigma_\sqcup \times \Sigma_\sqcup)^k \times (Q \setminus \{q_{\mathsf{ini}}\})$$
$$\Delta^{\leftrightarrow} \subseteq (Q \setminus \{q_{\mathsf{fin}}\}) \times \ \{\leftarrow, \blacklozenge, \rightarrow\}^k \ \times (Q \setminus \{q_{\mathsf{ini}}\})$$

in which $\Sigma_\sqcup$ stands for $\Sigma \uplus \{\sqcup\}$. A *symbol rule* in Δ^{sym} has the form $(q, (s_1 \Rightarrow s'_1, \ldots, s_k \Rightarrow s'_k), q')$ with $s_1, \ldots, s_k, s'_1, \ldots, s'_k \in \Sigma_\sqcup$. A *move rule* in $\Delta^{\leftrightarrow}$ has the form $(q, (d_1, \ldots, d_k), q')$ with $d_1, \ldots, d_k \in \{\leftarrow, \blacklozenge, \rightarrow\}$. The second component of a transition rule is called an *action*. In particular, an action in $\{(s, s) \mid s \in \Sigma_\sqcup\}^k \cup \{\blacklozenge\}^k$ is called a *null action*.

As presented in [2], transition rules are separated into symbol rules and move rules for our convenience of further discussion, in particular, about the inversion of Turing machines. Although these two kinds of actions are caused by a single rule in ordinary Turing machines [17], the separation of rules does not change the expressiveness of functions. It is easy to simulate a transition rule in an ordinary Turing machine by two transition rules and extra states in the present model.

A configuration of a k-tape Turing machine is specified by the current internal state and k tapes with their tape head. Each configuration can be characterized by $\langle l, s, r\rangle \in \Sigma_\sqcup^\omega \times \Sigma_\sqcup \times \Sigma_\sqcup^\omega$ where s is the symbol at its head position and l and r are the left and right tapes of the head. Note that $\Sigma_\sqcup^\omega$ is a set of infinite strings over $\Sigma_\sqcup$ going infinitely to the right. Accordingly l is "mirrored" where its first symbol is the immediate left one of the head.

▶ **Definition 2** (Configuration). A *configuration* of a k-tape Turing machine $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ is a tuple $(q, \langle l_1, s_1, r_1\rangle, \ldots, \langle l_k, s_k, r_k\rangle)$ where $q \in Q$ is an *internal state*, $l_i, r_i \in \Sigma_\sqcup^\omega$ for each $i = 1, \ldots, k$ are the left and right of the i-th tape head, and $s_i \in \Sigma_\sqcup$ for each $i = 1, \ldots, k$ is the symbol at the i-th tape head. The set of all configurations of T is written by $\mathcal{C}_T$.

▶ **Definition 3** (Configuration step). Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape Turing machine. Then a single *configuration step* is defined as a relation $\vdash_T$ over $\mathcal{C}_T$ such that

$$(q, \tau_1, \ldots, \tau_k) \vdash_T (q', \tau'_1, \ldots, \tau'_k)$$

holds for each transition rule $(q, a, q') \in \Delta$ where

- when $a = (s_1 \Rightarrow s'_1, \ldots, s_k \Rightarrow s'_k)$, $(\tau_i, \tau'_i) = (\langle l, s_i, r\rangle, \langle l, s'_i, r\rangle)$ holds with some $l, r \in \Sigma_\sqcup^\omega$ for all $i = 1, \ldots, k$;
- when $a = (d_1, \ldots, d_k)$ with $d_i \in \{\leftarrow, \blacklozenge, \rightarrow\}$,
  - $(\tau_i, \tau'_i) = (\langle s'l, s, r\rangle, \langle l, s', sr\rangle)$ holds with some $l, r \in \Sigma_\sqcup^\omega$ and $s, s' \in \Sigma_\sqcup$ if $d_i = \leftarrow$
  - $(\tau_i, \tau'_i) = (\langle l, s, r\rangle, \langle l, s, r\rangle)$ holds with some $l, r \in \Sigma_\sqcup^\omega$ and $s, s' \in \Sigma_\sqcup$ if $d_i = \blacklozenge$
  - $(\tau_i, \tau'_i) = (\langle l, s, s'r\rangle, \langle sl, s', r\rangle)$ holds with some $l, r \in \Sigma_\sqcup^\omega$ and $s, s' \in \Sigma_\sqcup$ if $d_i = \rightarrow$

  for all $i = 1, \ldots, k$.

The subscript T may be omitted if clear from the context.

The semantics of a k-tape Turing machine T is given by a relation over k strings based on $\vdash_T^*$ as follows. In the rest of the paper, a finite string $w \in \Sigma^*$ is used to represent an infinite string $w\sqcup^\omega \in \Sigma_\sqcup^\omega$; thereby, ε denotes $\sqcup^\omega$.

▶ **Definition 4** (Semantics of Turing machines)**.** Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape Turing machine. The *semantics* of T, denoted by $[\![T]\!]$, is given by the relation

$$[\![T]\!] = \{((w_1, \ldots, w_k), (w'_1, \ldots, w'_k)) \in (\Sigma^*)^k \times (\Sigma^*)^k \mid (q_{\mathsf{ini}}, \langle \varepsilon, \sqcup, w_1\rangle, \ldots, \langle \varepsilon, \sqcup, w_k\rangle) \vdash_T^* (q_{\mathsf{fin}}, \langle \varepsilon, \sqcup, w'_1\rangle, \ldots, \langle \varepsilon, \sqcup, w'_k\rangle)\}.$$

Recall that we may write $[\![T]\!](w_1, \ldots, w_k) = (w'_1, \ldots, w'_k)$ if $[\![T]\!]$ is functional.

Following [14], we define the notion of *tidiness* of Turing machines, which is required for further discussion in particular, on the concatenation of Turing machines defined later. Roughly speaking, the tidiness of a Turing machine indicates that the initial configuration is valid if and only if so is the final one. The validity has been called a standard configuration in [1, 2].

▶ **Definition 5** (Tidiness of Turing machine)**.** A k-tape Turing machine $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ is said to be *tidy* if for any sequence $(q_{\mathsf{ini}}, \langle l_1, s_1, r_1\rangle, \ldots, \langle l_k, s_k, r_k\rangle) \vdash_T^* (q_{\mathsf{fin}}, \langle l'_1, s'_1, r'_1\rangle, \ldots, \langle l'_k, s'_k, r'_k\rangle)$ of computation steps, the following two conditions

- $(l_i, s_i, r_i) \in \{\varepsilon\} \times \{\sqcup\} \times \Sigma^*$ for each $i = 1, \ldots, k$
- $(l'_i, s'_i, r'_i) \in \{\varepsilon\} \times \{\sqcup\} \times \Sigma^*$ for each $i = 1, \ldots, k$

are equivalent.

In the rest of the paper, every k-tape Turing machine is assumed to be tidy. We may call it the *tidiness assumption*.

We shall show two examples of Turing machines whose semantics are both idempotent. As we will see later, due to the form of their transition rules, the second example is an idempotent Turing machine but the first example is not. The main theorem of the present paper claims that any non-idempotent Turing machine has an equivalent idempotent Turing machine whenever its semantics is idempotent, though.

▶ **Example 6.** The 1-tape Turing machine $T_{\mathit{ralz}} = (Q, \{0, 1\}, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ where

$$Q = \{q_{\mathsf{ini}}, q_{\mathsf{move}}, q_{\mathsf{ralz}}, q_{\mathsf{back}}, q_{\mathsf{fin}}\}$$
$$\Delta = \{(q_{\mathsf{ini}}, \sqcup{\Rightarrow}\sqcup, q_{\mathsf{move}}), (q_{\mathsf{move}}, \rightarrow, q_{\mathsf{ralz}}), (q_{\mathsf{ralz}}, 0{\Rightarrow}\sqcup, q_{\mathsf{move}}), (q_{\mathsf{ralz}}, 1{\Rightarrow}1, q_{\mathsf{back}}), (q_{\mathsf{ralz}}, \sqcup{\Rightarrow}\sqcup, q_{\mathsf{back}}), (q_{\mathsf{back}}, \leftarrow, q_{\mathsf{fin}})\}$$

computes the function that removes all leading zeros, i.e., we have $[\![T_{\mathit{ralz}}]\!](0 \ldots 0w) = w$ for $w \in \{\varepsilon\} \cup \{1v \mid v \in \{0, 1\}^*\}$.

▶ **Example 7.** The 2-tape Turing machine $T_{\mathit{copy}} = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ where

$$Q = \{q_{\mathsf{ini}}, q_{\mathsf{move}}, q_{\mathsf{copy}}, q_{\mathsf{defer}}, q_{\mathsf{trail}}, q_{\mathsf{erase}}, q_{\mathsf{return}}, q_{\mathsf{back}}, q_{\mathsf{check}}, q_{\mathsf{fin}}\}$$
$$\Delta = \{(q_{\mathsf{ini}}, (\sqcup{\Rightarrow}\sqcup, \sqcup{\Rightarrow}\sqcup), q_{\mathsf{move}}), (q_{\mathsf{move}}, (\rightarrow, \rightarrow), q_{\mathsf{copy}})\} \cup$$
$$\{(q_{\mathsf{copy}}, (s_1{\Rightarrow}s_2, s_2{\Rightarrow}s_2), q_{\mathsf{move}}) \mid s_1 \in \Sigma_\sqcup, s_2 \in \Sigma\} \cup$$
$$\{(q_{\mathsf{copy}}, (s{\Rightarrow}s, \sqcup{\Rightarrow}\sqcup), q_{\mathsf{trail}}) \mid s \in \Sigma\} \cup$$
$$\{(q_{\mathsf{copy}}, (\sqcup{\Rightarrow}\sqcup, \sqcup{\Rightarrow}\sqcup), q_{\mathsf{back}}), (q_{\mathsf{trail}}, (\rightarrow, \rightarrow), q_{\mathsf{defer}})\} \cup$$

$$\{(q_{\mathsf{defer}}, (s{\Rightarrow}s, {\sqcup}{\Rightarrow}{\sqcup}), q_{\mathsf{trail}}) \mid s \in \Sigma\} \cup$$
$$\{(q_{\mathsf{defer}}, ({\sqcup}{\Rightarrow}{\sqcup}, {\sqcup}{\Rightarrow}{\sqcup}), q_{\mathsf{return}}), (q_{\mathsf{return}}, (\leftarrow, \leftarrow), q_{\mathsf{erase}})\} \cup$$
$$\{(q_{\mathsf{erase}}, (s{\Rightarrow}{\sqcup}, {\sqcup}{\Rightarrow}{\sqcup}), q_{\mathsf{return}}) \mid s \in \Sigma\} \cup \{(q_{\mathsf{erase}}, (s{\Rightarrow}s, s{\Rightarrow}s), q_{\mathsf{back}}) \mid s \in \Sigma_{\sqcup}\} \cup$$
$$\{(q_{\mathsf{check}}, (s{\Rightarrow}s, s{\Rightarrow}s), q_{\mathsf{back}}) \mid s \in \Sigma\} \cup \{(q_{\mathsf{back}}, (\leftarrow, \leftarrow), q_{\mathsf{check}}), (q_{\mathsf{check}}, ({\sqcup}{\Rightarrow}{\sqcup}, {\sqcup}{\Rightarrow}{\sqcup}), q_{\mathsf{fin}})\}$$

computes the function that copies the second string to the first, i.e., we have $[\![T_{\mathit{copy}}]\!](w_1, w_2) = (w_2, w_2)$ for $w_1, w_2 \in \Sigma^*$. The 2-tape Turing machine T_{copy} can be straightforwardly generalized into $2k$-tape Turing machine $T_{\mathit{copy}(k)}$ so that $[\![T_{\mathit{copy}(k)}]\!](w_1, \ldots, w_k, v_1, \ldots, v_k) = (v_1, \ldots, v_k, v_1, \ldots, v_k)$ holds for $w_1, \ldots, w_k, v_1, \ldots, v_k \in \Sigma^*$. In particular, $T_{\mathit{copy}(1)} = T_{\mathit{copy}}$.

The second example could be given with fewer states and transition rules by merging q_{defer} with q_{copy}, q_{trail} with q_{move}, q_{erase} with q_{check}, and q_{return} with q_{back}, and removing some redundant rules. However, the smaller alternative is against the condition to be an idempotent Turing machine which will be presented in the next section. The Turing machine T_{copy} of the present form will play an important role in the proof of expressiveness of idempotent Turing machines.

Definition 4 implies that the semantics of a Turing machine returns a tuple that consists of the same number of strings as a given input. However, when the function either only accepts or always returns the empty string on some tapes, we may regard it as a function whose input or output tuple consists of fewer strings following the formalization by Axelsen and Glück [2]. For example, let T be a 3-tape Turing machine over Σ such that $[\![T]\!](w_1, w_2, w_3) = (w'_1, w'_2, w'_3)$ implies $w_2 = w_3 = w'_2 = \varepsilon$. Then we may say that T *computes* a function $f : \Sigma^* \to \Sigma^* \times \Sigma^*$ defined by $f(w) = (v_1, v_2)$ where $[\![T]\!](w, \varepsilon, \varepsilon) = (v_1, \varepsilon, v_2)$ holds. We may simply write $[\![T]\!] = f$ by ignoring empty input/output strings.

▶ **Definition 8** (Forward/backward determinism). Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape Turing machine. Then T is *forward deterministic* if, for any distinct pair $(q, a_1, q_1), (q, a_2, q_2) \in \Delta$ of transition rules with the common source state $q \in Q$, their actions a_1 and a_1 have the form of $(s_{1,1}{\Rightarrow}s'_{1,1}, \ldots, s_{1,k}{\Rightarrow}s'_{1,k})$ and $(s_{2,1}{\Rightarrow}s'_{2,1}, \ldots, s_{2,k}{\Rightarrow}s'_{2,k})$, respectively, such that $s_{1,i} \neq s_{2,i}$ holds for some $i = 1, \ldots, k$. The Turing machine T is *backward deterministic* if, for any distinct pair $(q_1, a_1, q), (q_2, a_2, q) \in \Delta$ of transition rules with the common target state $q \in Q$, their actions a_1 and a_1 have the form of $(s_{1,1}{\Rightarrow}s'_{1,1}, \ldots, s_{1,k}{\Rightarrow}s'_{1,k})$ and $(s_{2,1}{\Rightarrow}s'_{2,1}, \ldots, s_{2,k}{\Rightarrow}s'_{2,k})$, respectively, such that $s'_{1,i} \neq s'_{2,i}$ holds for some $i = 1, \ldots, k$.

Example 6 and Example 7 are both forward deterministic but not backward deterministic. It is easy to see that every configuration step induced by a forward deterministic Turing machine is functional. In the rest of the paper, we deal with only forward deterministic Turing machines, and hence their semantics are all functional. We may simply say Turing machines even for forward deterministic ones.

Turing machines can be concatenated to synthesize a single one which computes the composition of their semantics.

▶ **Definition 9** (Concatenation of Turing machines). Let $\{T_i = (Q_i, \Sigma, q_{\mathsf{ini},i}, q_{\mathsf{fin},i}, \Delta_i)\}_{i=1,\ldots,n}$ be a family of k-tape Turing machines where $Q_1, \ldots, Q_n$ are disjoint without loss of generality. Their *concatenation*, denoted by $T_n \circ \cdots \circ T_1$, is a k-tape Turing machine $T = (Q_1 \uplus \cdots \uplus Q_n, \Sigma, q_{\mathsf{ini},1}, q_{\mathsf{fin},n}, \Delta)$ where $\Delta = \Delta_1 \uplus \cdots \uplus \Delta_n \uplus \{(q_{\mathsf{fin},i-1}, (\underbrace{\blacklozenge, \ldots, \blacklozenge}_{k}), q_{\mathsf{ini},i}) \mid i = 2, \ldots, n\}$.

When $Q_1, \ldots, Q_n$ are not disjoint, every state in either should be renamed before the concatenation.

▶ **Proposition 10** (Semantics of concatenation of Turing machines). *For k-tape reversible Turing machines $T_1, \ldots, T_n$, we have $[\![T_n \circ \cdots \circ T_1]\!] = [\![T_n]\!] \circ \cdots \circ [\![T_1]\!]$.*

Proof. It can be shown straightforwardly with taking notice of the tidiness assumption. ◀

3.2 Reversible Turing Machines

We define a reversible Turing machine, which can be used for the proof of expressiveness of idempotent Turing machines.

▶ **Definition 11** (Reversible Turing machine). A k-tape Turing machine T is *reversible* if T is forward and backward deterministic.

▶ **Example 12.** The 2-tape reversible Turing machine $T_{dup} = (\{q_{\mathsf{ini}}, q_{\mathsf{move}}, q_{\mathsf{copy}}, q_{\mathsf{back}}, q_{\mathsf{check}}, q_{\mathsf{fin}}\}, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ where

$$\begin{aligned}\Delta = {} & \{(q_{\mathsf{ini}}, (\text{␣}\Rightarrow\text{␣}, \text{␣}\Rightarrow\text{␣}), q_{\mathsf{move}}), (q_{\mathsf{move}}, (\rightarrow, \rightarrow), q_{\mathsf{copy}})\} \cup \\ & \{(q_{\mathsf{copy}}, (s\Rightarrow s, \text{␣}\Rightarrow s), q_{\mathsf{move}}) \mid s \in \Sigma\} \cup \{(q_{\mathsf{copy}}, (\text{␣}\Rightarrow\text{␣}, \text{␣}\Rightarrow\text{␣}), q_{\mathsf{back}})\} \cup \\ & \{(q_{\mathsf{check}}, (s\Rightarrow s, s\Rightarrow s), q_{\mathsf{back}}) \mid s \in \Sigma\} \cup \{(q_{\mathsf{back}}, (\leftarrow, \leftarrow), q_{\mathsf{check}}), (q_{\mathsf{check}}, (\text{␣}\Rightarrow\text{␣}, \text{␣}\Rightarrow\text{␣}), q_{\mathsf{fin}})\}\end{aligned}$$

computes the function $[\![T_{dup}]\!]$, which satisfies $[\![T_{dup}]\!](w, \varepsilon) = (w, w)$ for $w \in \Sigma^*$. The 2-tape reversible Turing machine T_{dup} can be straightforwardly generalized into $2k$-tape reversible Turing machine $T_{dup(k)}$ so that $[\![T_{dup(k)}]\!](w_1, \ldots, w_k, \underbrace{\varepsilon, \ldots, \varepsilon}_{k}) = (w_1, \ldots, w_k, w_1, \ldots, w_k)$ holds for $w_1, \ldots, w_k \in \Sigma^*$. In particular, we have $T_{dup(1)} = T_{dup}$.

Although the Turing machines T_{dup} and T_{copy} are similar in a sense that they both output pairs of the same string, T_{dup} differs in that the second component of its input is restricted to the empty string. Because of this difference, $[\![T_{dup}]\!]$ is not idempotent while so is $[\![T_{copy}]\!]$.

As seen from the definition, a Turing machine obtained by inverting all transition rules of a reversible Turing machine is also reversible. Its semantics is naturally the inverse function of the semantics of the original reversible Turing machine as formally stated below.

▶ **Definition 13** (Inversion of reversible Turing machines). Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape reversible Turing machine. We define $T^{-1} = (Q, \Sigma, q_{\mathsf{fin}}, q_{\mathsf{ini}}, \Delta^{-1})$ with $\Delta^{-1} = \{(p, a^{-1}, q) \mid (q, a, p) \in \Delta\}$ where $(a_1, \ldots, a_k)^{-1} = ({a_1}^{-1}, \ldots, {a_k}^{-1})$, $(s\Rightarrow s')^{-1} = (s'\Rightarrow s)$, $(\leftarrow)^{-1} = (\rightarrow)$, $(\blacklozenge)^{-1} = (\blacklozenge)$, and $(\rightarrow)^{-1} = (\leftarrow)$.

▶ **Proposition 14** (Bennet [3]). *Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape reversible Turing machine. Then, T^{-1} forms a k-tape reversible Turing machine such that $[\![T^{-1}]\!] = [\![T]\!]^{-1}$.*

In reversible Turing machines, by definition, there is no distinct pair of configurations that have the same successive configuration. This implies that the semantics of a reversible Turing machine is always injective.

Axelsen and Glück [2] have shown its converse, i.e., every injective computable function can be defined by a reversible Turing machine. Their proof is constructive so that an equivalent reversible Turing machine can be constructed effectively from a given non-reversible Turing machine whose semantics is injetive.

▶ **Theorem 15** (Expressiveness of reversible Turing machines [2]). *The reversible Turing machines can compute exactly all injective computable functions. That is, given a k-tape Turing machine T such that $[\![T]\!]$ is injective, there is a k-tape reversible Turing machine T' such that $[\![T']\!] = [\![T]\!]$.*

4 Idempotent Turing Machines

We introduce an idempotent Turing machine and its properties, expressiveness, robustness, and universality.

4.1 Definition and Expressiveness

An idempotent Turing machine is defined by imposing a restriction upon the form of transition rules of a standard Turing machine. Informally, the key of the restriction forces every valid run to include an internal configuration C such that the sequence of configuration steps from C to the final configuration can also become a valid run by concatenating its reversed sequence at the front. Note that in the obtained run, the initial and final tapes contain the same string. This indicates that $[\![T]\!](y) = y$ holds for any x and y such that $[\![T]\!](x) = y$ for the Turing machine T, which concludes that $[\![T]\!]$ is idempotent. Formally, idempotent Turing machines are defined as below.

▶ **Definition 16** (Idempotent Turing machine)**.** Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape forward deterministic Turing machine. Then T is said to be *idempotent* if there exist a set $Q' \subset Q$ of states and a total function $\psi : Q' \to Q \setminus Q'$ that satisfy the following conditions:

(I-1) $q_{\mathsf{ini}} \notin Q'$, $q_{\mathsf{fin}} \in Q'$, and $\psi(q_{\mathsf{fin}}) = q_{\mathsf{ini}}$;

(I-2) there is no transition rule $(q', a, q) \in \Delta$ with $q' \in Q'$ and $q \in Q \setminus Q'$;

(I-3) there is a transition rule $(\psi(p'), a^{-1}, \psi(q')) \in \Delta$ for each $(q', a, p') \in \Delta$ with $p', q' \in Q'$; and

(I-4) there is a transition rule $(\psi(q'), a_0, q') \in \Delta$ with a null action a_0 for each $(q, a, q') \in \Delta$ with $q \in Q \setminus Q'$ and $q' \in Q'$.

Each state $q' \in Q'$ is called a *rear state* and the function ψ is called a *rear state map*.

Definition 16 implies that every valid sequence of configuration steps of an idempotent Turing machine can be split into two parts of non-rear and rear states because of the **(I-1)** and **(I-2)** conditions. Moreover, with the **(I-3)** and **(I-4)** conditions, we can conclude that the semantics of an idempotent Turing machine is idempotent.

▶ **Theorem 17** (Semantics of idempotent Turing machine)**.** *Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape idempotent Turing machine. Then $[\![T]\!]$ is idempotent.*

Proof. For simplicity of the proof, only the case of $k = 1$ is shown. The proof can be easily generalized to the other cases. Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a 1-tape idempotent Turing machine with a set Q' of rear states and a rear state map ψ. It suffices to show that $[\![T]\!](v) = v$ holds for any $v = [\![T]\!](w)$ with $w \in \Sigma^*$. Suppose that $v = [\![T]\!](w)$ for some $v, w \in \Sigma^*$. Because of the definition of idempotent Turing machines, there must exist a valid sequence of configuration steps of the form

$$\begin{aligned}(q_{\mathsf{ini}}, \langle \varepsilon, \sqcup, w\rangle) \vdash (q_1, \langle l_1, s_1, r_1\rangle) \vdash \ldots \vdash (q_m, \langle l_m, s_m, r_m\rangle) \vdash \\ (q'_1, \langle l'_1, s'_1, r'_1\rangle) \vdash \ldots \vdash (q'_n, \langle l'_n, s'_n, r'_n\rangle) \vdash (q_{\mathsf{fin}}, \langle \varepsilon, \sqcup, v\rangle)\end{aligned}$$

with $q_1, \ldots, q_m \in Q \setminus Q'$ and $q'_1, \ldots, q'_n \in Q'$ from the **(I-1)** and **(I-2)** conditions. Because of the **(I-4)** condition, there is a transition rule $(\psi(q'_1), a_0, q'_1) \in \Delta$ with a null action a_0. Then we have a valid sequence

$$\begin{aligned}(q_{\mathsf{ini}}, \langle \varepsilon, \sqcup, v\rangle) \vdash (\psi(q'_n), \langle l'_n, s'_n, r'_n\rangle) \vdash \ldots \vdash (\psi(q'_1), \langle l'_1, s'_1, r'_1\rangle) \vdash \\ (q'_1, \langle l'_1, s'_1, r'_1\rangle) \vdash \ldots \vdash (q'_n, \langle l'_n, s'_n, r'_n\rangle) \vdash (q_{\mathsf{fin}}, \langle \varepsilon, \sqcup, v\rangle)\end{aligned}$$

due to the **(I-1)** and **(I-3)** conditions, which demonstrates $[\![T]\!](v) = v$. ◀

The Turing machine T_{copy} introduced in Example 7 can be shown to be idempotent accompanied by a set Q' of rear states and a rear state map ψ specified by

$$Q' = \{q_{\mathsf{fin}}, q_{\mathsf{check}}, q_{\mathsf{back}}\} \qquad \psi = \{q_{\mathsf{fin}} \mapsto q_{\mathsf{ini}}, q_{\mathsf{check}} \mapsto q_{\mathsf{move}}, q_{\mathsf{back}} \mapsto q_{\mathsf{copy}}\}.$$

The conditions **(I-1)** and **(I-2)** obviously hold. The condition **(I-3)** can be confirmed by the correspondence of the pairs of transition rules,

$$\begin{array}{ll} (q_{\mathsf{check}}, (\sqcup\Rightarrow\sqcup, \sqcup\Rightarrow\sqcup), q_{\mathsf{fin}}) & (q_{\mathsf{ini}}, (\sqcup\Rightarrow\sqcup, \sqcup\Rightarrow\sqcup), q_{\mathsf{move}}) \\ (q_{\mathsf{back}}, (\leftarrow, \leftarrow), q_{\mathsf{check}}) & (q_{\mathsf{move}}, (\rightarrow, \rightarrow), q_{\mathsf{copy}}) \\ (q_{\mathsf{check}}, (s\Rightarrow s, s\Rightarrow s), q_{\mathsf{back}}) & (q_{\mathsf{copy}}, (s\Rightarrow s, s\Rightarrow s), q_{\mathsf{move}}) \end{array}$$

with $s \in \Sigma$. The condition **(I-4)** holds because of the transition rule $(q_{\mathsf{copy}}, (\sqcup\Rightarrow\sqcup, \sqcup\Rightarrow\sqcup), q_{\mathsf{back}})$. The general Turing machine $T_{copy(k)}$ can be checked to be idempotent as well.

In contrast, the 1-tape Turing machine T_{ralz} introduced in Example 6 and the smaller alternative of T_{copy} mentioned after Example 7 are not idempotent even though their semantics is idempotent. As for T_{ralz}, we can check it as follows. In order to satisfy the conditions **(I-1)** and **(I-3)**, the q_{back} state cannot be a rear state. Then it is impossible to satisfy the condition **(I-4)**. In general we can decide whether a given Turing machine is idempotent.

▶ **Proposition 18** (Decidability of idempotence of Turing machines). *Let T be a k-tape Turing machine. It is decidable whether T is idempotent.*

Proof. The proof depends on the finiteness of the set of states and accordingly the finiteness of the choice of the set of rear states and the rear state map, which is required to be idempotent. ◀

The proof above indicates just the existence an extremely naive procedure for the decision problem. The complexity of the decision procedure is beyond double exponential to the number of states. We leave for future work a more efficient algorithm.

Note that the decision problem is only to decide if a Turing machine is idempotent but not to decide if the Turing machine computes an idempotent function. The latter problem is obviously undecidable. However, we will prove that an equivalent idempotent Turing machine can be constructed whenever the Turing machine computes an idempotent function.

Idempotent functions are closed under *conjugation* with injective functions, i.e., for any idempotent function f and injective function g, the conjugate $g^{-1} \circ f \circ g$ is idempotent. The following lemma shows that the idempotent Turing machines have a similar property which will be used to prove the main theorem. The proof of this lemma is given in Appendix A.

▶ **Lemma 19** (Closed under conjugation). *Let T be a k-tape idempotent Turing machine. For any k-tape reversible Turing machine T_r, the k-tape reversible Turing machine ${T_r}^{-1} \circ T \circ T_r$ is idempotent.*

Now we are ready to prove expressiveness of the idempotent Turing machines which is one of the main theorems in the present paper.

▶ **Theorem 20** (Expressiveness of idempotent Turing machines). *The idempotent Turing machines can compute any idempotent computable function. More specifically, given a k-tape Turing machine T such that $[\![T]\!]$ is idempotent, there is a $2k$-tape idempotent Turing machine T' such that $[\![T']\!] = [\![T]\!]$.*

Proof sketch. Let T be a k-tape Turing machine such that $[\![T]\!] : \Sigma^k \to \Sigma^k$ is idempotent. Consider a (partial) function $f : \Sigma^{2k} \to \Sigma^{2k}$ such that $f(w_1, \ldots, w_k, w_{k+1}, \ldots, w_{2k}) = (w_1, \ldots, w_k, [\![T]\!](w_1, \ldots, w_k))$ holds only if $[\![T]\!](w_1, \ldots, w_k)$ is defined and $w_{k+1} = \cdots = w_{2k} = \varepsilon$ holds. Since the function f is injective and computable, we can construct a $2k$-tape reversible Turing machine T_f such that $[\![T_f]\!] = f$ by Theorem 15. Then, define a $2k$-tape Turing machine $T' = {T_f}^{-1} \circ T_{\mathit{copy}(k)} \circ T_f$ where $T_{\mathit{copy}(k)}$ is a $2k$-tape idempotent Turing machine introduced in Example 7. The Turing machine T' is idempotent due to Lemma 19 and a simple calculation can verify $[\![T']\!] = [\![T]\!]$ as shown in Appendix A. ◀

4.2 Robustness under Tape Reduction

Single-tape Turing machines are as expressive as multitape Turing machines [17]. This property is known as one of the robustness of Turing machines. We will see the property for idempotent Turing machines, i.e., every multitape idempotent Turing machine has an equivalent single-tape idempotent Turing machine. To this end, we simulate a k-tuple of strings with a single string by an encoding function $\mathit{enc} : (\Sigma^*)^k \to (\Sigma \uplus \{\$\})^*$ using a special symbol $\$$ not in Σ. The encoding function is defined as

$$\mathit{enc}(s_{1,1}s_{1,2}\ldots s_{1,n}, \ldots, s_{k,1}s_{k,2}\ldots s_{k,n}) = s_{1,1}\ldots s_{k,1}s_{1,2}\ldots s_{k,2}s_{1,n}\ldots s_{k,n}$$

with the maximum length n of the input strings where the symbol $\$$ is filled at the end of the shorter strings as necessary, e.g., $\mathit{enc}(\mathtt{ab}, \mathtt{cdef}, \mathtt{ghi}) = \mathtt{acgbdh\$ei\$f\$}$. The encoding function is injective and computable where k is fixed. We will write $\Sigma_\$$ for $\Sigma \uplus \{\$\}$ and enc may be used even for encoded strings, i.e., $\mathit{enc} : (\Sigma_\$^*)^k \to \Sigma_\$^*$.

We first show how to construct a 2-tape idempotent Turing machine equivalent to a given multitape idempotent Turing machine. This is easily shown by the expressiveness of idempotent Turing machines.

▶ **Theorem 21** (Reduction to 2-tape idempotent Turing machine). *Let T be a k-tape idempotent Turing machine. Then there exists a 2-tape idempotent Turing machine T' that simulates T, that is, $[\![T]\!](w_1, \ldots, w_k) = (v_1, \ldots, v_k)$ if and only if $[\![T']\!](\mathit{enc}(w_1, \ldots, w_k), \varepsilon) = (\mathit{enc}(v_1, \ldots, v_k), \varepsilon)$.*

Proof. Let T be a k-tape idempotent Turing machine. Consider a function $f : \Sigma_\$^* \to \Sigma_\$^*$ satisfying $f(\mathit{enc}(w_1, \ldots, w_k)) = \mathit{enc}([\![T]\!](w_1, \ldots, w_k))$ for any $w_1, \ldots, w_k \in \Sigma^*$ only if $[\![T]\!](w_1, \ldots, w_k)$ is defined. Since f is computable, we have a 1-tape Turing machine that computes f. Note that f is idempotent because of the idempotence of $[\![T]\!]$. Therefore, there exists a 2-tape idempotent Turing machine that computes f by Theorem 20 with $k = 1$. ◀

The theorem above is not satisfactory because it still requires at least two tapes to simulate arbitrary multitape idempotent Turing machines. We need a further idea to show the robustness under the tape reduction down to a single tape. The idea is similar to that of the proof for expressiveness. We employ a 1-tape idempotent Turing machine T_{blur}, which behaves like T_{copy} as shown by the following lemma. Its proof is given in Appendix A.

▶ **Lemma 22.** *There exists a 1-tape idempotent Turing machine T_{blur} computing the idempotent function $f_{\mathit{blur}} : \Sigma_\$^* \to \Sigma_\$^*$, which satisfies $f_{\mathit{blur}}(\varepsilon) = \varepsilon$ and $f_{\mathit{blur}}(s_1 s_2 w) = s_1 s_1 f_{\mathit{blur}}(w)$ for any $s_1, s_2 \in \Sigma_\$$ and $w \in \Sigma_\$^*$.*

Now we are ready to prove the robustness under tape reduction for idempotent Turing machines. In the proof of the robustness theorem, the idempotent Turing machine T_{blur} plays a similar role to $T_{\mathit{copy}(k)}$ in the proof of Theorem 20.

▶ **Theorem 23** (Robustness of idempotent Turing machines). *Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape idempotent Turing machine. Then there exists a 1-tape idempotent Turing machine T' such that $[\![T]\!](w_1, \ldots, w_k) = (v_1, \ldots, v_k)$ if and only if $[\![T']\!](\mathsf{enc}(w_1, \ldots, w_k)) = \mathsf{enc}(v_1, \ldots, v_k)$.*

Proof sketch. Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape idempotent Turing machine. Consider a function $f : \Sigma_{\$}^* \to \Sigma_{\$}^*$ satisfying $f(\mathsf{enc}(w_1, \ldots, w_k)) = \mathsf{enc}(\mathsf{enc}([\![T]\!](w_1, \ldots, w_k)), \mathsf{enc}(w_1, \ldots, w_k))$ for any $w_1, \ldots, w_n \in \Sigma^*$ only if $[\![T]\!](w_1, \ldots, w_k)$ is defined. Since the function f is injective and computable, we can construct a 1-tape Turing machine T_f such that $[\![T_f]\!] = f$ by Theorem 15. Then, define a 1-tape Turing machine $T' = {T_f}^{-1} \circ T_{\mathit{blur}} \circ T_f$ where T_{blur} is an idempotent Turing machine given in Lemma 22. The Turing machine T' is idempotent because of Lemma 19 and a simple calculation can verify $[\![T']\!] \circ \mathsf{enc} = \mathsf{enc} \circ [\![T]\!]$ as shown in Appendix A. ◀

4.3 Universality

A standard Turing machine is called *universal* if it is capable of simulating an arbitrary Turing machine on arbitrary input. A universal Turing machine takes a pair of strings: one is the description of the given Turing machine T, which is typically provided as the Gödel number $\ulcorner T \urcorner$ as a string; another is an input w of T. Then it is expected to return the output string $[\![T]\!](w)$. In essence, a universal Turing machine U must satisfy $[\![U]\!](\ulcorner T \urcorner, x) = [\![T]\!](w)$ for any Turing machine T and its input string w.

With regard to idempotent Turing machines, there is no universal machines in the sense above, i.e., no idempotent Turing machine simulate arbitrary idempotent Turing machines. Since the domain and codomain of idempotent functions must be equal, the universal machine cannot be idempotent. Therefore, we relax the definition of universality as Axelsen and Glück have done to introduce the universality of reversible Turing machines [2] where the universal machine returns not only the expected output string but also the given Turing machine itself. Under this relaxed definition, the universal machine computes an idempotent function as far as the given Turing machine is idempotent.

▶ **Definition 24** (Universality). A k-tape idempotent Turing machine U is said to be *IdTM-universal* if $[\![U]\!](\ulcorner T \urcorner, w) = (\ulcorner T \urcorner, [\![T]\!](w))$ holds for any idempotent Turing machine T and its input string w.

In the present paper, a universal model of ordinary Turing machines is called a *classically universal Turing machine* to distinguish from our IdTM-universal machines.

It is not difficult to show the existence of an IdTM-universal machine because of the expressiveness of idempotent Turing machines.

▶ **Theorem 25.** *There exists an IdTM-universal idempotent Turing machine.*

Proof. Let f be a function satisfying $f(\ulcorner T \urcorner, w) = (\ulcorner T \urcorner, [\![T]\!](w))$ for any idempotent Turing machine T and its input string w. Note that f is idempotent because $f(f(\ulcorner T \urcorner, w)) = f(\ulcorner T \urcorner, [\![T]\!](w)) = (\ulcorner T \urcorner, [\![T]\!]([\![T]\!](w))) = (\ulcorner T \urcorner, [\![T]\!](w))$ where the last equality comes from the idempotence of $[\![T]\!]$. By Theorem 20, we obtain an idempotent Turing machine U that computes f. ◀

The theorem above just shows the existence of an universal machine, which is considered impractical as noticed in the prior work [1, 2, 14] because its proof relies on Theorem 15 (via Theorem 20), which is the expressive theorem of reversible Turing machine. As mentioned in [2], the proof of Theorem 15 is based on very inefficient generate-and-test method by

McCarthy [12]. To avoid this problem, we shall show the construction of an universal Turing machine without Theorem 15 where we construct a universal idempotent Turing machine from a classically universal Turing machine. Following the existing work [1,2,14], we employ Bennett's trick with Landauer's trace embedding to construct a special type of reversible Turing machines that computes the function $\lambda x.(x, f(x))$ for an arbitrary computable function f. We present a multitape version of the theorem as below. Its proof is provided in Appendix A.

▶ **Proposition 26** (Bennett's trick [3]). *Let T be a k-tape Turing machine. There exists a $(2k+1)$-tape reversible Turing machine $\mathsf{Ben}_k(T)$ such that $[\![\mathsf{Ben}_k(T)]\!](w_1, \ldots, w_k, \underbrace{\varepsilon, \ldots, \varepsilon}_{k+1}) = (w_1, \ldots, w_k, [\![T]\!](w_1, \ldots, w_k), \varepsilon)$ for any input $w_1, \ldots, w_k$ of T.*

With the Bennett's trick, we can construct a universal idempotent Turing machine from a classically universal Turing machine.

▶ **Theorem 27** (IdTM-universal Turing machine constructed with Bennett's trick). *Let U be a (2-tape) classically universal Turing machine, i.e., $[\![U]\!](\ulcorner T \urcorner, w) = (\ulcorner T \urcorner, [\![T]\!](w))$ for any Turing machine T and its input w. Then a 5-tape idempotent Turing machine $U' = \mathsf{Ben}_2(U)^{-1} \circ T'_{\mathit{copy}(2)} \circ \mathsf{Ben}_2(U)$ is IdTM-universal where $T'_{\mathit{copy}(2)}$ is obtained by adding one extra tape as the fifth tape to $T_{\mathit{copy}(2)}$, i.e., $[\![T'_{\mathit{copy}(2)}]\!](w_1, w_2, w_3, w_4, w_5) = (w_3, w_4, w_3, w_4, w_5)$.*

Proof sketch. Let U be a 2-tape classically universal Turing machine such that $[\![U]\!](\ulcorner T \urcorner, w) = (\ulcorner T \urcorner, [\![T]\!](w))$ for any Turing machine T and its input w. Note that the Turing machine $U' = \mathsf{Ben}_2(U)^{-1} \circ T'_{\mathit{copy}(2)} \circ \mathsf{Ben}_2(U)$ is idempotent due to the idempotence of $T'_{\mathit{copy}(2)}$ and Lemma 19. In addition, we can show the 5-tape idempotent Turing machine U' is IdTM-universal, that is, $[\![U']\!](\ulcorner T \urcorner, w, \varepsilon, \varepsilon, \varepsilon) = (\ulcorner T \urcorner, [\![T]\!](w), \varepsilon, \varepsilon, \varepsilon)$ holds for any Turing machine T and its input w as shown in Appendix A. ◀

The author [14] has constructed a universal machine for involutory Turing machines using Bennett's trick in a way similar to Theorem 27. The difference is that he uses the Turing machine permuting some of tapes for the center one of the composition instead of $T'_{\mathit{copy}(2)}$.

5 Related work

This work proposes idempotent Turing machines, which can compute exactly all idempotent computable functions. The present work has followed along the lines of prior work on special Turing machines for particular classes of computable functions [1,2,14] under function semantics, in which the meaning of a Turing machine is specified by a function whose input and output correspond to strings on the tape at the initial and final configuration, respectively.

Axelsen and Glück [1,2] have investigated several properties of reversible Turing machines under function semantics. Even though the notion of reversible Turing machines had been already introduced and studied before the work [3,11], Axelsen and Glück gave much clearer semantics to reversible Turing machines to observe what they compute. They showed that reversible Turing machines can compute exactly all injective computable functions. They also proved the robustness under tape and symbol reduction: 1-tape 3-symbol reversible Turing machines can simulate arbitrary multitape reversible Turing machines. Furthermore, they showed the existence of universal reversible Turing machines under an appropriate redefinition of universality and gave an efficient construction of a universal machine. The present work has followed their approach and utilized their results even though idempotent functions are not necessarily injective.

The author [14] introduced involutory Turing machines and showed that they can compute exactly all involutory computable functions, which are own inverse, i.e., a function f such that $f(f(x)) = x$ holds whenever $f(x)$ is defined. He naturally followed Axelsen and Glück's work since every involutory function is injective. He defined an involutory Turing machine as a Turing machine whose transition rules are related each other under an involutory map over states. The idea is found in discrete time-symmetric systems [6, 9]. In the present work, we refer to the idea to define an idempotent Turing machine with the rear state map.

Besides Turing machines, there are many other model of computable functions, e.g., untyped lambda calculus, combinatory logic, and term/string rewriting systems. We could consider a restricted model of them for idempotent functions. We have selected Turing machines because there is a well-studied subclass of the model, namely reversible Turing machine, for injective functions that can be invertible. A candidate of other such models would be Mu et al's injective language Inv [13] in which every function is defined by a few primitives including the fixed-point operator. They have shown that the Inv language is expressive enough to simulate reversible Turing machines. However, it is not simple to utilize the injective language for a computational model of idempotent functions. We need to add more primitives to describe non-injective functions and impose some syntactic restrictions to define only idempotent functions. It would be interesting if such a programming language can be defined, though.

6 Conclusion

We have introduced a computation model for idempotent functions, called an idempotent Turing machine. This model is necessarily and sufficiently expressive: every idempotent Turing machine computes an idempotent function and every idempotent function can be computed by an idempotent Turing machine. The class of Turing machines has been shown to be robust under tape reduction. We have also shown the existence of an universal idempotent Turing machine and its construction.

Our computation model is expected to be a basis of special-purpose (or domain-specific) programming languages in which only but all idempotent computable functions can be defined. A computation model is said to be *Turing-complete* if it can simulate any Turing machine. The notion of Turing-completeness is often used to show the expressiveness of not only a computation model but also a programming language or a set of machine instructions. For reversible computation, the notion of *r-Turing completeness* has been proposed [1,2,19]. There have been reversible programming languages, e.g., Janus with dynamic storage [18], reversible flowchart language [19], and R-WHILE [7], that have been shown to be r-Turing complete. Similarly, the notion of *idempotent-Turing-completeness* can be defined and applied to a special-purpose (or domain-specific) programming languages in which only but all idempotent computable functions can be defined. Such a special programming language will be required for string sanitizers, automated program optimizers, and bidirectional transformation.

In addition, it is interesting to consider computational models that exactly cover all computable functions with some constraints in the general case. Axelsen and Glück [1] have shown that reversible Turing machines can exactly cover injective computable functions. The author [14] has shown a computational model that exactly covers involutory computable functions, and he shows such a result for idempotent functions in the present paper. It is a natural question to ask what kind of semantic constraints on computable functions corresponds to syntactically constrained Turing machines in general, which is left for future work.

References

1. Holger Bock Axelsen and Robert Glück. What Do Reversible Programs Compute? In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 42–56, 2011. doi:10.1007/978-3-642-19805-2_4.
2. Holger Bock Axelsen and Robert Glück. On reversible Turing machines and their function universality. *Acta Inf.*, 53(5):509–543, 2016. doi:10.1007/s00236-015-0253-y.
3. Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973. doi:10.1147/rd.176.0525.
4. Sebastian Fischer, Zhenjiang Hu, and Hugo Pacheco. The essence of bidirectional programming. *Sci. China Inf. Sci.*, 58(5):1–21, 2015. doi:10.1007/s11432-015-5316-8.
5. J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17, 2007. doi:10.1145/1232420.1232424.
6. Anahí Gajardo, Jarkko Kari, and Andrés Moreira. On time-symmetry in cellular automata. *J. Comput. Syst. Sci.*, 78(4):1115–1126, 2012. doi:10.1016/j.jcss.2012.01.006.
7. Robert Glück and Tetsuo Yokoyama. A Linear-Time Self-Interpreter of a Reversible Imperative Language. *Computer Software*, 33(3):3_108–3_128, 2016. doi:10.11309/jssst.33.3_108.
8. Pieter Hooimeijer, Benjamin Livshits, David Molnar, Prateek Saxena, and Margus Veanes. Fast and Precise Sanitizer Analysis with BEK. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.
9. Martin Kutrib and Thomas Worsch. Time-Symmetric Machines. In *Reversible Computation - 5th International Conference, RC 2013, Victoria, BC, Canada, July 4-5, 2013. Proceedings*, pages 168–181, 2013. doi:10.1007/978-3-642-38986-3_14.
10. R. Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961. doi:10.1147/rd.53.0183.
11. Yves Lecerf. Machines de Turing réversibles. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 257:2597–2600, 1963.
12. John McCarthy. The Inversion of Functions Defined by Turing Machines. In *Automata Studies. (AM-34)*, pages 177–182. Princeton University Press, 1956. doi:10.1515/9781400882618-009.
13. Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. An Injective Language for Reversible Computation. In Dexter Kozen and Carron Shankland, editors, *Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings*, volume 3125 of *Lecture Notes in Computer Science*, pages 289–313. Springer, 2004. doi:10.1007/978-3-540-27764-4_16.
14. Keisuke Nakano. Involutory Turing Machines. In Ivan Lanese and Mariusz Rawski, editors, *Reversible Computation - 12th International Conference, RC 2020, Oslo, Norway, July 9-10, 2020, Proceedings*, volume 12227 of *Lecture Notes in Computer Science*, pages 54–70. Springer, 2020. doi:10.1007/978-3-030-52482-1_3.
15. Keisuke Nakano. A Tangled Web of 12 Lens Laws. In Shigeru Yamashita and Tetsuo Yokoyama, editors, *Reversible Computation - 13th International Conference, RC 2021, Virtual Event, July 7-8, 2021, Proceedings*, volume 12805 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2021. doi:10.1007/978-3-030-79837-6_11.
16. OWASP. Double Encoding. https://owasp.org/www-community/Double_Encoding. [Online; 21-January-2021].
17. Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, Boston, MA, 1997.
18. Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Principles of a reversible programming language. In *Proceedings of the 5th Conference on Computing Frontiers, 2008, Ischia, Italy, May 5-7, 2008*, pages 43–54, 2008. doi:10.1145/1366230.1366239.

19 Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Reversible Flowchart Languages and the Structured Reversible Program Theorem. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 258–270. Springer, 2008. doi:10.1007/978-3-540-70583-3_22.

A Proofs of Lemmas, Theorems and Proposition

This Appendix provides proofs that are omitted or condensed in the main text.

▶ **Lemma 19** (Closed under conjugation). *Let T be a k-tape idempotent Turing machine. For any k-tape reversible Turing machine T_r, the k-tape reversible Turing machine ${T_r}^{-1} \circ T \circ T_r$ is idempotent.*

Proof. Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape idempotent Turing machine with a set $Q' \subset Q$ of rear states and a rear state map ψ. Let $T_r = (Q_r, \Sigma_r, q_{\mathsf{ini},r}, q_{\mathsf{fin},r}, \Delta_r)$ be a k-tape reversible Turing machine where Q and Q_r are disjoint without loss of generality. In order to concatenate T_r, T, and ${T_r}^{-1}$ following Definition 9, we rename every state $q \in Q_r$ of ${T_r}^{-1}$ with $\widehat{q}$. Let $\widehat{Q_r}$ be a set of states of ${T_r}^{-1}$, i.e., $\widehat{Q_r} = \{\widehat{q} \mid q \in Q_r\}$ and ${\Delta_r}^{-1} = \{(\widehat{p}, a^{-1}, \widehat{q}) \mid (q, a, p) \in \Delta_r\}$. Then we can define $T_c = {T_r}^{-1} \circ T \circ T_r$, which is to be shown idempotent, as $T_c = (Q_c, \Sigma, q_{\mathsf{ini},r}, \widehat{q_{\mathsf{ini},r}}, \Delta_c)$ where

$$Q_c = Q_r \uplus Q \uplus \widehat{Q_r}$$
$$\Delta_c = \Delta_r \uplus \Delta \uplus \{(\widehat{p}, a^{-1}, \widehat{q}) \mid (q, a, p) \in \Delta_r\} \uplus \{(q_{\mathsf{fin},r}, (\underbrace{\blacklozenge, \ldots, \blacklozenge}_{k}), q_{\mathsf{ini}}), (q_{\mathsf{fin}}, (\underbrace{\blacklozenge, \ldots, \blacklozenge}_{k}), \widehat{q_{\mathsf{fin},r}})\}.$$

Let Q'_c be a subset of Q_c defined by $Q' \uplus \widehat{Q_r}$. We define a function $\psi_c : Q'_c \to Q_c \setminus Q'_c$ as

$$\psi_c(q) = \begin{cases} \psi(q) & (q \in Q') \\ q_r & (q = \widehat{q_r} \in \widehat{Q_r}) \end{cases}$$

where the codomain of ψ_c is $(Q \setminus Q') \uplus Q_r$ that is equal to $Q_c \setminus Q'_c$.

We shall check the four conditions of Q'_c and ψ_c for T_c to be idempotent.

- The **(I-1)** condition holds since $q_{\mathsf{ini},r} \in Q_r \subset Q_c \setminus Q'_c$, $\widehat{q_{\mathsf{ini},r}} \in \widehat{Q_r} \subset Q'_c$, and $\psi_c(\widehat{q_{\mathsf{ini},r}}) = q_{\mathsf{ini},r}$.
- The **(I-2)** condition holds because of the definition of concatenation and the **(I-2)** condition for T.
- Concerning the **(I-3)** condition, suppose that we have $(q', a, p') \in \Delta_c$ with $p', q' \in Q'_c = Q' \uplus \widehat{Q_r}$. Since it is impossible to have $p' \in Q' \subset Q$ and $q' \in \widehat{Q_r}$ due to the construction of Δ_c, we divide it into three cases:
  - When $p', q' \in Q'$, we have $(\psi_c(p'), a^{-1}, \psi_c(q')) \in \Delta_c$ because of the **(I-3)** condition of T with $\psi_c(p') = \psi(p')$, $\psi_c(q') = \psi(p')$, and $\Delta \subset \Delta_c$;
  - When $p' \in \widehat{Q_r}$ and $q' \in Q'$, we must have $p' = \widehat{q_{\mathsf{fin},r}}$ and $q' = q_{\mathsf{fin}}$ according to the construction of Δ_c. Hence, we have $(\psi_c(p'), a^{-1}, \psi_c(q')) \in \Delta_c$ because of $\psi_c(p') = q_{\mathsf{fin},r}$, $\psi_c(q') = \psi(q_{\mathsf{fin}}) = q_{\mathsf{ini}}$, and the construction of Δ_c; and
  - When $p', q' \in \widehat{Q_r}$, there exist $p_r, q_r \in Q_r$ such that $p' = \widehat{p_r}$ and $q' = \widehat{q_r}$. Hence, we have $(\psi_c(p'), a^{-1}, \psi_c(q')) \in \Delta_c$ because of $\psi_c(p') = p_r$, $\psi_c(q') = q_r$, and the construction of Δ_c with Δ_r.

  Thus, the condition **(I-3)** holds.

- Concerning **(I-4)** condition, suppose that we have $(q, a, q') \in \Delta_c$ with $q \in (Q_c \setminus Q') = (Q \setminus Q' \uplus Q_r)$ and $q' \in Q'_c = Q' \uplus \widehat{Q_r}$. From the construction of Δ_c, we must have $q \in Q \setminus Q'$ and $q' \in Q'$. Thus, we obtain $q = \psi(q') = \psi_c(q')$ and $a = a^{-1}$ because of the condition **(I-4)** of T.

Therefore we conclude that T_c is idempotent with the set Q_c of rear states and the rear state map ψ_c. ◀

▶ **Theorem 20** (Expressiveness of idempotent Turing machines)**.** *The idempotent Turing machines can compute any idempotent computable function. More specifically, given a k-tape Turing machine T such that $[\![T]\!]$ is idempotent, there is a $2k$-tape idempotent Turing machine T' such that $[\![T']\!] = [\![T]\!]$.*

Proof. Let T be a k-tape Turing machine such that $[\![T]\!] : \Sigma^k \to \Sigma^k$ is idempotent. Consider a (partial) function $f : \Sigma^{2k} \to \Sigma^{2k}$ such that $f(w_1, \ldots, w_k, w_{k+1}, \ldots, w_{2k}) = (w_1, \ldots, w_k, [\![T]\!](w_1, \ldots, w_k))$ holds only if $[\![T]\!](w_1, \ldots, w_k)$ is defined and $w_{k+1} = \cdots = w_{2k} = \varepsilon$ holds. Since the function f is injective and computable, we can construct a $2k$-tape reversible Turing machine T_f such that $[\![T_f]\!] = f$ by Theorem 15.

Let us define a $2k$-tape Turing machine $T' = {T_f}^{-1} \circ T_{\mathit{copy}(k)} \circ T_f$ where $T_{\mathit{copy}(k)}$ is a $2k$-tape idempotent Turing machine introduced in Example 7. Since T' is idempotent because of Lemma 19, it suffices to show $[\![T']\!] = [\![T]\!]$, i.e., $[\![T']\!](w_1, \ldots, w_k, \underbrace{\varepsilon, \ldots, \varepsilon}_{k}) = ([\![T]\!](w_1, \ldots, w_k), \underbrace{\varepsilon, \ldots, \varepsilon}_{k})$. This can be shown by

$$
\begin{aligned}
[\![T']\!](w_1, \ldots, w_k, \varepsilon, \ldots, \varepsilon) &= \{ \text{ by the definition of } T' \text{ and Proposition 10 } \} \\
&\quad [\![{T_f}^{-1}]\!]([\![T_{\mathit{copy}(k)}]\!]([\![T_f]\!](w_1, \ldots, w_k, \varepsilon, \ldots, \varepsilon))) \\
&= \{ \text{ by the definition of } T_f \text{ and Proposition 14 } \} \\
&\quad [\![T_f]\!]^{-1}([\![T_{\mathit{copy}(k)}]\!](w_1, \ldots, w_k, [\![T]\!](w_1, \ldots, w_k))) \\
&= \{ \text{ by the definition of } T_{\mathit{copy}(k)} \ \} \\
&\quad [\![T_f]\!]^{-1}([\![T]\!](w_1, \ldots, w_k), [\![T]\!](w_1, \ldots, w_k)) \\
&= \{ \text{ by the idempotence of } [\![T]\!] \ \} \\
&\quad [\![T_f]\!]^{-1}([\![T]\!](w_1, \ldots, w_k), [\![T]\!]([\![T]\!](w_1, \ldots, w_k))) \\
&= \{ \text{ by the semantics of } T_f \text{ and its injectivity } \} \\
&\quad ([\![T]\!](w_1, \ldots, w_k), \varepsilon, \ldots, \varepsilon).
\end{aligned}
$$
◀

▶ **Lemma 22.** *There exists a 1-tape idempotent Turing machine T_{blur} computing the idempotent function $f_{\mathit{blur}} : \Sigma_{\$}^* \to \Sigma_{\$}^*$, which satisfies $f_{\mathit{blur}}(\varepsilon) = \varepsilon$ and $f_{\mathit{blur}}(s_1 s_2 w) = s_1 s_1 f_{\mathit{blur}}(w)$ for any $s_1, s_2 \in \Sigma_{\$}$ and $w \in \Sigma_{\$}^*$.*

Proof. Let $T = (Q, \Sigma_{\$}, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a 1-tape Turing machine where

$$
\begin{aligned}
Q &= \{q_{\mathsf{ini}}, q_{\mathsf{mov}}, q_{\mathsf{read}}, q_{\mathsf{back}}, q_{\mathsf{pick}}, q_{\mathsf{fin}}\} \cup \bigcup_{s \in \Sigma_{\$}} \{q_{\mathsf{mem}(s)}, q_{\mathsf{write}(s)}, q_{\mathsf{keep}(s)}, q_{\mathsf{check}(s)}\} \\
\Delta &= \{(q_{\mathsf{ini}}, \sqcup \Rightarrow \sqcup, q_{\mathsf{mov}}), (q_{\mathsf{mov}}, \rightarrow, q_{\mathsf{read}}), (q_{\mathsf{pick}}, \sqcup \Rightarrow \sqcup, q_{\mathsf{fin}}), (q_{\mathsf{back}}, \leftarrow, q_{\mathsf{pick}}), (q_{\mathsf{read}}, \sqcup \Rightarrow \sqcup, q_{\mathsf{back}})\} \cup \\
&\quad \{(q_{\mathsf{read}}, s \Rightarrow s, q_{\mathsf{mem}(s)}) \mid s \in \Sigma_{\$}\} \cup \{(q_{\mathsf{mem}(s)}, \rightarrow, q_{\mathsf{write}(s)}) \mid s \in \Sigma_{\$}\} \cup \\
&\quad \{(q_{\mathsf{write}(s)}, s' \Rightarrow s, q_{\mathsf{mov}}) \mid s, s' \in \Sigma_{\$}\} \cup \{(q_{\mathsf{pick}}, s \Rightarrow s, q_{\mathsf{keep}(s)}) \mid s \in \Sigma_{\$}\} \cup \\
&\quad \{(q_{\mathsf{keep}(s)}, \leftarrow, q_{\mathsf{check}(s)}) \mid s \in \Sigma_{\$}\} \cup \{(q_{\mathsf{check}(s)}, s \Rightarrow s, q_{\mathsf{back}}) \mid s \in \Sigma_{\$}\}.
\end{aligned}
$$

Then T is found to be idempotent by the set $Q' \subset Q$ of rear states and the rear state map ψ defined as

$$Q' = \{q_{\mathsf{fin}}, q_{\mathsf{pick}}, q_{\mathsf{back}}\} \cup \bigcup_{s \in \Sigma_{\$}} \{q_{\mathsf{check}(s)}, q_{\mathsf{keep}(s)}\}$$

$$\psi = \{q_{\mathsf{fin}} \mapsto q_{\mathsf{ini}}, q_{\mathsf{pick}} \mapsto q_{\mathsf{mov}}, q_{\mathsf{back}} \mapsto q_{\mathsf{read}}\} \cup \bigcup_{s \in \Sigma_{\$}} \{q_{\mathsf{check}(s)} \mapsto q_{\mathsf{mem}(s)}, q_{\mathsf{keep}(s)} \mapsto q_{\mathsf{write}(s)}\}$$

since the four conditions obviously hold. We can check that T computes the function f_{blur} in the statement as follows. Firstly the relation

$$(q_{\mathsf{read}}, \langle \varepsilon, s, ws'w' \rangle) \vdash^* (q_{\mathsf{read}}, \langle \overleftarrow{f_{\mathit{blur}}(sw)}, s', w' \rangle)$$

holds for any $s, s' \in \Sigma_{\$}$ and $w, w' \in \Sigma_{\$}^*$ where the length of w is odd and $\overleftarrow{x}$ denotes the reversed string of x, which can be proved by induction on the length of w. Moreover the relation

$$(q_{\mathsf{back}}, \langle \overleftarrow{sw}, s', w' \rangle) \vdash^* (q_{\mathsf{back}}, \langle \varepsilon, s, ws'w' \rangle)$$

holds for any $s, s' \in \Sigma_{\$}$ and $w, w' \in \Sigma_{\$}^*$ where the length of w and w' are odd and the strings sw and $s'w'$ are in the domain of f_{blur}, i.e., every even-numbered symbol in them is the same as the previous symbol. This can also be proved by induction on the length of w. Then, we have the relation

$$(q_{\mathsf{ini}}, \langle \varepsilon, \sqcup, w \rangle) \vdash^* (q_{\mathsf{fin}}, \langle \varepsilon, \sqcup, f_{\mathit{blur}}(w) \rangle)$$

whenever the length of w is even, which indicates that T computes f_{blur}. ◀

▶ **Theorem 23** (Robustness of idempotent Turing machines). *Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape idempotent Turing machine. Then there exists a 1-tape idempotent Turing machine T' such that $[\![T]\!](w_1, \ldots, w_k) = (v_1, \ldots, v_k)$ if and only if $[\![T']\!](\mathit{enc}(w_1, \ldots, w_k)) = \mathit{enc}(v_1, \ldots, v_k)$.*

Proof. Let $T = (Q, \Sigma, q_{\mathsf{ini}}, q_{\mathsf{fin}}, \Delta)$ be a k-tape idempotent Turing machine. Consider a function $f : \Sigma_{\$}^* \to \Sigma_{\$}^*$ satisfying

$$f(\mathit{enc}(w_1, \ldots, w_k)) = \mathit{enc}(\mathit{enc}([\![T]\!](w_1, \ldots, w_k)), \mathit{enc}(w_1, \ldots, w_k))$$

for any $w_1, \ldots, w_n \in \Sigma^*$ only if $[\![T]\!](w_1, \ldots, w_k)$ is defined. Since the function f is injective and computable, we can construct a 1-tape Turing machine T_f such that $[\![T_f]\!] = f$ by Theorem 15.

Let us define a 1-tape Turing machine $T' = {T_f}^{-1} \circ T_{\mathit{blur}} \circ T_f$ where the idempotent Turing machine T_{blur} is given by Lemma 22. Note that

$$f_{\mathit{blur}}(\mathit{enc}(x, y)) = \mathit{enc}(x, x) \tag{1}$$

holds for any $x, y \in \Sigma_{\$}^*$. Since T' is idempotent because of Lemma 19, it suffices to show T' simulates T under encoding with enc. This can be shown by

$$[\![T']\!](\mathit{enc}(w_1, \ldots, w_k)) = \{ \text{ by the definition of } T' \text{ and Proposition 10 } \} \\ [\![{T_f}^{-1}]\!]([\![T_{\mathit{blur}}]\!]([\![T_f]\!](\mathit{enc}(w_1, \ldots, w_k))))$$

$$
\begin{aligned}
&= \{ \text{ by the semantics of } T_f \text{ and Proposition 14 } \} \\
&\quad [\![T_f]\!]^{-1}([\![T_{\mathit{blur}}]\!](\mathit{enc}(\mathit{enc}([\![T]\!](w_1,\ldots,w_k))), \mathit{enc}(w_1,\ldots,w_k))) \\
&= \{ \text{ by the semantics of } T_{\mathit{blur}} \text{ with Equation (1) } \} \\
&\quad [\![T_f]\!]^{-1}(\mathit{enc}(\mathit{enc}([\![T]\!](w_1,\ldots,w_k))), \mathit{enc}([\![T]\!](w_1,\ldots,w_k))) \\
&= \{ \text{ by the idempotence of } T \text{ and Theorem 17 } \} \\
&\quad [\![T_f]\!]^{-1}(\mathit{enc}(\mathit{enc}([\![T]\!]([\![T]\!](w_1,\ldots,w_k)))), \mathit{enc}([\![T]\!](w_1,\ldots,w_k))) \\
&= \{ \text{ by the injectivity of } f \text{ and the semantics of } f^{-1} \text{ } \} \\
&\quad \mathit{enc}([\![T]\!](w_1,\ldots,w_k)).
\end{aligned}
$$

◀

▶ **Proposition 26** (Bennett's trick [3]). *Let T be a k-tape Turing machine. There exists a $(2k+1)$-tape reversible Turing machine $\mathsf{Ben}_k(T)$ such that $[\![\mathsf{Ben}_k(T)]\!](w_1,\ldots,w_k,\underbrace{\varepsilon,\ldots,\varepsilon}_{k+1}) = (w_1,\ldots,w_k,[\![T]\!](w_1,\ldots,w_k),\varepsilon)$ for any input $w_1,\ldots,w_k$ of T.*

Proof. Let $T = (Q,\Sigma,q_{\mathsf{ini}},q_{\mathsf{fin}},\Delta)$ be a k-tape Turing machine. Firstly, by Landauer embedding [10], we can construct a $(k+1)$-tape Turing machine T_L such that

$$[\![T_L]\!](w_1,\ldots,w_k,\varepsilon) = ([\![T]\!](w_1,\ldots,w_k), \mathit{trace}(T,w_1,\ldots,w_k))$$

where the function *trace* encodes the history of applied transition rules on the run into Σ^*. Since the history tells the previous configuration for each step, the Turing machine T_L is backward deterministic, that is, reversible. We extend T_L with k extra tapes in between working tapes and the history tape where the extra tapes are never touched during computation. Let T'_L be the $(2k+1)$-reversible Turing machine obtained by the extension. Then we have

$$[\![T'_L]\!](w_1,\ldots,w_k,v_1,\ldots,v_k,\varepsilon) = ([\![T]\!](w_1,\ldots,w_k),v_1,\ldots,v_k,\mathit{trace}(T,w_1,\ldots,w_k))$$

for $w_1,\ldots,w_k,v_1,\ldots,v_k \in \Sigma^*$. In addition, we similarly extend the $2k$-tape Turing machine $T_{\mathit{dup}(k)}$ with one extra tape to obtain $(2k+1)$-tape Turing machine $T'_{\mathit{dup}(k)}$ such that

$$[\![T'_{\mathit{dup}(k)}]\!](w_1,\ldots,w_k,\underbrace{\varepsilon,\ldots,\varepsilon}_{k},v) = (w_1,\ldots,w_k,w_1,\ldots,w_k,v)$$

holds for $w_1,\ldots,w_k,v \in \Sigma^*$.

Let us define the $(2k+1)$-tape Turing machine $\mathsf{Ben}_k(T) = {T'_L}^{-1} \circ T'_{\mathit{dup}(k)} \circ T'_L$ which is reversible because so are all constituents. Because of the domain of $T_{\mathit{dup}(k)}$ and the semantics of T'_L, the input of the Turing machine $\mathsf{Ben}_k(T)$ is restricted to $(2k+1)$-tuples of strings whose $(k+1)$-th through $2k+1$-th strings are the empty string. Therefore the equation in the statement can be checked by

$$
\begin{aligned}
&[\![\mathsf{Ben}_k(T)]\!](w_1,\ldots,w_k,\underbrace{\varepsilon,\ldots,\varepsilon}_{k},\varepsilon) \\
&= \{ \text{ by the definition of } \mathsf{Ben}_k(T) \text{ and Proposition 10 } \} \\
&\quad [\![{T'_L}^{-1}]\!]([\![T'_{\mathit{dup}(k)}]\!]([\![T'_L]\!](w_1,\ldots,w_k,\underbrace{\varepsilon,\ldots,\varepsilon}_{k},\varepsilon))) \\
&= \{ \text{ by the semantics of } T'_L \text{ and Proposition 14 } \} \\
&\quad [\![T'_L]\!]^{-1}([\![T'_{\mathit{dup}(k)}]\!]([\![T]\!](w_1,\ldots,w_k),\underbrace{\varepsilon,\ldots,\varepsilon}_{k},\mathit{trace}(T,w_1,\ldots,w_k)))
\end{aligned}
$$

$$
\begin{aligned}
&= \{ \text{ by the semantics of } T'_{\mathit{dup}(k)} \ \} \\
&\quad [\![T'_L]\!]^{-1}([\![T]\!](w_1,\ldots,w_k), [\![T]\!](w_1,\ldots,w_k), \mathit{trace}(T,w_1,\ldots,w_k)) \\
&= \{ \text{ by the inverse of the semantics of } T'_L \ \} \\
&\quad (w_1,\ldots,w_k,[\![T]\!](w_1,\ldots,w_k),\varepsilon). \qquad \blacktriangleleft
\end{aligned}
$$

▶ **Theorem 27** (IdTM-universal Turing machine constructed with Bennett's trick)**.** *Let U be a (2-tape) classically universal Turing machine, i.e., $[\![U]\!](\ulcorner T \urcorner, w) = (\ulcorner T \urcorner, [\![T]\!](w))$ for any Turing machine T and its input w. Then a 5-tape idempotent Turing machine $U' = \mathsf{Ben}_2(U)^{-1} \circ T'_{\mathit{copy}(2)} \circ \mathsf{Ben}_2(U)$ is IdTM-universal where $T'_{\mathit{copy}(2)}$ is obtained by adding one extra tape as the fifth tape to $T_{\mathit{copy}(2)}$, i.e., $[\![T'_{\mathit{copy}(2)}]\!](w_1,w_2,w_3,w_4,w_5) = (w_3,w_4,w_3,w_4,w_5)$.*

Proof. Let U be a 2-tape classically universal Turing machine such that $[\![U]\!](\ulcorner T \urcorner, w) = (\ulcorner T \urcorner, [\![T]\!](w))$ for any Turing machine T and its input w. Note that the Turing machine $U' = \mathsf{Ben}_2(U)^{-1} \circ T'_{\mathit{copy}(2)} \circ \mathsf{Ben}_2(U)$ is idempotent due to Lemma 19. We shall show the 5-tape idempotent Turing machine U' is IdTM-universal, that is, $[\![U']\!](\ulcorner T \urcorner, w, \varepsilon, \varepsilon, \varepsilon) = (\ulcorner T \urcorner, [\![T]\!], \varepsilon, \varepsilon, \varepsilon)$ holds for any Turing machine T and its input w. Because of the domain of $\mathsf{Ben}_2(U)$, the input of the Turing machine U' is restricted to 5-tuples whose third through fifth are the empty string. Therefore, the statement can be verified by

$$
\begin{aligned}
[\![U']\!](\ulcorner T \urcorner, w, \varepsilon, \varepsilon, \varepsilon) &= \{ \text{ by the definition of } U' \text{ and Proposition 10 } \} \\
&\quad [\![\mathsf{Ben}_2(U)^{-1}]\!]([\![T'_{\mathit{copy}(2)}]\!]([\![\mathsf{Ben}_2(U)]\!](\ulcorner T \urcorner, w, \varepsilon, \varepsilon, \varepsilon))) \\
&= \{ \text{ by the semantics of } \mathsf{Ben}_2(U) \text{ and } U \ \} \\
&\quad [\![\mathsf{Ben}_2(U)^{-1}]\!]([\![T'_{\mathit{copy}(2)}]\!](\ulcorner T \urcorner, w, \ulcorner T \urcorner, [\![T]\!](w), \varepsilon)) \\
&= \{ \text{ by the semantics of } T'_{\mathit{copy}(2)} \ \} \\
&\quad [\![\mathsf{Ben}_2(U)^{-1}]\!](\ulcorner T \urcorner, [\![T]\!](w), \ulcorner T \urcorner, [\![T]\!](w), \varepsilon) \\
&= \{ \text{ by the idempotence of } T \text{ and Theorem 17 } \} \\
&\quad [\![\mathsf{Ben}_2(U)^{-1}]\!](\ulcorner T \urcorner, [\![T]\!](w), \ulcorner T \urcorner, [\![T]\!]([\![T]\!](w)), \varepsilon) \\
&= \{ \text{ by the inverse of the semantics of } \mathsf{Ben}_2(U) \ \} \\
&\quad (\ulcorner T \urcorner, [\![T]\!](w), \varepsilon, \varepsilon, \varepsilon). \qquad \blacktriangleleft
\end{aligned}
$$