Decision Problems on Iterated Length-Preserving Transducers

Alan Pierce
Advisor: Klaus Sutner
April 29, 2011

Abstract

Finite-state transducers are simple theoretical machines that are useful in expressing easily-computable functions and relations. This investigation considers the relations formed when transducers are iterated arbitrarily many times, a construction which is useful in model checking. In particular, we consider a number of decision problems over various classes of transducers, and attempt to determine whether each decision problem is decidable. For example, a Turing machine construction can show that the string reachability problem on arbitrary iterated transducers reduces from the halting problem, and is thus undecidable. This setup is useful for investigating how restricted a transducer must be before its iteration is no longer able to simulate a Turing machine (for different notions of simulation). The main result of the paper is that both reset transducers—which have a highly restricted concept of memory—and binary toggle transducers—which must express all letter transformations as permutations—are capable of simulating a Turing machine computation.

1 Introduction

Finite-state transducers are reasonably simple machines that, in a sense, capture the notion of an easily-computable function (or, more generally, an easily-computable relation). From a theoretical standpoint, transducers are important because they generalize the theory of regular languages to the theory of rational sets and relations, which are obtained by a natural algebraic construction, as seen in [3]. In practice, transducers have been used in various areas of computer science, including natural language processing and model checking.

This paper focuses on the relation formed when a transducer is iterated arbitrarily many times. In particular, if τ is the relation computed by a transducer, then its iteration, τ^*, is the reflexive transitive closure of τ. This construction is especially useful in model checking: for example, if the memory of a system is modeled as a string x, and the transition system is modeled as a transducer with transduction τ, then deciding properties of τ^* is equivalent to reasoning about the long-term behavior of the system.

It is easy to show that a fully-general transducer is able to compute a single step of a Turing machine, so an iterated transducer is able to simulate a Turing machine running for arbitrarily-long amounts of time. Thus, many meaningful properties of general iterated transducers are undecidable. A number of partial algorithms have been developed for understanding iterated transducers when possible, such as in [5]. This paper takes more of a theoretical approach and focuses on showing various problems to be undecidable.

One goal of this paper is to determine how restricted a transducer must be before it can no longer simulate a Turing machine. In papers such as [9] and [4], the definition of an iterated transducer is modified slightly so that it recognizes languages rather than transductions. Using this definition the class of all iterated transducers generates the class of recursively enumerable languages. Our approach is different: we keep the definition of a transducer intact, and show that iterated transducers can simulate Turing machines by showing certain decision problems to be undecidable.

The main contribution of this paper is in showing that several highly-restricted variants of transducers are still capable of simulating Turing machines. In particular, we show that certain problems about binary toggle transducers and about reset transducers are undecidable.

The remainder of the paper is organized as follows. Section 2 provides basic definitions, defines the types of transducers that will be considered, and introduces several meaningful decision problems that will be considered. Section 3 states and proves some facts that are not particularly difficult, but provide context and are useful in the later results. Section 4 contains the main results of the paper. Section 5 discusses conclusions and poses future problems.

2 Preliminaries

2.1 Definitions

The reader is assumed to be familiar with basic formal language theory and computability theory; the purpose of this section is to fix definitions and notation. For general background information on the subject, see [6] and [11].

If Σ is a finite set of symbols (an *alphabet*), then Σ^* refers to the set of strings over the alphabet Σ. ε denotes the empty string.

A *deterministic finite automaton* (DFA), usually denoted D, is a simple type of finite state machine which either accepts or rejects any string given as an input.

A *language*, usually denoted L, is a set of strings. The language recognized by a DFA D, denoted $\mathcal{L}(D)$, is the set of all strings accepted by D. A *regular* language, usually denoted R, is a language in which $R = \mathcal{L}(D)$ for some DFA D. The precise definition of a DFA will not be necessary for this paper; we will only need the concept of regular and non-regular languages. We will also use well-known non-regular languages, such as $\{a^n b^n \mid n \in \mathbb{N}\}$.

A *transducer*, usually denoted T, is a 5-tuple $(Q, \Sigma, \delta, I, F)$. Q is a finite set of states, Σ is a finite alphabet, $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the transition relation, I is the set of initial states, and F is the set of final states. Intuitively, each element of the transition relation reads some character, writes some character, and changes state. A transition $(q, \sigma_1, \sigma_2, s) \in \delta$ is written as $q\sigma_1 \to \sigma_2 s$. A pair of strings (x, y) is *accepted* by T if there is some transition path from a state in I to a state in F such that the "read" symbols concatenate to x and the "write" symbols concatenate to y. Note that, in general, the transition relation may be nondeterministic.

A *transduction*, usually denoted τ, is a relation on strings (that is, a subset of $\Sigma^* \times \Sigma^*$ for some alphabet Σ). If $x, y \in \Sigma^*$, then the notation $x\tau y$ means that $(x, y) \in \tau$. The notation $x\tau$ means the language $\{y \mid x\tau y\}$. Similarly, if L is a language, then $L\tau$ refers to the language $\{y \mid x\tau y \text{ for some } x \in L\ \}$.

We will also consider sequential transductions: functional length-preserving transductions τ such that for any $n \in \mathbb{N}$ and $x \in \Sigma^*$ the first n letters in $x\tau$ depend only on the first n characters of x. In this case, if we have an infinite string $s \in \Sigma^\omega$, then $s\tau$ is the infinite string $t \in \Sigma^\omega$ such that for any $k \in \mathbb{N}$, if x is the prefix of length k of s, then $x\tau$ is the prefix of length k of t.

If T is a transducer, then $\mathcal{T}(T)$ refers to the transduction τ consisting of all ordered pairs of strings accepted by τ. A transduction τ is *rational* if $\tau = \mathcal{T}(T)$ for some transducer T.

If τ is a transduction, then τ^* refers to the transitive reflexive closure of τ. That is, $x\tau^*y$ if and only if there exists some finite sequence of strings $z_0, \ldots, z_n$, where $x = z_0$, $y = z_n$, and for each $i < n$, $z_i \tau z_{i+1}$.

A *Turing machine*, usually denoted M, is a 5-tuple $(Q, \Sigma, \delta, q_0, q_H)$, where Q is the state set, Σ is a set of symbols which must include the blank symbol $\Box$, $\delta : Q \times \Sigma \to \Sigma \times \{\mathrm{L}, \mathrm{R}\} \times Q$ is the transition function, q_0 is the start state, and q_H is the halt state. The basic semantics are well-known, so they are not described in detail here. Instead, we provide the important features of the formulation of Turing machines that we will use:

- The tape is one-way infinite. That is, each Turing machine is implicitly modified so that it will never move to the left of the start of the tape. It is well-known that this does not change the expressiveness of Turing machines.
- Transitions must move the tape head either left or right; there is no option to remain on the same tape cell.
- For this paper, we only need to consider a Turing machine run with the empty string as input. We do not need to use Turing machines which recognize sets or compute functions. Thus, we simply have a single halt state.

RE and co-RE refer to the recursively enumerable languages and their complements, respectively. Let $K_0 = \{M \mid M \text{ halts when given the input } \varepsilon\ \}$. It is well-known that K_0 is RE-complete. Thus, if there is a many-one reduction from K_0 to a particular set L, then L is RE-hard, and thus undecidable.

2.2 Classes of Transducers

This paper is concerned with a number of decision problems about a variety of classes of transducers. The following classes will be considered:

- **TRANS** refers to the set of all transducers.
- **LP** refers to the set of length-preserving transducers. A transducer T is *length-preserving* if for every $(x, y) \in \mathcal{T}(T)$, $\mathrm{len}(x) = \mathrm{len}(y)$.

The following transducers generate maps that are *sequential*—that is, for any string $x \in \Sigma^*$ the first n letters of $x\tau$ depend only on the first n letters of x.

- **ALPHA** refers to the set of alphabetic transducers. A transducer T is *alphabetic* if for every transition $q\sigma_1 \to \sigma_2 s$, neither σ_1 nor σ_2 is ε. In addition, the transition relation must be deterministic (that is, there is exactly one transition of the form $q\sigma_1 \to \sigma_2 s$ for each q and σ_1). In addition, it must be the case that $|I| = 1$ (that is, there is a unique start state), and it must be the case that $F = Q$. These restrictions ensure that exactly one output string is generated for each input string. Informally, an alphabetic transducer requires each transition to be an exact letter-for-letter replacement, and "looking ahead" is not allowed. Notice that alphabetic transducers are always length-preserving.

- **RESET** refers to the set of reset transducers. A transducer is a *reset transducer* if it is alphabetic and there is some function $\pi : \Sigma \to Q$, such that every transition is of the form $q\sigma_1 \to \sigma_2\pi(\sigma_1)$. That is, the transition state for a reset transducer is exactly determined by the input character, so reset transducers cannot remember information except the most recently read character.

  Reset transducers have been useful in algebraic automata theory; see [8] for additional information.

  It can be seen that reset transducers correspond to one-way cellular automata. A construction similar to one in [1] is presented for reset transducers, although the exact setup is not the same.

- **REV** refers to the set of reversible transducers. A transducer is *reversible* if it is alphabetic and for each state $q \in Q$, there is some permutation $\pi_q : \Sigma \to \Sigma$ such that every transition is of the form $q\sigma \to \pi_q(\sigma)s$. One consequence of this restriction is that a transducer state is no longer able to "write" character; it can only perform a permutation on the alphabet and then branch to a different state based on what character was read. Notice that by swapping the input/output characters for a reversible transducer, the transducer remains reversible and computes the inverse relation of the original transducer. This means that if T is reversible, then $\mathcal{T}(T)$ is a bijection on Σ^*.

- **BTT** refers to the set of binary toggle transducers. A *binary toggle transducer* is a reversible transducer over the alphabet $\{0, 1\}$. Note that in such a transducer, each state is either an "identity" state or a "toggle" state, depending on which of the two permutations on $\{0, 1\}$ is chosen for that state.

  Binary toggle transducers have a close connection to group theory, as described in [10] and [7].

The following example of a binary toggle transducer will be useful in later proofs.

Example 1. The binary toggle transducer in Figure 1 computes the increment function on $\{0, 1\}^*$, treating its input in reverse binary. In this example, and all future examples of transducers, the notation σ_1/σ_2 means that σ_1 is read and σ_2 is written.

2.3 Decision Problems

This section defines all of the decision problems that will be considered. Each one is defined with respect to an arbitrary class of transducers.

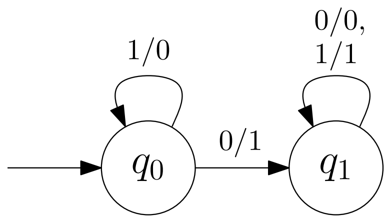


Figure 1: BTT for increment function.

Because a number of decision problems will be considered, we introduce a system for concisely describing decision problems. A decision problem is described as $L\,[p]$, where p is a predicate. The existence of certain variables implies that certain values appear as parameters to the decision problem:

- C refers to a class of transducers. This is not a parameter to the decision problem; the actual class must be specified for the decision problem to be fully defined.
- For each instance of τ (or τ_1 or τ_2), a transducer T is included as a parameter, and it is assumed that $\tau = \mathcal{T}(T)$.
- Σ refers to the alphabet of T.
- If Γ appears, then it is assumed to be included as a parameter, and it is assumed that $\Gamma \subseteq \Sigma$.
- For each instance of R (or S), a DFA D is included as a parameter, and it is assumed that $\mathcal{L}(D) = R$.
- Each instance of a, b, etc. is assumed to be an element of Σ that is specified as a parameter.
- Each instance of x, y, etc. is assumed to be a string in Σ^*, specified as a parameter.
- An instance of B denotes a language of the form

  $$\bigcup_{n\in\mathbb{N}} ((x^n y)^* z)^*$$

  where x, y, and z are arbitrary strings in Σ^*. We call any such language a *constant block language*. Whenever B appears in the definition of a decision problem, it is assumed that $\langle x, y, z\rangle$ is included as a parameter, thus uniquely specifying the language B.

  Constant block languages will be useful for the proofs involving reversible transducers.

We now present some specific examples of decision problems and the motivation for studying them.

- The reachability problem is defined as follows:

  $$L\,[x\tau^* y]_C = \{\langle T, x, y\rangle \mid T \in C,\ x, y \in \Sigma^*, \text{ and } x\mathcal{T}(T)^* y\}$$

  This is one of the most basic questions to ask about an iterated transducer; it simply queries membership in the iterated transduction.

- The following problem is useful for determining whether an iterated transducer computes a "simple" relation:

  $$L\left[\mathrm{Rat}\left(\tau^*\right)\right]_C = \{\langle T\rangle \mid\ T \in C \text{ and } \mathcal{T}(T)^* \text{ is rational }\}$$

  One hope when analyzing a system modeled by an iterated transducer is that it can be characterized by a transducer without iteration. It is shown in the paper that for general enough transducers, it is impossible to determine whether such a characterization exists.

- The following problem asks about the behavior of a transducer when applied to all strings in a language:

  $$L\left[R\tau^* \subseteq \Gamma^*\right]_C = \{\langle D, T, \Gamma\rangle \mid D \text{ is a DFA, } T \in C,\ \Gamma \subseteq \Sigma, \text{ and } \mathcal{L}(D)\mathcal{T}(T)^* \subseteq \Gamma^*\}$$

  One use of this setup is to let R be the allowed starting configurations for a system. We then may wish to check that an "error" symbol is never reachable, so we would let $\Gamma = \Sigma - \{\text{error}\}$. This setup also is useful for demonstrating that length-preserving transducers can simulate Turing machines. By letting R be the set of all lengths of empty tape, we know that $R\tau^*$ contains a string with halting state as a character if and only if there is some length of tape causing a Turing machine to halt.

3 Preliminary Results

The results presented in this section follow relatively easily, and are either useful for later results or are useful in motivating later results.

First, I will provide a small example of a Turing machine, which will be used throughout the paper to demonstrate simulations.

Example 2. In various examples throughout the paper, we will consider the Turing machine M, defined by Figure 2, run on the input ε. The alphabet is $\{\square, 0, 1\}$. This machine is chosen because it runs for relatively few steps and performs all interesting actions that a Turing machine would do. It is not meant to compute anything meaningful.

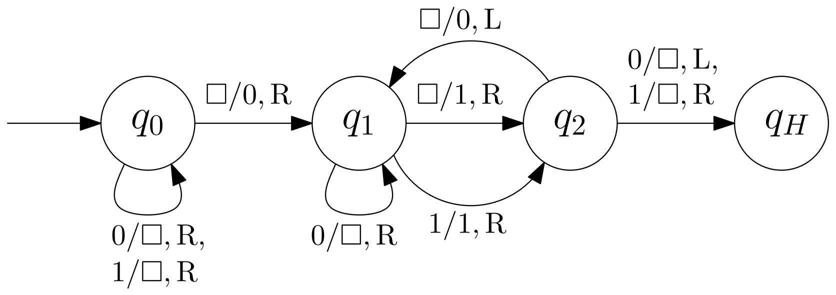


Figure 2: State diagram for the Turing machine M in Example 2.

We define the *one-step relation* for a Turing machine M as the relation mapping instantaneous configurations of the machine M to the configuration after one computation step, with all trailing $\square$ symbols removed so that each string is finite. In addition, ε maps to the string q_0.

Example 3. In the following trace of the Turing machine M from Example 2, each string is related to the next by the one-step relation on M.

(empty)			
q_0			
0	q_1		
0	1	q_2	
0	q_1	1	0
0	1	q_2	0
0	q_H	1	

Lemma 4. *For each fixed Turing machine M, the one-step relation on M is rational.*

Proof idea. The construction here is straightforward and well-known, so it will only be outlined. The transducer reads several characters before writing anything. Then, it copies the characters read in order. Because there are only finitely many possible transformations, a finite state machine can easily compute the new area around the tape head. If the tape ends with $q\square$ for any state q and the Turing machine moves left, the $\square$ character is erased. ■

Similar to the one-step relation, we define the *length-preserving one-step relation* τ_M for a Turing machine M to have the same behavior as the one-step relation, except that it can read tape strings with trailing $\square$ characters and preserves length. If the tape head points to a character past the end of the string, then the state symbol is replaced with $\square$. In addition, for any $k > 0$, $\square^k$ maps to $q_0\square^{k-1}$. An example is given in 5.

The choice of τ_M is useful because $\square^*\tau_M^*$ contains a string with the halt state q_H of M if and only if M halts when run on the empty string ε.

Example 5. If M is the Turing machine from Example 2, and τ_M is the length-preserving one-step relation for M, then τ_M iterated on $\square^7$ gives the following result. There is no particular significance of $\square^7$; a starting string of $\square^k$ for any $k \geq 4$ gives a similar trace.

$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
q_0	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
0	q_1	$\square$	$\square$	$\square$	$\square$	$\square$
0	1	q_2	$\square$	$\square$	$\square$	$\square$
0	q_1	1	0	$\square$	$\square$	$\square$
0	1	q_2	0	$\square$	$\square$	$\square$
0	q_H	1	$\square$	$\square$	$\square$	$\square$

Lemma 6. *The length-preserving one-step relation is rational.*

Proof idea. Again, this construction is relatively straightforward. The same steps are done as in the construction from Lemma 4, but $\square$ symbols are added to the end to ensure that the mapping is length-preserving. The fact that the transducer can be nondeterministic means that it can handle the case of $\square^k$ and the case of the tape head running off the end of the string as special cases. ■

We now focus on the reachability problem, which can be easily characterized.

Lemma 7. *$L\left[x\tau^*y\right]_{\mathbf{TRANS}}$ is undecidable. In particular, it is* RE*-complete.*

Proof idea. We reduce from K_0. Let M be a Turing machine, and let τ be the transducer provided by Lemma 4 for M. WLOG, assume that M erases its tape contents and moves to the start of the tape just before halting. Let x be the string ε and let y be the string containing q_H as its only character. Then $x\tau^*y$ if and only if M halts on input ε. So, we have a many-one reduction from K_0 to $L\left[x\tau^*y\right]_{\mathbf{TRANS}}$. It can also easily be seen that $L\left[x\tau^*y\right]_{\mathbf{TRANS}}$ is r.e., so $L\left[x\tau^*y\right]_{\mathbf{TRANS}}$ is RE-complete. ■

Lemma 8. *$L\left[x\tau^*y\right]_{\mathbf{LP}}$ is decidable.*

Proof idea. Let T be a given length-preserving transducer, and let $\tau = \mathcal{T}(T)$. If $x \in \Sigma^*$ is fixed, then we can iterate τ repeatedly on x until we find a cycle. Because τ is length-preserving, the cycle length is at most $|\Sigma|^{\mathrm{len}(x)}$, so the algorithm must always halt. We can then simply check membership in the cycle. ■

4 Undecidability Results

4.1 Overview

This section proves the main results of the paper. They are summarized in the following table. U means that the problem is undecidable and D means that the problem is decidable. Where applicable, the corresponding result in this paper is referenced.

	TRANS	**LP**	**ALPHA**	**RESET**	**REV**	**BTT**
$L\left[x\tau^*y\right]$	U (Lem. 7)	D (Lem. 8)	D	D	D	D
$L\left[\mathrm{Rat}\left(\tau^*\right)\right]$	U (Lem. 9)	U (Thm. 12)				
$L\left[\tau_1^* = \tau_2\right]$	U (Cor. 10)					
$L\left[\tau_1^* = \tau_2^*\right]$	U (Cor. 11)					
$L\left[\mathrm{Reg}\left(R\tau^*\right)\right]$	U	U	U (Cor. 17)			
$L\left[R\tau^* \subseteq \Gamma^*\right]$	U	U	U (Cor. 15)	U (Cor. 20)		
$L\left[B\tau^* \subseteq \Gamma^*\right]$	U	U	U	U	U (Thm. 22)	
$L\left[B\tau^* \subseteq R\right]$	U	U	U	U	U	U (Thm. 24)

Note that, generally, transducer classes to the right are special cases of ones to the left. The two exceptions are the neither reversible nor binary toggle transducers are special cases of reset transducers. So, a problem being decidable means that all problems to the right are decidable, and a problem being undecidable means that all problems to the left are undecidable.

4.2 Decision Problems for Characterizing Iterated Transducers

A natural goal when studying transducers is to find a simple characterization of an iterated transduction. That is, given a transducer T, we would like to find a transducer recognizing $\mathcal{T}(T)^*$, whenever this is possible. As the following results show, even for length-preserving transducers, it is undecidable to determine if such a transducer exists.

Lemma 9 shows this for the case of general transducers. Theorem 12 shows the stronger result for length-preserving transducers.

Lemma 9. $L\left[\mathrm{Rat}\left(\tau^*\right)\right]_{\mathbf{TRANS}}$ *is undecidable. In particular, it is* RE-*hard.*

Proof idea. Given a Turing machine, M, we map all strings under τ^* to the string ε, then have ε map to each string in the language $\{a^n b^n \mid n \in \mathbb{N}\}$ (where a and b are not used elsewhere). We also have ε start computing the Turing machine M on the empty input. Any string containing the halt state q_H maps to the set of all strings. This construction causes τ^* to be rational if and only if M halts on the empty input. ■

Proof. Let M be an arbitrary Turing machine. It's sufficient to show that there exists a transducer T with transduction τ such that τ^* is rational if and only if M halts on the empty input ε.

Let T be defined by the state diagram in Figure 3.

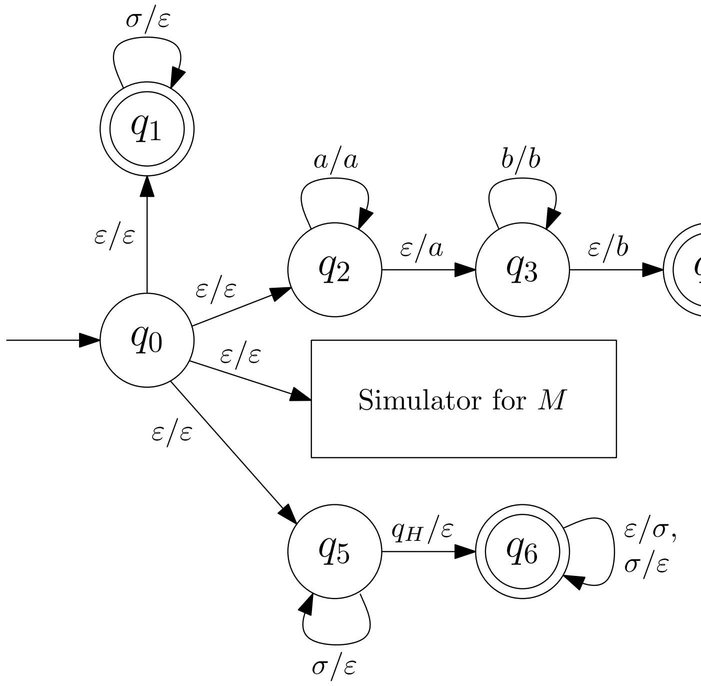


Figure 3: State diagram for the transducer T used in the proof of Lemma 9.

The letters a and b are chosen to be distinct from any states or tape symbols in M. The letter q_H is the halting state in M. The simulator for M is a tansducer with transduction ρ such that $\varepsilon\rho^*$

contains a string containing the halting state q_H if and only if M halts on the empty input ε. We know that some such transducer exists by Lemma 4.

In showing the correctness of T, there are two cases to consider:

M **halts on input** ε**:** In this case, we claim that τ^* is the universal relation $\Sigma^* \times \Sigma^*$. Let $x \in \Sigma^*$. Then $x\tau\varepsilon$ using states q_0 and q_1. Using the simulator for M, we know that $\varepsilon\tau^*z$ for some $z \in \Sigma^*$ containing the symbol q_H. Using states q_0, q_5, and q_6, we know that $z\tau^*y$ for any $y \in \Sigma^*$. By transitivity, this means that $x\tau^*y$ for any y. The universal relation is trivially rational, so τ^* is rational in this case.

M **does not halt on input** ε**:** In this case, we claim that τ^* is not rational. Because ε does not map under ρ^* to any string containing the symbol q_H, we know that the computation for $\varepsilon\tau^*$ will never reach state q_6 in T, so the letters a and b will only be generated by states q_2, q_3, and q_4. It is easy to see that the states q_0, q_2, q_3, and q_4, when applied to the empty string, generate the strings of the form a^nb^n for all $n \in \mathbb{N}$.

Suppose for a contradiction that τ^* is rational. Then by basic closure properties, $\varepsilon\tau^* \cap \{a, b\}^*$ is regular. By the work above, this means that $\{a^nb^n \mid n \in \mathbb{N}\}$ is regular, which is a contradiction. So, it must be the case that τ^* is not rational.

So, we have that τ^* is rational if and only if M halts on ε, so we have shown a many-one reduction from K_0 to $L\left[\mathrm{Rat}\left(\tau^*\right)\right]_{\mathbf{TRANS}}$, so $L\left[\mathrm{Rat}\left(\tau^*\right)\right]_{\mathbf{TRANS}}$ is RE-hard. ■

We can easily extend this proof to other decision problems involving the characterization of iterated transducers.

Corollary 10. *$L\left[\tau_1^* = \tau_2\right]_{\mathbf{TRANS}}$ is* RE-*hard.*

Proof. Using the proof in Lemma 9, given a Turing machine M, we can construct a transducer T with transduction τ such that $\tau^* = \Sigma^* \times \Sigma^*$ if and only if M halts on input ε. So, letting $T_1 = T$ and letting T_2 be a transducer recognizing $\Sigma^* \times \Sigma^*$, we know that $\mathcal{T}(T_1)^* = \mathcal{T}(T_2)$ if and only if M halts on input ε.

Thus, there is a many-one reduction from K_0 to $L\left[\tau_1^* = \tau_2\right]_{\mathbf{TRANS}}$, so $L\left[\tau_1^* = \tau_2\right]_{\mathbf{TRANS}}$ is RE-hard. ■

Corollary 11. *$L\left[\tau_1^* = \tau_2^*\right]_{\mathbf{TRANS}}$ is* RE-*hard.*

Proof. Let M be a Turing machine. Choosing the same transducers T_1 and T_2 from Corollary 10, we know that $\mathcal{T}(T_2) = \Sigma^* \times \Sigma^*$, so $\mathcal{T}(T_2)^* = \Sigma^* \times \Sigma^*$. So, $\mathcal{T}(T_1)^* = \mathcal{T}(T_2)^*$ if and only if M halts on input ε. So, $L\left[\tau_1^* = \tau_2^*\right]_{\mathbf{TRANS}}$ reduces from the halting problem K_0, so $L\left[\tau_1^* = \tau_2^*\right]_{\mathbf{TRANS}}$ is RE-hard. ■

We now extend Lemma 9 to length-preserving transducers. Note that this theorem is similar to the standard proof that it is undecidable to determine if a linear-bounded automaton recognizes the empty language.

Theorem 12. *$L\left[\mathrm{Rat}\left(\tau^*\right)\right]_{\mathbf{LP}}$ is* RE-*hard.*

Proof idea. This is essentially the same proof as the proof of Lemma 9, but the details are more involved. The transducer constructed has the following properties:

- For each $n \in \mathbb{N}$, all strings of length n map to $\Box^n$.
- $a^i b^i \Box^{n-2i}$ maps to $a^{i+1} b^{i+1} \Box^{n-2(i+1)}$.
- $\Box^n$ simulates the Turing machine M until the simulation has used n tape cells.
- Strings containing q_H map to every string of the same length.

The proof of non-rationality is the same as in Lemma 9. If M halts, then the transduction created is well-behaved for all strings longer than some finite length, so it is rational. ■

Proof. The setup for the proof is nearly identical to the proof of Lemma 9. We let M be an arbitrary Turing machine, and we wish to construct a transducer T such that $\mathcal{T}(T)^*$ is rational if and only if M halts on input ε. The state diagram in Figure 4 defines the transducer T.

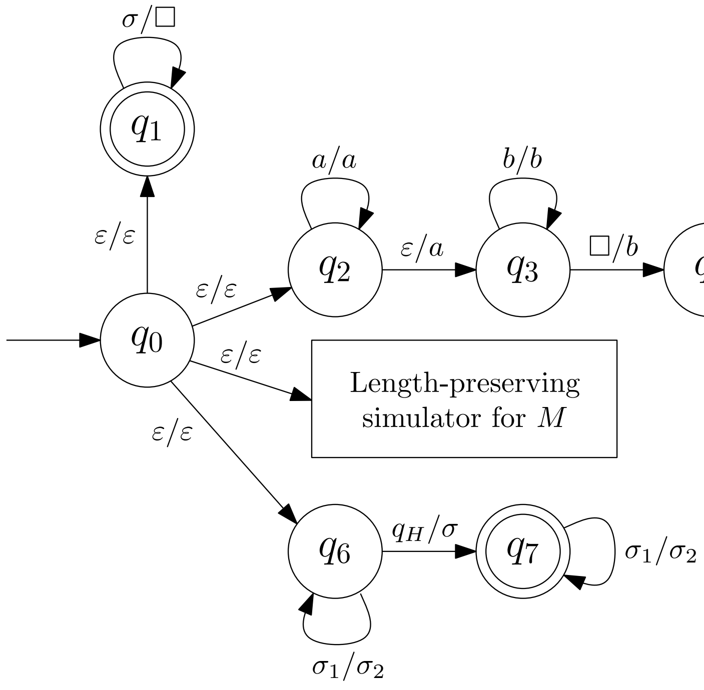


Figure 4: State diagram for the transducer T used in the proof of Lemma 12.

Again, a and b are chosen to be distinct from all other characters, and q_H is the halting state in M. The symbol $\Box$ is the "blank tape cell" symbol for M. The length-preserving simulator for M is the transducer from Lemma 6.

It is easily seen that every accepting computation in T is length-preserving. To show that τ^* is rational if and only if M halts on input ε, we consider the same cases as before:

M **halts on input** ε**:** Let k be the least natural number such that M halts on ε with the computation using at most $k-1$ tape cells.

Pick $x \in \Sigma^n$, where $n \geq k$. Then $x\tau\square^n$ using the transitions in q_0 and q_1. By Lemma 6, we know that $\square^n\tau^*z$ for some $z \in \Sigma^n$ containing the symbol q_H. By the computation in states q_0, q_6, and q_7, $z\tau y$ for every $y \in \Sigma^n$. So, for each $n \geq k$ and for each $x, y \in \Sigma^n$, we have that $x\tau^*y$.

Let $\rho = \{(x,y) \in \tau^* \mid \text{len}(x) = \text{len}(y) < k\}$. Since ρ is finite, it is rational. We can then write τ^* as

$$\tau^* = \rho \cup \{(x,y) \in \Sigma^* \times \Sigma^* \mid \text{len}(x) = \text{len}(y) \geq k\}$$

So, τ^* is the union of two rational transductions, so τ^* is rational.

M **does not halt on input** ε**:** This is nearly the same proof as in Lemma 9. There is no n such that $\square^n\tau^*$ contains a string with the character q_H (although the self-loop in q_6 is able to output q_H, it is impossible to do so and reach an accepting state unless the input string contains q_H). So, the strings containing the characters a and b are exactly the ones of the form $a^ib^i\square^{n-2i}$ for all natural numbers i and n such that $n \geq 2i$.

If τ^* is rational, then by closure properties, $\square^*\tau^* \cap \{a,b\}^*$ is regular. But this language is $\{a^nb^n \mid n \in \mathbb{N}\}$, so we have a contradiction, so τ^* must not be rational.

So, τ^* is rational if and only if M halts on input ε, so the reduction has been shown, so $L\left[\text{Rat}\left(\tau^*\right)\right]_{\mathbf{LP}}$ is RE-hard. ■

4.3 Alphabetic Transducers

This section explores the capabilities of alphabetic transducers. Because we no longer allow nondeterminism, the proof of Theorem 12 does not extend easily to this case.

Our first result about alphabetic transducers is that they can, in some sense, simulate Turing machines.

Lemma 13. $L\left[a^*\tau^* \subseteq (\Sigma - \{b\})^*\right]_{\mathbf{ALPHA}}$ *is* RE*-hard.*

Although this proof is subsumed by either Theorem 18 or Theorem 24, the construction strategy outlined here is useful for those proofs.

Proof idea. We reduce from K_0 by showing that the construction $\square^*\tau^*$ can be used to simulate an arbitrary Turing machine.

Fundamentally, with (deterministic) alphabetic transducers, information cannot flow to the left, so there is no obvious way to simulate a left move by a Turing machine. This can be overcome by having the entire tape contents move to the right at each step. With this approach, the Turing machine head never needs to move left on the transducer's tape. The idea of the construction is to keep a constant-size queue of upcoming letters and rearrange the letters as necessary. Given these details, the construction is not difficult, but I will go through the details for completeness and because the construction will be modified in the proof of Theorem 16. ■

Proof. Let M be an arbitrary Turing machine. We let Γ be the alphabet of M (which must include the blank symbol $\square$), and we let S be the state set of M. s_0 denotes the start state and s_H denotes the halt state of M.

We define a transducer T as follows. The alphabet of T is $\Sigma = \Gamma \cup S \cup \{\blacksquare\}$. The symbol $\blacksquare$ is used to indicate the portion of the tape to the left of the usable region. The state set of T is

$$Q = \{q_0, q_1, q_2, q_3\} \cup (\{q_4\} \times \Gamma) \cup (\{q_5\} \times \Gamma \times (\Gamma \cup S)) \cup (\{q_6\} \times (\Gamma \cup S) \times (\Gamma \cup S))$$

The transition function for T is defined by Figure 5. In every transition in the diagram, σ is chosen to be an arbitrary element of Γ, s is chosen arbitrarily from S, and γ is chosen arbitrarily from $\Gamma \cup S$, with all combinations of choices included as transitions. Any unspecified transitions will not be reached in the construction, and may be chosen arbitrarily. It can be easily verified that the machine is deterministic, and meets all necessary properties of an alphabetic transducer.

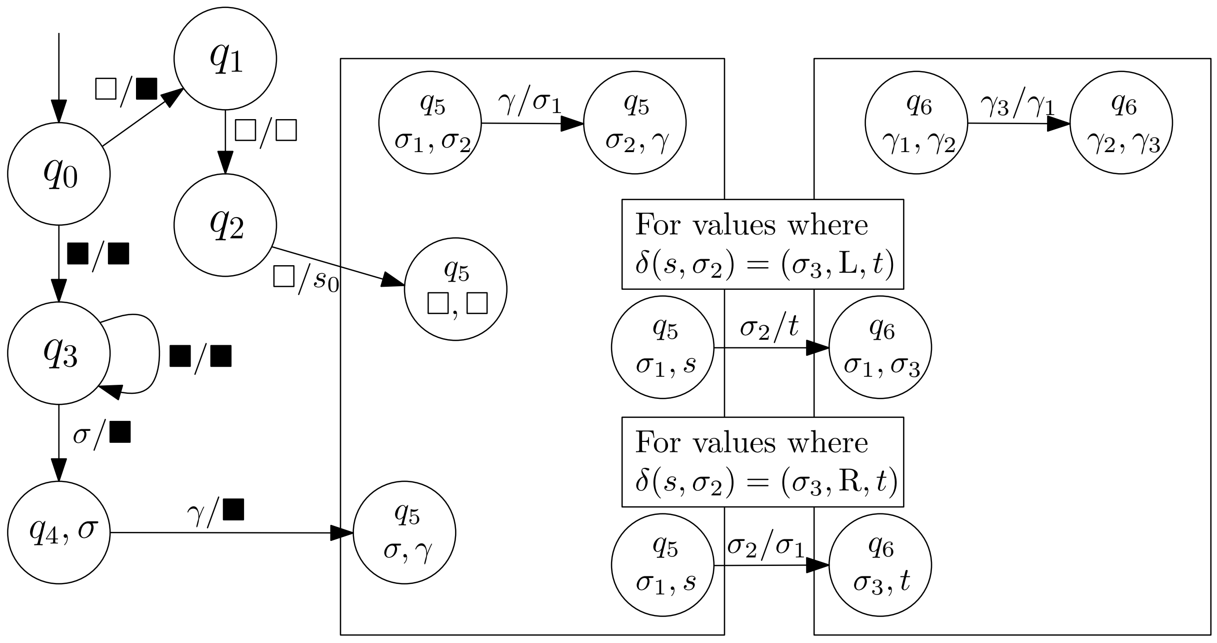


Figure 5: Transitions in the alphabetic transducer used in the proof of Lemma 13.

I will now show that M halts on ε if and only if $\square^*\tau^* \subseteq (\Sigma - \{s_H\})^*$.

- The transitions between states q_0, q_1, q_2, and $(q_5, \square, \square)$ are a special case for the first iteration, and introduce the start state s_0. They ensure that for any $n \geq 3$, the string $\square^n$ maps to the string $\blacksquare\square s_0\square^{n-3}$ under τ. We need to insert the character $\blacksquare$ so that the normal path is taken in all subsequent runs, and we need the character $\square$ because the rest of the transducer cannot handle a state as the first symbol after the $\blacksquare$ characters.
- In the common case, states q_0 and q_3 copy all $\blacksquare$ symbols at the start of the tape. States q_3 and q_4 read the first two symbols from the tape.

- States of the form q_5 copy characters in the obvious way until they reach the Turing machine state. The transitions from q_5 states to q_6 states rearrange and replace the upcoming characters appropriately, depending on whether the Turing machine head should move left or right. State q_6 simply copies all remaining characters.

So, if M, when run on ε, halts at tape position m after n time steps, then $\square^{2n+m+3}\tau^*$ will contain a string with the symbol s_H. If M does not halt on ε, then the symbol s_H will never appear, so it must be the case that M halts on ε if and only if $\square^*\tau^* \subseteq (\Sigma - \{s_H\})^*$. So, an appropriate many-one reduction has been shown, so the proof is complete. ■

Example 14. If M is the Turing machine from Example 2, then the following trace describes the computation of M on ε provided by Lemma 13. Note that the states of M are denoted q_i and not s_i.

□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
■	□	q_0	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
■	■	■	□	0	q_1	□	□	□	□	□	□	□	□	□	□	□	□
■	■	■	■	■	□	0	1	q_2	□	□	□	□	□	□	□	□	□
■	■	■	■	■	■	■	□	0	q_1	1	0	□	□	□	□	□	□
■	■	■	■	■	■	■	■	■	□	0	1	q_2	0	□	□	□	□
■	■	■	■	■	■	■	■	■	■	■	□	0	q_H	1	□	□	□

Lemma 13 generalizes as follows:

Corollary 15. *$L\left[R\tau^* \subseteq \Gamma^*\right]_{\mathbf{ALPHA}}$ is* RE*-hard.*

We can extend the simulation ideas from Lemma 13 to the following result:

Theorem 16. *$L\left[\mathrm{Reg}\left(a^*\tau^*\right)\right]_{\mathbf{ALPHA}}$ is* RE*-hard.*

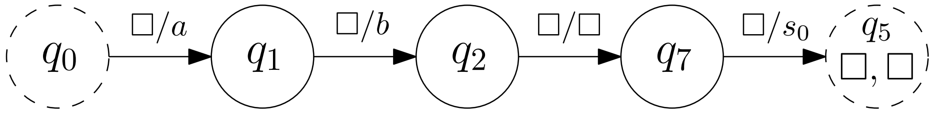
Proof idea. We modify the construction in Lemma 13. Instead of intermediate strings starting with $■^n$ for some n, they start with a^nb^n for some n. If the halt state is ever reached, we stop all computation. This ensures that if M halts, $\square^*\tau^*$ is regular, and if M does not halt, $\square^*\tau^*$ essentially contains $\{a^nb^n \mid n \in \mathbb{N}\}$, and is thus not regular. ■

Proof. Given a Turing machine M, we create a transducer T with transduction τ such that $\square^*\tau^*$ is regular if and only if M halts on input ε.

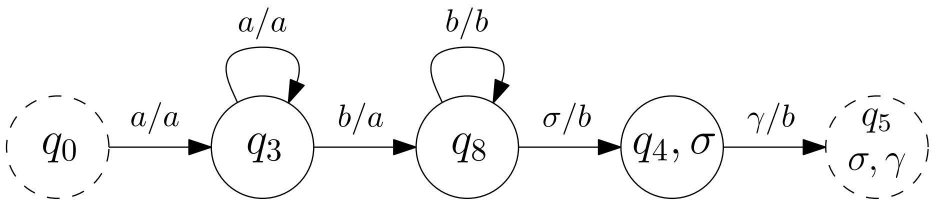
Because the transducer is so similar to the one given in Lemma 13, we only describe the differences.

The alphabet for T is $\Gamma \cup S \cup \{a, b\}$, where a and b are letters not contained in Γ or S.

We replace the transitions among q_0, q_1 and q_2 by the following construction:



The transitions among q_0, q_3 and states of the form q_4 are replaced by the following:



In addition, every state, upon reading the character s_H, writes s_H and transitions to a new state q_9 (even if this transition replaces a previous transition). State q_9 performs the identity mapping and has a self-loop at every input character.

I now claim that if $\tau = \mathcal{T}(T)$, then $\square^*\tau^*$ is regular if and only if M halts on input ε.

There are now two cases to consider:

M **halts on** ε**:** Because alphabetic transducers are sequential, we can analyze the set of infinite strings $\square^\omega\tau^*$ and use the fact that $\square^*\tau^*$ is the set of all finite prefixes of strings in $\square^\omega\tau^*$. Because the halt state s_H is eventually reached, we know that $\square^\omega\tau^*$ must reach a fixed point at some string of the form $a^n s_H x$ for some $x \in \Sigma^\omega$, so $\square^\omega\tau^*$ is finite. Since every element of $\square^\omega\tau^*$ consists of a finite string followed by infinitely many $\square$ characters, there is some integer n such that every infinite string in $\square^\omega\tau^*$ contains only the letter $\square$ after the first n letters. So, we can let

$$A = \{x \mid \text{len}(x) \leq n \text{ and } x \text{ is a prefix of } y \text{ for some } y \in \square^\omega\tau^* \}$$

and

$$B = \{x \mid \text{len}(x) = n+1 \text{ and } x \text{ is a prefix of } y \text{ for some } y \in \square^\omega\tau^* \}$$

Since A and B are both finite, they are regular. We then observe that $\square^*\tau^* = A \cup B\square^*$ to see that $\square^*\tau^*$ is regular.

M **does not halt on** ε**:** I claim that $\square^*\tau^*$ is not regular. Suppose that it is regular. Then $\square^*\tau^* \cap a^*b^*$ must also be regular. Because a and b are not contained in $\Gamma \cup S$, we know that they only appear at the start of any string in $\square^*\tau^*$. Also, there are always at least as many instances of a as instances of b, and if $n \geq m$, then $a^n b^m$ will eventually be reached from $\square^{m+n}\tau^*$. So, $\square^*\tau^* \cap a^*b^* = \{a^n b^m \mid n \geq m\}$. This language is not regular, so we have a contradiction.

Since we know that $\square^*\tau^*$ is regular if and only if M halts on ε. So, an appropriate many-one reduction has been shown, so the proof is complete. ■

To fit this result into our list of known problems, we have the following corollary:

Corollary 17. *$L\left[\text{Reg}\left(R\tau^*\right)\right]_{\textbf{ALPHA}}$ is* RE*-hard.*

4.4 Simulation with Iterated Reset Transducers

First, notice that reset transducers can be expressed in terms of a *replacement function.* That is, if T is a reset transducer over an alphabet Σ, then there is some function $\rho : (\Sigma \cup \{\text{start}\}) \times \Sigma \to \Sigma$ such that one iteration of T is equivalent to taking every adjacent pair σ_1, σ_2 in the input string, and replacing σ_2 by $\rho(\sigma_1, \sigma_2)$. This ρ exists because the state of T depends exactly on the most recent character read, so exactly two characters of information are available to make the decision.

The following theorem shows that, essentially, reset transducers are capable of simulating Turing machines.

Theorem 18. *$L\,[a^*\tau^* \subseteq (\Sigma - \{b\})^*]_{\mathbf{RESET}}$ is* RE*-hard.*

Proof idea. As in previous proofs, we are given a Turing machine M. Our goal is to construct a transducer that simulates this machine, so that the language $\square^*\tau^*$ contains a string with the character q_H if and only if M halts. Since the number of states in the transducer is bounded by the alphabet size, we choose a large alphabet. We divide a single Turing machine step into four "phases", where the phase number is encoded into the alphabet. ■

Proof. Let $M = (Q, \Sigma, \delta, q_0, q_H)$ be a Turing machine. Let $\Sigma' = \Sigma \cup \{■\}$ (where $■ \notin \Sigma$). The purpose of the ■ symbol is to ensure that we only generate a tape head at the first step in our construction. We also extend δ to be defined when reading ■ as input (its values on these inputs are chosen arbitrarily, and do not matter for this construction).

We construct a transducer T with alphabet Γ, where Γ is defined as follows:

- $\Gamma_1 = Q \cup \Sigma'$
- $\Gamma_2 = (Q \times \Sigma') \cup \Sigma'$
- $\Gamma_3 = (\Sigma' \times Q \times \Sigma') \cup \Sigma'$
- $\Gamma_4 = (\Sigma' \times Q \times \Sigma') \cup \Sigma'$
- $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$, where all unions are disjoint unions.

The implementation details are specified in Figure 6. σ, σ', σ_1, σ_2, σ_{L}, and σ_{R} are assumed to be elements of Σ'. γ is assumed to be either an element of Σ' or an element of Q. q and q' are assumed to be elements of Q, and Δ is assumed to be an element of $\{\mathrm{L}, \mathrm{R}\}$. For each i, a superscript of i means that the character is taken from Γ_i.

It can be easily checked that each transition is defined at most once. Undefined transitions will not come up in this construction, so they can be defined arbitrarily.

The four phases have the following behavior:

1. Read the character under the tape head, and merge it into the Turing machine state. This requires us to shift the tape one cell to the right.
2. Read the character to the left of the tape head, and merge it into the Turing machine state. Replace the character under the tape head with the output character.

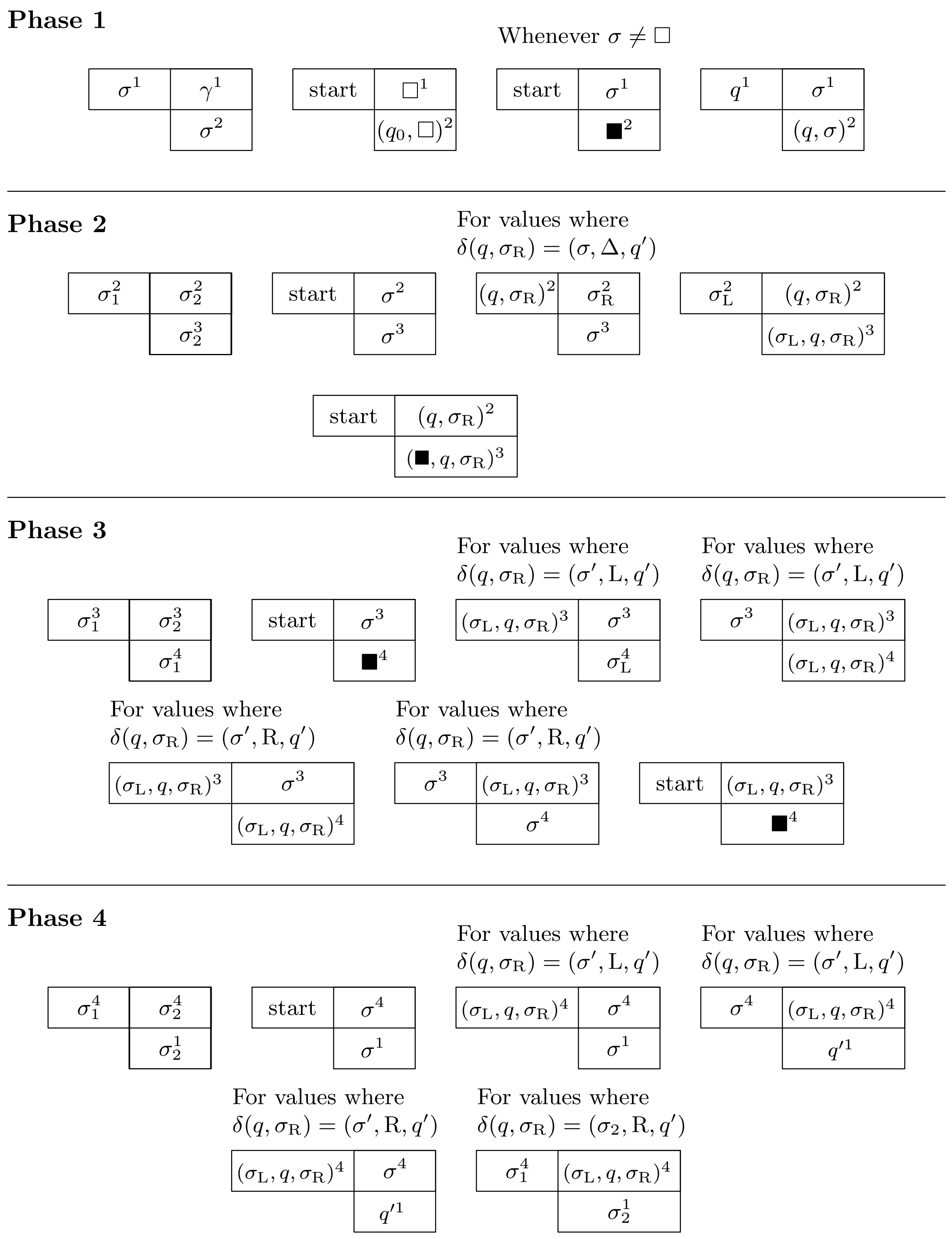

Phase 1
Whenever $\sigma \neq \square$
σ^1 γ^1 σ^2
start $\square^1$ $(q_0, \square)^2$
start σ^1 $\blacksquare^2$
q^1 σ^1 $(q, \sigma)^2$
Phase 2
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma, \Delta, q')$
σ_1^2 σ_2^2 σ_2^3
start σ^2 σ^3
$(q, \sigma_{\mathrm{R}})^2$ σ_{R}^2 σ^3
σ_{L}^2 $(q, \sigma_{\mathrm{R}})^2$ $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^3$
start $(q, \sigma_{\mathrm{R}})^2$ $(\blacksquare, q, \sigma_{\mathrm{R}})^3$
Phase 3
σ_1^3 σ_2^3 σ_1^4
start σ^3 $\blacksquare^4$
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{L}, q')$
$(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^3$ σ^3 σ_{L}^4
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{L}, q')$
σ^3 $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^3$ $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^4$
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{R}, q')$
$(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^3$ σ^3 $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^4$
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{R}, q')$
σ^3 $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^3$ σ^4
start $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^3$ $\blacksquare^4$
Phase 4
σ_1^4 σ_2^4 σ_2^1
start σ^4 σ^1
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{L}, q')$
$(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^4$ σ^4 σ^1
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{L}, q')$
σ^4 $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^4$ q'^1
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma', \mathrm{R}, q')$
$(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^4$ σ^4 q'^1
For values where $\delta(q, \sigma_{\mathrm{R}}) = (\sigma_2, \mathrm{R}, q')$
σ_1^4 $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})^4$ σ_2^1

Figure 6: Transitions in the reset transducer used in the proof of Theorem 18.

3. If the tape head should move left, swap it with the character to its left. This requires us to shift the tape one cell to the right.

4. If the tape head should move right, swap it with the character to its right. Transition to the new Turing machine state.

Given this behavior, it is clear that τ^4 is essentially the one-step relation for M, in the sense that $\square^*\tau^*$ contains the character q_H^1 if and only if M halts on the empty input. So, letting $a = \square$ and $b = q_H^1$, we have a many-one reduction from K_0 to the language $L\left[a^*\tau^* \subseteq (\Sigma - \{b\})^*\right]_{\textbf{RESET}}$, so the proof is complete. $\blacksquare$

Example 19. The following is a trace of the Turing machine M from Example 2 when run using the reset transducer from Theorem 18. By invariant, all characters in any intermediate string are in the same phase. The number at the left denotes the phase.

1 :	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
2 :	$(q_0, \square)$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
3 :	$(\blacksquare, q_0, \square)$	0	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
4 :	$\blacksquare$	$(\blacksquare, q_0, \square)$	0	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
1 :	$\blacksquare$	0	q_1	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
2 :	$\blacksquare$	$\blacksquare$	0	$(q_1, \square)$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
3 :	$\blacksquare$	$\blacksquare$	0	$(0, q_1, \square)$	1	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
4 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	$(0, q_1, \square)$	1	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
1 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	1	q_2	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
2 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	1	$(q_2, \square)$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
3 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	1	$(1, q_2, \square)$	0	$\square$	$\square$	$\square$	$\square$	$\square$
4 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	$(1, q_2, \square)$	1	0	$\square$	$\square$	$\square$	$\square$
1 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	q_1	1	0	$\square$	$\square$	$\square$	$\square$
2 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	$(q_1, 1)$	1	0	$\square$	$\square$	$\square$
3 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	$(0, q_1, 1)$	1	0	$\square$	$\square$	$\square$
4 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	$(0, q_1, 1)$	1	0	$\square$	$\square$
1 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	1	q_2	0	$\square$	$\square$
2 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	1	$(q_2, 0)$	0	$\square$
3 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	1	$(1, q_2, 0)$	$\square$	$\square$
4 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	$(1, q_2, 0)$	1	$\square$
1 :	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	0	q_H	1	$\square$

We can generalize Theorem 18 to the following corollary:

Corollary 20. *$L\left[R\tau^* \subseteq \Gamma^*\right]_{\textbf{RESET}}$ is* RE-*hard.*

4.5 Simulation with Iterated Reversible Transducers

The following sequence of constructions has the end effect of showing that binary toggle transducers are capable of simulating Turing machines. The proof is somewhat involved, and will be broken into four major parts, each of which solves a specific problem:

1. Since the one-step transformation must be a bijection, we must express the TM computation in a reversible way. To do this, we represent each TM tape cell as a list containing the entire history of that cell. During each run, we append one tape symbol to the end of each list. The idea of using computation history to enforce reversibility is a well-known technique; for further background, see [2].

2. To append to the end of a list using a reversible transducer, we keep a counter before each list element. Each time we move through the list, we increment the counter of every list element before the last. Upon seeing the number 0, we know that we have reached the end of the list.

3. When any computation runs out of space, we need to make sure that the tape symbols do not become corrupted in such a way that the halting state erroneously appears. To avoid this issue, we take advantage of reversibility and run the transducer in reverse until we reach the starting configuration.

4. The above steps are more easily expressed as a reversible transducer over a large alphabet, so we give such a presentation first. We then show that all information can be encoded in binary in such a way that the construction still works.

Ideally, the set of starting configurations would be intuitively "simple" in some way. In previous proofs, the starting string was taken from a regular language, but unfortunately, the construction in this section requires the language to be non-regular. Recall from Subsection 2.3 that a *constant block language* is a language B of the form

$$\bigcup_{n\in\mathbb{N}}((x^n y)^* z)^*$$

It is notable that the complement of any such language is context-free. Constant block languages will be used in the constructions below.

The first two steps in the above list are presented in the proof of the following lemma. In order for the construction to be correct, it must be the case that the provided Turing machine M halts on ε if and only if the halt state ever appears in the output of $B\tau^*$. The following lemma ensures that the left-to-right implication holds:

Lemma 21. *For every Turing machine M, there is a constant block language B and a reversible transducer T with transduction τ such that if M halts on input ε, then $B\tau^*$ contains a string with the symbol q_H (the halt state of M).*

Note that as a self-contained statement, Lemma 21 is trivial. We present it because the construction given in the proof will be extended into a meaningful theorem.

Proof idea. We assume that the tape is structured as some collection of "major blocks", each of which represents the history of one tape cell in a list of "minor blocks". The idea is illustrated in the following table, although a number of details are missing. The symbols in bold are the "current" ones being computed.

$\boldsymbol{q_0}$	$\sqcup$	$\sqcup$	$\sqcup$	$\boldsymbol{a}$	$\sqcup$	$\sqcup$	$\sqcup$	$\boldsymbol{a}$	$\sqcup$	$\sqcup$	$\sqcup$	$\boldsymbol{a}$	$\sqcup$	$\sqcup$	$\sqcup$
q_0	$\boldsymbol{b}$	$\sqcup$	$\sqcup$	a	$\boldsymbol{q_1}$	$\sqcup$	$\sqcup$	a	$\boldsymbol{a}$	$\sqcup$	$\sqcup$	a	$\boldsymbol{a}$	$\sqcup$	$\sqcup$
q_0	b	$\boldsymbol{q_2}$	$\sqcup$	a	q_1	$\boldsymbol{b}$	$\sqcup$	a	a	$\boldsymbol{c}$	$\sqcup$	a	a	$\boldsymbol{a}$	$\sqcup$
q_0	b	q_2	$\boldsymbol{a}$	a	q_1	b	$\boldsymbol{q_3}$	a	a	c	$\boldsymbol{c}$	a	a	a	$\boldsymbol{a}$

To simplify the transitions, and to account for the fact that T must be alphabetic, we have T simulate the reset transducer in Theorem 18 rather than simulating a Turing machine directly.

Normally, a "write" operation is difficult to encode in a reversible transducer, since the transformation must be expressed as a permutation. By ensuring that we always write over the symbol $\square$, though, we can perform the write operation by setting $\square$ to map to the desired character.

Given these ideas, the main challenge is appending to the end of a list. Focusing on a single major block, we store a counter at the start of each minor block. At each iteration, T increments each counter until it finds the number 0, which denotes the end of the list. For example, the leftmost major block in the above example is repeated in more detail in the following computation:

1	q_0	0	$\sqcup$	0	$\sqcup$	0	$\sqcup$
2	q_0	1	b	0	$\sqcup$	0	$\sqcup$
3	q_0	2	b	1	q_2	0	$\sqcup$
4	q_0	3	b	2	q_2	1	a

In the actual construction, the counters are represented in reverse binary. We know from Example 1 that the increment function can be computed in reverse binary with a reversible transducer, so we extend that construction. ■

Proof. Let M be given. We choose $B = \bigcup_{n\in\mathbb{N}}((0^n\,\mathrm{E}\sqcup)^*\$)^*$. The symbol \$ acts as a delimiter between major blocks. Each minor block starts with some number of 0s, which represent the minor block's counter in reverse binary, with E denoting the end of the counter. The symbol E is not strictly necessary for this construction, but it will be extended to be more useful in the proof of Theorem 22. The symbol $\sqcup$ means that the minor block does yet not have a character; it is semantically different from $\square$, the blank symbol used for Turing machines.

Let T_0 be the reset transducer defined for M in Theorem 18, and let $\tau_0 = \mathcal{T}(T_0)$. Let Γ and $\rho : (\Gamma \cup \{\text{start}\}) \times \Gamma \to \Gamma$ be the alphabet and corresponding replacement function for T_0.

We construct a transducer T with alphabet $\Sigma = \{0, 1, \mathrm{E}, \sqcup, \$\} \cup \Gamma$. The state set Q is defined as $\{q_0, \ldots, q_6\} \times (\Gamma \cup \{\text{start}\}) \times \Gamma$.

The transition function T is defined by Figure 7, which defines the transitions for states of the form (q, m, m'), with m and m' fixed and $q \in \{q_0, \ldots, q_6\}$. The idea is that m denotes the value of the current cell and m' denotes the value of the previous cell. The solid circles define states in this part of the transducer, and the dashed circles define states elsewhere in the transducer. A state marked as i, m, m' is the state (q_i, m, m'). Any transitions that are not specified may be defined arbitrarily (within the limitations of a reversible transducer). Note that unlike in previous proofs, undefined transitions might be taken in this construction, but they do not affect the correctness of the proof. The start state is $(q_0, \square^1, \text{start})$.

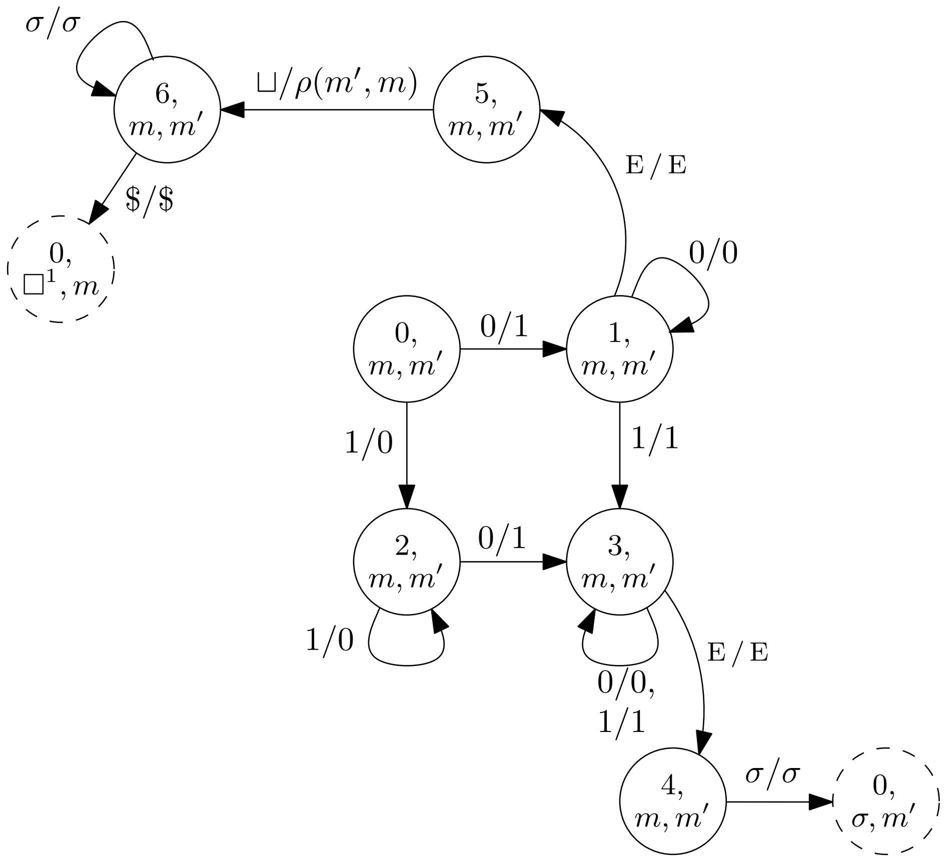


Figure 7: State diagram for one portion of the transducer T used in the proof of Lemma 21.

The transducer behaves as follows:

- States 0, 1, 2, and 3 are the increment transducer from Example 1, modified to remember whether the original binary string was 0 or nonzero. If we reach the character E, if we have seen a 1, then we transition to state 4, and if the string was 0, we transition to state 5.

  Note that we do not handle the case where E is reached before any 0 is seen.

- State 4 reads the symbol from Γ that is written on the tape, and stores it into the transducer's state. This ensures that when writing to the end of a major block, the previous letter is known.

- States 5 and 6 write the appropriate character to the end of the major block, then move to the start of the next major block. The transition from state 6 also stores the "current" tape value to be the "previous" tape value.

Given this description, it is clear that if M simulates the reset transducer defined by ρ, in the sense that if the string $((0^k \,\mathrm{E} \sqcup)^n \$)^\ell$ is given as input to the transducer and $2^k \geq n$, then the first n iterations of $\mathcal{T}(T)$ correspond to the first n iterations of τ_0 on the input $(\square^1)^k$.

So, if the Turing machine M halts on input ε, then the symbol q_H will appear in some string in $B\tau^*$, which is what needed to be shown. ■

The next result extends the proof of Lemma 21 so that the converse of Lemma 21 holds.

Theorem 22. *$L\left[B\tau^* \subseteq (\Sigma - \{a\})^*\right]_{\mathbf{REV}}$ is RE-hard.*

Proof idea. We extend the construction from Lemma 21 so that the character q_H never mistakenly appears in the output. This ensures that q_H eventually appears if and only if M halts on ε.

The challenge in avoiding errors is that, because reversible transducers recognize bijections, the computation for each string must eventually reach the starting string, so any simple attempts to "crash" will inevitably result in unpredictable behavior. In other words, we must be able to track the behavior of the transducer from the point an error occurs to the point that it reaches its starting string.

To overcome this challenge, we take advantage of the reversible property of the transducer. When an error occurs, the transducer is run in reverse until the string gets back to the start string. In particular, if each minor block has a counter with n bits, then the long-term behavior of the transducer is to alternate between running the first 2^n computation steps forward and running the first 2^n computation steps in reverse.

Because of our choice of language, we know that all counters use the same number of bits, so the reversal construction will only happen for the first counter.

To handle the case where a major block runs out of minor blocks, we simply do nothing for the remainder of the run. Eventually, the first counter will overflow and unwind the state, so we will never reach a corrupted state. ■

Proof. We continue the proof of Lemma 21. That is, we let M be an arbitrary Turing machine, and we let T_0, τ_0, Γ, and ρ correspond to the reset transducer for M provided by Theorem 18 (in the same way as in Lemma 21).

Define our input language as $B = \bigcup_{n\in\mathbb{N}}((0^n\, \mathrm{E}_{\mathrm{OK}} \sqcup)^*\$)^*$

We construct a transducer T with alphabet $\Sigma = \{0, 1, \mathrm{E}_{\mathrm{OK}}, \mathrm{E}_{\mathrm{ERR}}, \sqcup, \$\} \cup \Gamma$. The state set Q is defined as $\{q_{-1}\} \cup (\{q_0, \ldots, q_6\} \times \{\mathrm{OK}, \mathrm{ERR}\} \times (\Gamma \cup \{\mathrm{start}\}) \times \Gamma)$.

The transition function for T is defined by Figure 8. The notation and details of the diagram are similar to those of Figure 7, with a few differences. The transitions are complete in the sense that missing transitions do not come up in the construction, so they may be assigned arbitrarily. There is a single q_{-1} state, while the other six solid-bordered states are repeated for each element of $\{\mathrm{OK}, \mathrm{ERR}\} \times (\Gamma \cup \{\mathrm{start}\}) \times \Gamma$. The vertical dashed line divides the OK states and the ERR states. In all ERR states, the inputs and outputs are reversed, and all transitions form OK states to ERR states have corresponding transitions from ERR states to OK states. The start state for T is $(0, \mathrm{OK}, \square^1, \mathrm{start})$.

Since the transducer T extends the transducer from the proof of Lemma 21, the basic behavior is the same. The transducer behaves in the following way:

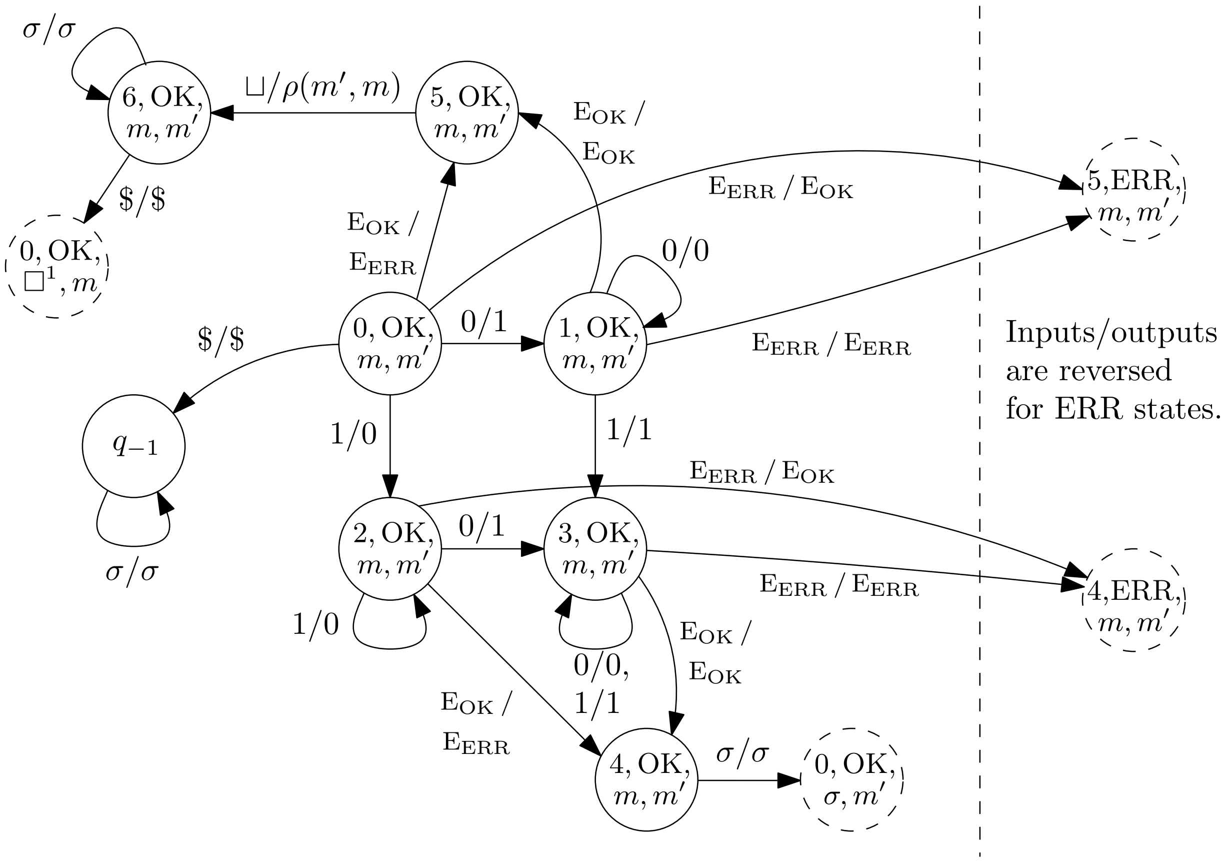


Figure 8: State diagram for one portion of the transducer T used in the proof of Theorem 22.

- If the transducer is in the start state $(q_0, \mathrm{OK}, \square^1, \mathrm{start})$ and reads the string $0^n\,\mathrm{E}_{\mathrm{OK}}$, then the first 2^n iterations will run the remainder of the computation forwards and result in the string $0^n\,\mathrm{E}_{\mathrm{ERR}}$. At this point, the next 2^n iterations will run the remainder of the string backwards (back to the original string) and result in the string $0^n\,\mathrm{E}_{\mathrm{OK}}$.

- Similar to the previous case, if the transducer is in any state of the form $(q_0, \mathrm{ERR}, m, m')$ or $(q_2, \mathrm{ERR}, m, m')$ and reaches the symbol E_{OK}, then it means that we have underflowed a counter while decrementing it. If we are in the process of running the transducer in reverse (that is, if we are in a state containing ERR) and we reach the symbol $\mathrm{E}_{\mathrm{ERR}}$, then we transition to an OK state, since we are running the reverse transducer in reverse, i.e. running the transducer normally.

- If the transducer is in a state containing q_0 and reads the character \$, then this indicates that all minor blocks have been used in a major block. In this case, T transitions to state q_{-1} and does nothing for the remainder of the computation. This ensures that the halt symbol q_H^1 will not be output mistakenly due to this error. Note that because the transition from q_{-1} is the identity, it does not need to have a separate version for OK and ERR.

To prove the correctness of T, all that needs to be shown is that if M does not halt on input ε, then

$((0^* \, \mathrm{E_{OK}} \sqcup)^* \$)^* \tau^*$ does not contain a string with the symbol q_H^1, since the proof of the converse extends trivially from the proof given of Lemma 21. We consider the iteration of T given a string $x \in ((0^n \, \mathrm{E} \sqcup)^* \$)^*$ as input, where n is fixed. It can be easily checked that the only character positions that might contain q_H^1 are the ones that start as $\sqcup$. From the above observations, and the work in the proof of Lemma 21, we know by invariant that any time a symbol changes, it either changes from $\sqcup$ to a symbol that appears in the computation of M on input ε, or it changes from such a symbol to $\sqcup$. So, q_H^1 will never appear, since M does not halt on ε.

So, we have that $B\tau^*$ contains a string with the character q_H^1 if and only if M halts on ε, so an appropriate reduction has been shown, so the proof is complete. ■

We can generalize Theorem 22 to the following:

Corollary 23. *$L\,[B\tau^* \subseteq \Gamma^*]_{\mathbf{REV}}$ is* RE*-hard.*

Transforming the construction in Theorem 22 to use a binary alphabet requires some work, but is mostly straightforward. The details are presented in the proof of the following theorem:

Theorem 24. *$L\,[B\tau^* \subseteq R]_{\mathbf{BTT}}$ is* RE*-hard.*

Proof idea. We encode every symbol from Γ in binary, and let S be the set of strings not containing the encoding of the halt state q_H^1 as a substring.

Encoding an arbitrary reversible transducer in binary is challenging because there may not be a simple way to perform a bijective transformation from one binary string to another using a binary toggle transducer. However, with the transducer provided for Theorem 22 and the right encoding, all operations are read-only, write-only, or change a single bit of information, so the problem does not occur here.

To ensure that the halt state q_H^1 is never reached accidentally, we ensure that the encoding q_H^1 is the only possible way to obtain the substring 1111. ■

Proof. All that is necessary for this proof is to convert the transducer from Theorem 22 to binary. Define M, T_0, τ_0, Γ, and ρ as in Theorem 22. We will define a new binary toggle transducer T based on the following encodings:

- 0 is encoded as 000
- 1 is encoded as 001
- $\mathrm{E_{OK}}$ is encoded as 010
- $\mathrm{E_{ERR}}$ is encoded as 011
- \$ is encoded as 100
- Choose some $k \in \mathbb{N}$ and some injection $\varphi : \Gamma \cup \{\sqcup\} \to \{0,1\}^k$ such that
  - $\varphi(q_H^1)$ contains 1111 as a substring.
  - q_H^1 is the only character with this property.

It is trivial that some valid pair (k, φ) exists. Let $\theta : \Gamma \cup \{\sqcup\} \to \{0,1\}^{k+2}$ be defined so that $\theta(\gamma) = 0\varphi(\gamma)0$. We let $\theta(\gamma)$ denote the encoding of γ.

For this construction, let $B = \bigcup_{n\in\mathbb{N}}(((000)^n 010\theta(\sqcup))^* 100)^*$. The changes to the transducer are described in the following list. Throughout the description, we assume the notation used in Figure 8.

- OK states of the form q_0 and q_2 take the transformation described in Figure 9.

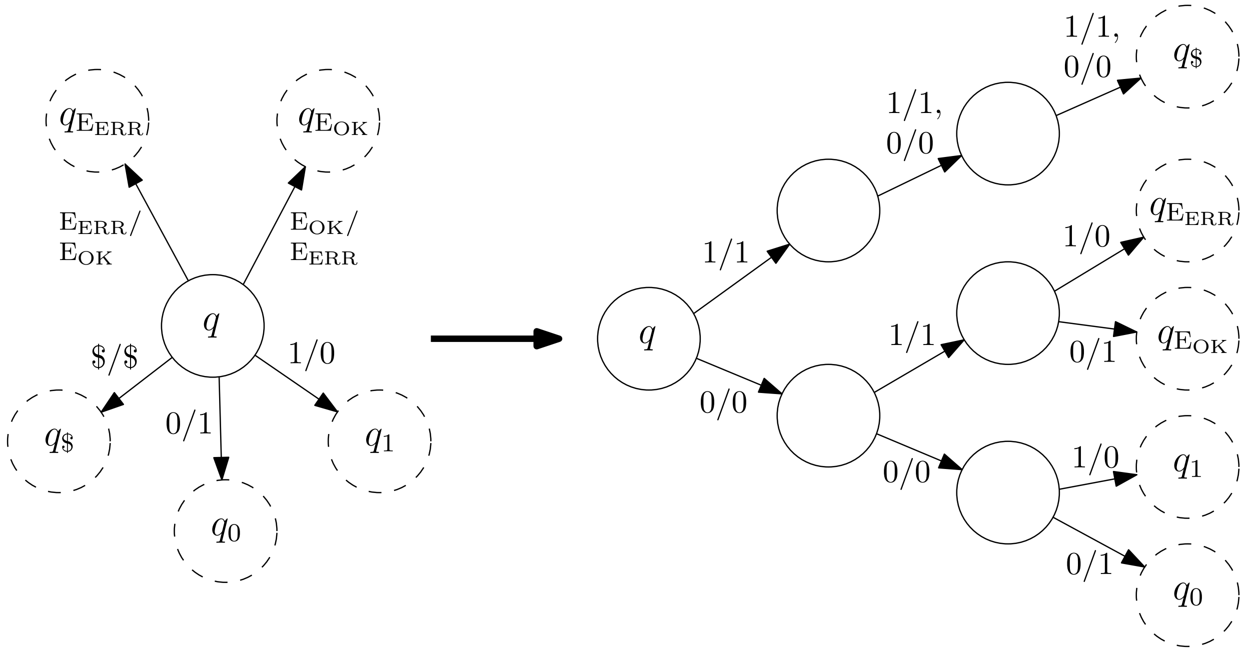


Figure 9: Encoding of states q_0 and q_2 used in Theorem 24.

- OK states of the form q_1 and q_3 perform the identity mapping for all inputs. So, we can trivially construct a tree such as the one in Figure 9, but where all transformations are of the form 0/0 or 1/1. This allows the transition system to move to the correct state.

- OK states of the form q_4 read $k+2$ symbols using a decision tree of height $k+2$, where all transitions do the identity mapping. The state reached in the decision tree determines the character σ to read.

- OK states of the form q_5 are transformed to a sequence of $k+2$ transitions, where the transitions correspond to the unique bit mask required to map $\theta(\sqcup)$ to $\theta(\rho(m', m))$.

- OK states of the form q_6 need to only accept well-formed instances of \$. They can do this by repeatedly running the following algorithm:

  - Repeatedly read sets of 3 bits until 100, 010, or 011 is seen. If 100 is seen, stop and transition to state $(q_0, \mathrm{OK}, \square^1, m)$. If 010 or 011 is seen, continue to the next step.

  - Read $k+2$ bits and ignore them.

- State q_{-1} simply has a self-loop for both inputs and performs the identity mapping.
- As before, all ERR states are identical to the corresponding OK states, but with all transitions reversed.

The start state of T remains the same.

It can be easily checked that the transducer T described here has the same behavior as the one in Theorem 22.

Let $R = \Sigma^* - \Sigma^* 1111 \Sigma^*$. That is, R is the set of all strings not containing the string 1111 as a substring. Notice that the string 1111 cannot occur anywhere in a valid configuration unless that configuration contains a well-formed encoding of q_H^1, and that 1111 must occur whenever a configuration contains the encoding of q_H^1. Therefore, we know that $B\tau^* \subseteq R$ if and only if a well-formed encoding of q_H^1 never appears in any string in $B\tau^*$. So, by the work in Theorem 22, we know that $L\left[B\tau^* \subseteq R\right]_{\mathbf{BTT}}$ is RE-hard. ■

5 Conclusions and Future Work

At first glance, it may seem that both reset transducers and reversible transducers are far too weak to perform a Turing machine simulation. So, the results of this paper are somewhat surprising. Although only specialized decision problems were shown undecidable, they are not completely contrived; as explained in the preliminaries, the language $L\left[R\tau^* \subseteq \Gamma^*\right]$ can be used to express the claim that an error is never reached. Also, the general fact that Turing machine simulation is possible provides an indication that related meaningful problems are also undecidable.

One particular proof in this paper that could be improved is the proof contained in Lemma 21, Theorem 22, and Theorem 24. For this proof, we needed to define the concept of a constant block language in order for the details to work. While constant block languages are intuitively simple, a more meaningful result would be one with a regular language as the set of input configurations.

In addition, this investigation could be extended naturally in a number of ways. The grid at the start of Section 4 leaves a number of open problems. It seems likely that nearly all of these problems are undecidable, since many involve multiple quantifier alternations when expressed in the obvious way. In addition, different classes of transducers and different meaningful decision problems could be considered. An interesting open problem is to find a natural class of transducers that is weak enough that it cannot simulate Turing machines. However, as this paper indicates, it is likely that it would be difficult to use such a transducer to model a system in practice.

References

[1] Jürgen Albert and Karel Culik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, pages 1–16, 1987.

[2] C. H. Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17:525–532, 1973.

[3] Jean Berstel and Luc Boasson. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.

[4] Henning Bordihn, Henning Fernau, Markus Holzer, Vincenzo Manca, and Carlos Martín-Vide. Iterated sequential transducers as language generating devices. *Theoretical Computer Science*, pages 67–81, 2006.

[5] Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers, 2001.

[6] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[7] R. R. Grigorchuk, V. V. Nekrashevich, and V. I. Sushchanski. Automata, dynamical systems and groups. *Proc. Steklov Institute of Math.*, 231:128–203, 2000.

[8] W. M. L. Holcombe. *Algebraic Automata Theory.* Cambridge University Press, 1982.

[9] Vincenzo Manca, Carlos Martín-Vide, and Gheorghe Paun. Iterated gsm mappings: A collapsing hierarchy. Technical report, 1998.

[10] V. Nekrashevych. *Self-Similar Groups*, volume 117 of *Math. Surveys and Monographs.* AMS, 2005.

[11] J. Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.