Extended Abstract:
Decision Problems on Iterated Length-Preserving Transducers

Alan Pierce
Advisor: Klaus Sutner

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 [2]. 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 [3]. 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 [4], the definition of an iterated transducer is modified slightly so that it recognizes languages, with the class of all iterated transducers generating the class of recursively enumerable languages. Our approach is different: we keep the definitions 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.

This extended abstract is organized as follows. Section 2 provides necessary definitions and introduces the classes of transducers and the decision problems that are under consideration. Section 3 introduces simple results which provide a context and proof strategy for the more significant proofs. Section 4 contains outlines of the main results of the paper. Section 5 discusses conclusions and future work.

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.

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. Arbitrary elements of Σ are usually denoted σ. Arbitrary strings are usually denoted x, y, etc.

A *DFA* is a machine consisting of a finite state set, a single start state, and a set of final states. The exact semantics can be found in any introduction to automata theory. A language (that is, a set of strings) is *regular* if it is recognized by some DFA.

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 $(x, y) \in \tau$. If L is a language, then $L\tau$ refers to the language $\{y \in \Sigma^* \mid x\tau y \text{ for some } x \in L\}$.

If T is a transducer, then $\mathcal{T}(T)$ refers to the transduction τ consisting of all ordered pairs of strings accepted by T. 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, τ^* is the set of all ordered pairs $(x, y) \in \Sigma^* \times \Sigma^*$ such that there exists some finite sequence of strings $z_0, \ldots, z_n$, where $x = z_0$, $y = z_n$, and for each $i < n$, $z_i \tau z_{i+1}$.

Turing machines, usually denoted M, will be used in the proofs below. We assume that a Turing machine is deterministic and has a single start state and a single halting state. The precise definition won't be necessary, although we make the following simplifying assumptions when they are convenient:

- Turing machines operate on one-way infinite tapes.
- The tape head must move either left or right at each step.
- Before halting, a Turing machine erases its tape and moves to the start of the tape.

Let $K_0 = \{M \mid M \text{ halts when given the input } \varepsilon\ \}$. It is well-known that K_0 is 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 $(x, y) \in \mathcal{T}(T)$, $|x| = |y|$. In particular, length-preserving transducers may be nondeterministic, and may have ε transitions, as long as all accepting computations are length-preserving.
- **ALPHA** refers to the set of alphabetic transducers. A transducer is *alphabetic* if for every transition $q\sigma_1 \to \sigma_2 s$, neither σ_1 nor σ_2 is ε, and such that the transition relation is 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.

  It can be seen that reset transducers are equivalent 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.

- **REVERSIBLE** 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 $q\sigma \to \pi_q(\sigma)s$. In other words, instead of freely writing a character based on the input, the transducer must decide ahead of time a permutation for the character transformation, but may transition to different states based on which character was read/written. We can invert the transduction generated by swapping the input and output symbols for each transition. This implies that reversible transducers always compute bijections.

- **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.

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. The decision problems considered are described as $L\,[p]$, where p is some predicate. The existence of certain variables implies that certain values appear as inputs to the decision problem

C refers to a class of transducers. For each instance of τ (or τ_1 or τ_2), a transducer T is included in the input, 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 in the input, and it is assumed that $\Gamma \subseteq \Sigma$. For each instance of R (or S), a DFA D is included in the input, and it is assumed that $\mathcal{L}(D) = R$. In addition, $\mathrm{Rat}\,(()\,\tau)$ means that the transduction τ is rational, and $\mathrm{Reg}\,(()\,L)$ means that the language L is regular.

For example, $L\,[x\tau^*y]_C = \{\langle T \in C, x \in \Sigma^*, y \in \Sigma^* \rangle \mid x\mathcal{T}(T)^*y\}$. This is simply the reachability problem between strings.

Another example is $L\,[\mathrm{Rat}\,(\tau^*)]_C = \{\langle T \in C \rangle \mid \mathcal{T}(T)^* \text{ is rational } \}$. 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.

A final example is $L\,[R\tau^* \subseteq \Gamma^*]_C = \{\langle \text{DFA } D, T \in C, \Gamma \subseteq \Sigma \rangle \mid \mathcal{L}(D)\mathcal{T}(T)^* \subseteq \Gamma^*\}$. 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.

The other decision problems considered all have motivations similar to the ones above.

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.

Lemma 1. *For each fixed Turing machine M, the one-step relation on M (mapping each tape configuration to its configuration after one computation step) is rational.*

Proof idea. We construct a transducer recognizing the one-step relation on properly-encoded Turing machine configurations. We assume that the input string contains the tape contents with a single tape head (represented by the state) somewhere in the string. By having ε outputs, the transducer reads a constant number of characters ahead before writing, so that all necessary context is encoded in the state control. In the normal case, simply copy the tape symbol, and in the case of the tape head or the surrounding characters, write the correct symbol to the tape, and write the tape head in the correct place, using the Turing machine logic encoded into the transducer. If the tape head moves beyond the bounds of the string, grow the tape, and if the tape contains a $\square$ symbol at the end, shorten the string so that it does not. $\square$

Lemma 2. *$L\left[x\tau^*y\right]_{\mathbf{TRANS}}$ is undecidable.*

Proof idea. I will show that if $L\left[x\tau^*y\right]_{\mathbf{TRANS}}$ was decidable, then K_0 would be decidable. Let M be a Turing machine, and let τ be the transducer provided by Lemma 1 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 containing q_0 as its only character, 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, if $L\left[x\tau^*y\right]_{\mathbf{TRANS}}$ was decidable, we could use it to determine if $x\tau^*y$, which would determine if $M \in K_0$, so we would have a decision algorithm for K_0. $\square$

Lemma 3. *$L\left[x\tau^*y\right]_{\mathbf{LP}}$ 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 $|\Sigma|^{|x|}$, so the algorithm must always halt. We can then simply check membership in the cycle. $\square$

An easy corollary of Lemma 3 is that for any of the transducer classes C other than **TRANS** listed above, $L\left[x\tau^*y\right]_C$ is decidable.

4 Undecidability Results

The results of the paper are summarized in the following table. A cell marked with U means that the problem is undecidable, and a cell marked with D means that the problem is decidable. It is generally the case (although not strictly true) that an undecidability result implies that all problems to the left in the table are undecidable, and a decidability result implies that all problems to the right in the table are decidable.

	TRANS	**LP**	**ALPHA**	**RESET**	**REVERSIBLE**	**BTT**
$L\left[x\tau^* y\right]$	U	D	D	D	D	D
$L\left[\mathrm{Rat}\left(\tau^*\right)\right]$	U	U				
$L\left[\mathrm{Rat}\left(\tau^* \cap \left(\Gamma^* \times \Gamma^*\right)\right)\right]$	U	U				
$L\left[\tau_1^* = \tau_2\right]$	U					
$L\left[\tau_1^* = \tau_2^*\right]$	U					
$L\left[\mathrm{Reg}\left(R\tau^*\right)\right]$	U	U				
$L\left[\mathrm{Reg}\left(R\tau^* \cap \Gamma^*\right)\right]$	U	U				
$L\left[R\tau^* \cap S \neq \varnothing\right]$	U	U	U	U	U	U
$L\left[R\tau^* \subseteq \Gamma^*\right]$	U	U	U	U	U	D
$L\left[0^*\tau^* \cap S \neq \varnothing\right]$	U	U	U			
$L\left[0^*\tau^* \subseteq \Gamma^*\right]$	U	U	U			D

Because of the limited space in this extended abstract, only sketches of proofs will be given, and only the most significant results are presented here.

The following theorem shows that alphabetic transducers can simulate Turing machines, for some informal notion of simulation.

Theorem 4. $L\left[R\tau^* \subseteq \Gamma^*\right]_{\mathbf{ALPHA}}$ *is undecidable.*

Although this proof is subsumed by either Theorem 5 or Theorem 6, the construction strategy outlined here is useful for those proofs.

Proof idea. Fundamentally, with (deterministic) alphabetic transducers, information cannot flow right, 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 one cell to the right at each step. With this approach, the Turing machine head stays still when it wants to move left, and moves two spaces to the right when it wants to move right. The exact transition details are somewhat tedious, but straightforward.

Using this approach, it can be shown that it is undecidable to determine if $(q_0\Box^*)\tau^* \subseteq (\Sigma - \{q_H\})^*$. In other words, we feed every possible length of starting tape, and we know that if our Turing machine halts on ε, then some input string will eventually reach a string with the character q_H. Conversely, since the tape head will eventually conceptually run off the end of the string, if the character q_H is ever seen, then it must be due to a valid halting computation, so it must be the case that the Turing machine halts on ε. $\Box$

The next result is significant because it implies, informally, that reset transducers are powerful enough to simulate Turing machines.

Theorem 5. $L\left[R\tau^* \subseteq \Gamma^*\right]_{\mathbf{RESET}}$ *is undecidable.*

Proof idea. We use the fact that reset transducers are equivalent to one-way cellular automata. That is, the symbol at position i at timestep t is a function of the symbols at positions $i-1$ and i at timestep $t-1$.

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 $(q_0\Box^*)\tau^*$ contains a string with the character q_H. Since the number of states in the transducer is bounded by the alphabet size, we choose a large alphabet. In each alphabet letter, we keep an integer from 1 to 4 to denote the current "stage" in the computation cycle. A full cycle consists of 4 stages, and simulates one step of the Turing machine. Within each stage, our alphabet takes values from $\Sigma \cup Q \cup (Q \times \Sigma) \cup (\Sigma \times Q \times \Sigma)$. The 4 transitions are described below. List item i describes the transition from stage i to stage $i+1$.

1. When transitioning to stage 2, shift all tape symbols one character to the right. If the Turing machine is at state q and reads symbol b, then, when writing q, instead write the symbol (q, σ_{R}).
2. Perform the identity mapping on the entire configuration, except when reading (q, σ_{R}) and σ_{R}. In this case, replace the σ_{R} on the right with the appropriate tape symbol to write. In addition, when writing the tape head, we are able to read the character to the left of the tape head, so replace the tape head pair with the triple $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})$.
3. Shift all tape symbols over by 1. If the Turing machine transition is a movement to the left, swap the tape head symbol with the symbol to its left.
4. Run the identity mapping on the tape, and if the Turing machine transition is a movement to the right, then swap the tape head with the symbol to its right. Also, when writing the tape symbol, reduce it from $(\sigma_{\mathrm{L}}, q, \sigma_{\mathrm{R}})$ to q', where q' is the new state in the Turing machine state control.

The implementation details for each stage are reasonably straightforward and are not included. Using this construction, we can construct a reset transducer that simulates a given Turing machine, in the sense that q_H appears on some transducer input if and only if the Turing machine halts on input ε. $\square$

The next result demonstrates that binary toggle transducers (and, thus, reversible transducers) can simulate Turing machines.

Theorem 6. $L\left[R\tau^* \cap S \neq \varnothing\right]_{\mathbf{BTT}}$ *is undecidable.*

A full version of this proof is especially difficult compared to the other proofs in this paper, so I will only provide the high-level idea.

Proof idea. The class $L\left[R\tau^* \cap S \neq \varnothing\right]$ is necessary, since claims about an alphabet $\Gamma \subseteq \Sigma$ often become trivial when $\Sigma = \{0, 1\}$. We let S be all strings containing x as a substring, for some $x \in \Sigma^*$. Letting x be the encoded version of the Turing machine's halting state, a decider for $L\left[R\tau^* \cap S \neq \varnothing\right]$ allows us to ask whether an iterated binary toggle transducer can ever conceptually reach the halting state.

For simplicity, I will only show that reversible transducers can simulate Turing machines. Converting the construction to work in binary is simply a matter of choosing the appropriate encoding.

A number of challenges arise when trying to show that a reversible transducer can simulate a Turing machine. Because the corresponding transduction must be a bijection, there is no concept of "overwriting" a value; we can only write a particular character if we know its previous tape value. Also, the Turing machine simulation must be done in an inherently-reversible way. A number of challenges arise:

Making the transition reversible: In order to make the transition reversible, we keep the entire history of each tape cell in a region called a "major block". The tape simply consists of some finite list of major blocks. Each major block consists of a finite list of minor blocks, where a minor block is used to denote the state of that major block at a particular point in time.

With the setup of major and minor blocks, the transition system becomes relatively easy: in order to make a transition, we scan the major blocks left-to-right, and for each one, we read the last minor block, write one more minor block. When writing to a minor block, we know by assumption that all symbols being written were the character $\square$, so it is possible to set up a permutation ahead of time so that the correct symbols are written. Assuming we can safely read and write blocks in this way, we are able to use a construction such as the one in Theorem 4. So, the problem reduces to figuring out the details of appending to a list of minor blocks, and handling the case where we run out of space.

Appending to each the end of each major block: For this part of the construction, we can focus on a single major block; the same algorithm is used independently for each major block. Although appending to the end of a list is conceptually an invertible operation, there is no trivial way to determine when the minor block being read is the last one; we must know this in order to know when the output permutation should be the identity and when it should write meaningful information. It is easy to show that the reverse-binary increment operation can be computed by a reversible transducer, so we can put a counter at the start of each minor block, and increment each counter before we reach the end. We need an END marker to determine when the binary string ends, since it must be variable size. As an example, this is what a major block would look like if two minor blocks were used and two remained unused:

0	1	0	END	σ_0	1	0	0	END	σ_1	0	0	0	END	$\square$	0	0	0	END	$\square$

It is simple to check during each increment if the value was zero or not, so we can tell when we are at the end of the list, so we know where to write the new data.

The set of input strings can be constructed to contain arbitrarily many major blocks, arbitrarily many minor blocks within each major block, and arbitrarily many bits for each counter. At this point, the construction has the property that if the Turing machine being simulated halts on ε, then there is some input to the transducer that causes the halt state q_H to appear in some string. It remains to handle the converse.

Handling errors: The challenge with ensuring that we never mistakenly reach the state q_H is that there is no way to simply crash as a result of an error. Because reversible transducers must generate permutations, the transducer must eventually loop back to its starting string. In order to ensure that we never compute on a corrupted tape, we can take advantage of the reversible property of the transducers.

The new rule can be added: if we are incrementing a counter for a minor block, and we overflow the entire counter, then we switch to a "reverse mode": that is, transition to a copy of the original transducer in which the inputs and outputs are switched. This is done by adding an ERR symbol that alternates with the END symbol. This construction ensures that, for example, if the first counter has n bits, and it is the smallest counter in the string, then then entire computation will be run 2^n times, then it will be run in reverse 2^n times to reach the start. Because running a transducer in reverse will never reach a corrupted state, we can be sure that errors will never cause us to see a false positive. It can be shown that this same approach nests recursively; regardless of the lengths of the different counters, we will never reach a corrupted state.

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

These techniques working together allow a reversible transducer to, in some sense, simulate a Turing machine, and thus, it is undecidable if a particular symbol (the halt state, in this case) is reachable.

□

5 Conclusion 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.

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

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

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