Single-Use Automata and Transducers for Infinite Alphabets

Mikołaj Bojańczyk
Institute of Informatics, University of Warsaw, Poland
bojan@mimuw.edu.pl

Rafał Stefański
Institute of Informatics, University of Warsaw, Poland
rafal.stefanski@mimuw.edu.pl

Abstract

Our starting point are register automata for data words, in the style of Kaminski and Francez. We study the effects of the single-use restriction, which says that a register is emptied immediately after being used. We show that under the single-use restriction, the theory of automata for data words becomes much more robust. The main results are: (a) five different machine models are equivalent as language acceptors, including one-way and two-way single-use register automata; (b) one can recover some of the algebraic theory of languages over finite alphabets, including a version of the Krohn-Rhodes Theorem; (c) there is also a robust theory of transducers, with four equivalent models, including two-way single use transducers and a variant of streaming string transducers for data words. These results are in contrast with automata for data words without the single-use restriction, where essentially all models are pairwise non-equivalent.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Automata, semigroups, data words, orbit-finite sets

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.113

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.10504`.

Funding Supported by the European Research Council under the European Unions Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080).

1 Introduction

One of the appealing features of regular languages for finite alphabets is the robustness of the notion: it can be characterised by many equivalent models of automata (one-way, two-way, deterministic, nondeterministic, alternating, etc.), regular expressions, finite semigroups, or monadic second-order logic. A similar robustness appears for transducers, see [11] for a survey; particularly for the class of regular string-to-string functions, which can be characterised using deterministic two-way transducers, streaming string transducers, or MSO transductions.

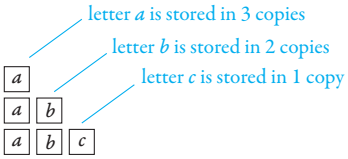
This robustness vanishes for infinite alphabets. We consider infinite alphabets that are constructed using an infinite set $\mathbb{A}$ of atoms, also called data values. Atoms can only be compared for equality. The literature for infinite alphabets is full of depressing diagrams like [15, Figure 1] or [6, p. 24], which describe countless models that satisfy only trivial relationships such as deterministic $\subseteq$ nondeterministic, one-way $\subseteq$ two-way, etc.

This lack of robustness has caused several authors to ask if there is a notion of "regular language" for infinite alphabets; see [3, p. 703] or [4, p. 2]. This question was probably rhetorical, with the assumed answer being "no". In this paper, we postulate a "yes" answer. The main theme is register automata, as introduced by Kaminski and Francez [13], but with

the single-use restriction, which says that immediately after a register is used, its value is destroyed. As we show in this paper, many automata constructions, which fail for unrestricted register automata, start to work again in the presence of the single-use restriction.

Before describing the results in the paper, we illustrate the single-use restriction.

▶ **Example 1.** Consider the language "there are at most three distinct letters in the input word, not counting repetitions", over alphabet $\mathbb{A}$. There is a natural register automaton which recognises this language: use three registers to store the distinct atoms that have been seen so far, and if a fourth atom comes up, then reject. This automaton, however, violates the single-use restriction, because each new input letter is compared to all the registers.



Here is a solution that respects the single-use restriction. The idea is that once the automaton has seen three distinct letters a, b, c, it stores them in six registers as explained in the picture on the right. Assume that a new input letter d is read. The behaviour of the automaton (when it already has three atoms in its registers) is explained in the flowchart in Figure 1.

A similar flowchart is used for the corner cases when the automaton has seen less than three letters so far.

Our first main result, Theorem 6 (in Section 3), says that the following models recognise the same languages over infinite alphabets:

1. deterministic one-way single-use automata;
2. deterministic two-way single-use automata;
3. orbit-finite monoids [5];
4. rigidly guarded MSO$^{\sim}$ [9];
5. string-to-boolean regular list functions with atoms.

The equivalence of the models in items 3 and 4 was shown in [9]; the remaining models and their equivalences are new (item 5 is an extension of the regular list functions from [7]).

Just like their classical versions, one-way and two-way single-use automata are equivalent as language acceptors, but they are no longer equivalent as transducers. For example, a two-way single-use transducer can reverse the input string, which is impossible for a one-way single-use transducer. In Sections 4 and 5 we develop the theory of single-use transducers:

In Section 4, we investigate single-use one-way transducers. For finite alphabets, one of the most important results about one-way transducers is the Krohn-Rhodes Theorem [14], which says that every Mealy machine (which is a length preserving one-way transducer) can be decomposed into certain "prime" Mealy machines. We show that the same can be done for infinite alphabets, using a single-use extension of Mealy machines. The underlying prime machines are the all the machines from the original Krohn-Rhodes theorem, plus one additional register machine which moves atoms to later positions.

In Section 5, we investigate single-use two-way transducers, and show that the corresponding class of string-to-string functions enjoys similar robustness properties as the languages discussed in Theorem 6, with four models being equivalent:

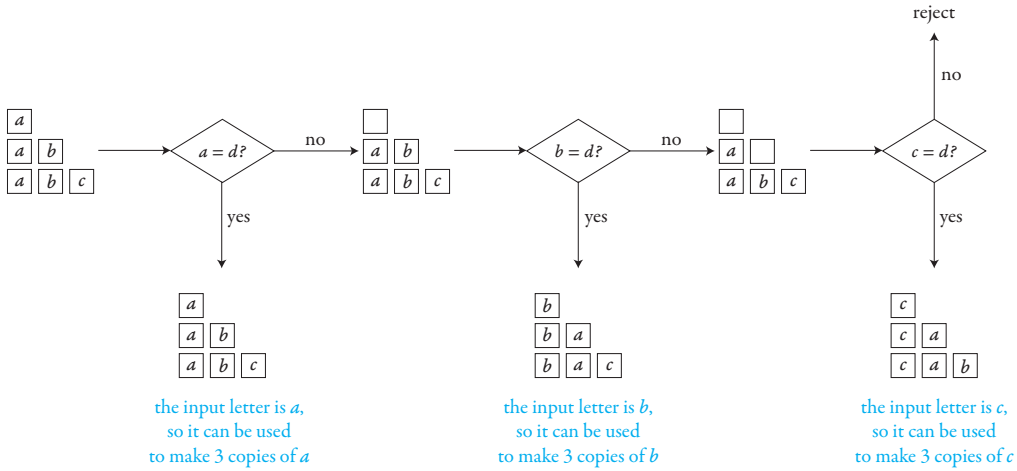


Figure 1 Updating the six registers.

1. single-use two-way transducers;
2. an atom extension of streaming string transducers [2];
3. string-to-string regular list functions with atoms;
4. compositions of certain "prime two-way machines" (Krohn & Rhodes style).

We also show other good properties of the string-to-string functions in the above items, including closure under composition (which follows from item 4) and decidable equivalence.

Summing up, the single-use restriction allows us to identify languages and string-to-string functions with infinite alphabets, which share the robustness and good mathematical theory usually associated with regularity for finite alphabets.

Due to space constraints, and a large number of results, virtually all of the proofs are in an appendix. We use the available space to explain and justify the many new models that are introduced.

2 Automata and transducers with atoms

For the rest of the paper, fix an infinite set $\mathbb{A}$, whose elements are called *atoms*. Atoms will be used to construct infinite alphabets. Intuitively speaking, atoms can only be compared for equality. It would be interesting enough to consider alphabets of the form $\mathbb{A} \times \Sigma$, for some finite Σ, as is typically done in the literature on data words [4, p. 1]. However, in the proofs, we use more complicated sets, such as the set $\mathbb{A}^2$ of pairs of atoms, the set $\mathbb{A} + \{\vdash, \dashv\}$ obtained by adding two endmarkers to the atoms, or the co-product (i.e. disjoint union) $\mathbb{A}^2 + \mathbb{A}^3$. This motivates the following definition.

▶ **Definition 2.** *A* polynomial orbit-finite set[1] *is any set that can be obtained from* $\mathbb{A}$ *and singleton sets by means of finite products and co-products (i.e. disjoint unions).*

We only care about properties of such sets that are stable under atom automorphisms, as described below. Define an *atom automorphism* to be any bijection $\mathbb{A} \to \mathbb{A}$. (This notion of automorphism formalises the intuition that atoms can only be compared for equality). Atom automorphisms form a group. There is a natural action of this group on polynomial

[1] The name "orbit-finite" is used because the above definition is a special case of orbit-finite sets discussed later in the paper, and the name "polynomial" is used to underline that the sets are closed under products and co-products.

orbit-finite sets: for elements of $\mathbb{A}$ we apply the atom automorphism, for singleton sets the action is trivial, and for other polynomial orbit-finite sets the action is lifted inductively along $+$ and $\times$ in the natural way. Let Σ and Γ be sets equipped with an action of the group of atom automorphisms – in particular, these could be polynomial orbit-finite sets. A function $f : \Sigma \to \Gamma$ is called *equivariant* if $f(\pi(x)) = \pi(f(x))$ holds for every $x \in \Sigma$ and every atom automorphism π. The general idea is that equivariant functions can only talk about equality of atoms. In the case of polynomial orbit-finite sets, equivariant functions can also be finitely represented using quantifier-free formulas [6, Lemma 1.3].

The model. We now describe the single-use machine models discussed in this paper. There are four variants: machines can be one-way or two-way, and they can recognise languages or compute string-to-string functions. We begin with the most general form – two-way string-to-string functions – and define the other models as special cases.

The machine reads the input string, extended with left and right endmarkers $\vdash, \dashv$. It uses registers to store atoms that appear in the input string. A register can store either an atom, or the undefined value $\bot$. The single-use restriction, which is highlighted in bold below, says that a register is set to $\bot$ immediately after being used.

▶ **Definition 3.** *The syntax of a* two-way single-use transducer[2] *consists of*

- *input and output alphabets Σ and Γ, both polynomial orbit-finite sets;*
- *a finite set of states Q, with a distinguished initial state $q_0 \in Q$;*
- *a finite set R of register names;*
- *a* transition function *which maps each state $q \in Q$ to an element of:*

$$\underbrace{\textit{questions}}_{\textit{question that is asked}} \times \underbrace{(Q \times \textit{actions})}_{\substack{\textit{what to do if the} \\ \textit{question has a yes answer}}} \times \underbrace{(Q \times \textit{actions})}_{\substack{\textit{what to do if the} \\ \textit{question has a no answer}}}$$

where the allowed questions and actions are taken from the following toolkit:

1. Questions.
   a. *Apply an equivariant function $f : \Sigma + \{\vdash, \dashv\} \to \{\textit{yes}, \textit{no}\}$ to the letter under the head, and return the answer.*
   b. *Are the atoms stored in registers r_1, r_2 equal and defined? If any of these registers is undefined, then the run immediately stops and rejects*[3]*.* ***This question has the side effect of setting the values of r_1 and r_2 to $\bot$.***
2. Actions.
   a. *Apply an equivariant function $f : \Sigma + \{\vdash, \dashv\} \to \mathbb{A} + \bot$ to the letter under the head, and store the result in register $r \in R$.*
   b. *Apply an equivariant function $f : \mathbb{A}^k \to \Gamma$ to the contents of distinct registers $r_1, \ldots, r_k \in R$, and append the result to the output string. If any of the registers is undefined, stop and reject.* ***This action has the side effect of setting the values of $r_1, r_2, \ldots, r_k$ to $\bot$.***
   c. *Move the head to the previous/next input position.*
   d. *Accept/reject and finish the run.*

[2] Unless otherwise noted, all transducers and automata considered in this paper are deterministic. The theory of nondeterministic single-use models seems to be less appealing.

[3] By remembering in the state which registers are defined, one can modify an automaton so that this never happens.

The semantics of the transducer is a partial function from strings over the input alphabet to strings over the output alphabet. Consider a string of the form $\vdash w \dashv$ where $w \in \Sigma^*$. A *configuration* over such a string consists of (a) a position in the string; (b) a state; (c) a register valuation, which is a function of type $R \to \mathbb{A} + \bot$; (d) an output string, which is a string over the output alphabet. A *run* of the transducer is defined to be a sequence of configurations, where consecutive configurations are related by applying the transition function in the natural way. The *output* of a run is defined to be the contents of the output string in the last configuration. An *accepting configuration* is one which executes the accept action from item 2d – accepting configurations have no successors. The *initial configuration* is a configuration where the head is over the left endmarker $\vdash$, the state is the initial state, the register valuation maps all registers to the undefined value, and the output string is empty. An *accepting run* is a run that begins in the initial configuration and ends in an accepting one. By determinism, there is at most one accepting run. The semantics of the transducer is defined to be the partial function $\Sigma^* \to \Gamma^*$, which inputs $w \in \Sigma^*$ and returns the output of the accepting run over $\vdash w \dashv$. If there is no accepting run, $f(w)$ has no value.

Special cases. A *one-way single-use transducer* is the special case of Definition 3 which does not use the "previous" action from item 2c. A *two-way single-use automaton* is the special case which does not use the output actions from item 2b. The *language* recognised by such an automaton is defined to be the set of words which admit an accepting run. A *one-way single-use automaton* is the special case of a two-way single-use automaton, which does not use the "previous" action from item 2c.

3 Languages recognised by single-use automata

In this section we discuss languages recognised by single-use automata. The main result is that one-way and two-way single-use automata recognise the same languages, and furthermore these are the same languages that are recognised by orbit-finite monoids [5], the logic rigidly guarded MSO$^\sim$ [9], and a new model called regular list functions with atoms, that will be defined in Section 5.

Orbit-finite monoids. We begin by defining orbit-finite sets and orbit-finite monoids, which play an important technical role in this paper. For more on orbit-finite sets, see the lecture notes [6]. For a tuple $\bar{a} \in \mathbb{A}^*$, an $\bar{a}$-automorphism is defined to be any atom automorphism that maps $\bar{a}$ to itself. Consider set X equipped with an action of the group of atom automorphisms. We say that $x \in X$ is *supported* by a tuple of atoms $\bar{a} \in \mathbb{A}^*$ if $\pi(x) = x$ holds for every $\bar{a}$-automorphism π. We say that a subset of X is $\bar{a}$-supported if it is an $\bar{a}$-supported element of the powerset of X; similarly we define supports of relations and functions. We say that x is finitely supported if it is supported by some tuple $\bar{a} \in \mathbb{A}^*$. Define the *$\bar{a}$-orbit of* x to be its orbit under the action of the group of $\bar{a}$-automorphisms.

▶ **Definition 4** (Orbit-finite sets). *Let X be a set equipped with an action of atom automorphisms. A subset $Y \subseteq X$ is called* orbit-finite *if (a) every element of Y is finitely supported; and (b) there exists some $\bar{a} \in \mathbb{A}^*$ such that Y is a union of finitely many $\bar{a}$-orbits.*

An equivariant orbit-finite set is the special case where the tuple $\bar{a}$ in item (b) is empty. The polynomial orbit-finite sets from Section 2 are a special case of equivariant orbit-finite sets[4]. The following notion was introduced in [5, Section 3].

▶ **Definition 5** (Orbit-finite monoid). *An* orbit-finite monoid *is a monoid where the underlying set is orbit-finite, and the monoid operation is finitely supported. If Σ is an orbit-finite set, then we say that a language $L \subseteq \Sigma^*$ is* recognised *by an orbit-finite monoid M if there is a finitely supported monoid morphism $h : \Sigma^* \to M$ and a finitely supported accepting set $F \subseteq M$ such that L contains exactly the words whose image under h belongs to F.*

In this paper, we are mainly interested in the case where both the morphism and the accepting set are equivariant. In this case, it follows that the alphabet Σ, the image of the morphism, and the recognised language all also have to be equivariant.

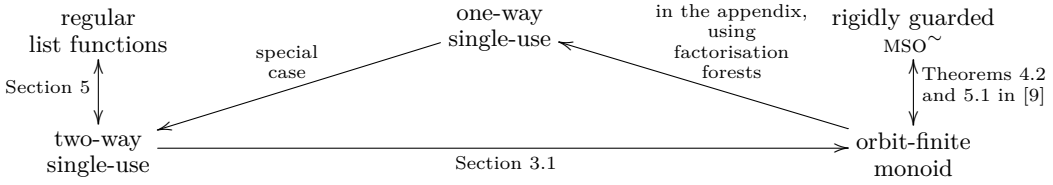
The structural theory of orbit-finite monoids was first developed in [5], where it was shown how the classical results about Green's relations for finite monoids extend to the orbit-finite setting. This theory was further investigated in [9], including a lemma stating that every orbit-finite group is necessarily finite. In the full version of this paper we build on these results, to prove an orbit-finite version of the Factorisation Forest Theorem of Simon [17, Theorem 6.1], which is used in proofs of Theorems 6 and 12.

Main theorem about languages. We are now ready to state Theorem 6, which is our main result about languages.

▶ **Theorem 6.** *Let Σ be a polynomial orbit-finite set. The following conditions are equivalent for every language $L \subseteq \Sigma^*$:*

1. *L is recognised by a single-use one-way automaton;*
2. *L is recognised by a single-use two-way automaton;*
3. *L is recognised by an orbit-finite monoid, with an equivariant morphism and an equivariant accepting set;*
4. *L can be defined in the rigidly guarded MSO$^\sim$ logic;*
5. *L's characteristic function $\Sigma^* \to \{yes, no\}$ is an orbit-finite regular list function.*

The equivalence of items 4 and 3 has been proved in [9, Theorems 4.2 and 5.1], and since we do not use rigidly guarded MSO$^\sim$ outside of the this theorem, we do not give a definition here (see [9, Section 3]). The orbit-finite regular list functions from item 5 will be defined in Section 5. The proof outline for Theorem 6 is given in the following diagram



All equivalences in the theorem are effective, i.e. there are algorithms implementing the conversions between any of the models.

The single-use restriction is crucial in the theorem. Automata without the single-use restriction – call them multiple-use – only satisfy the trivial inclusions:

[4] The converse does not hold – there exist sets that are equivariant orbit finite but not polynomial orbit finite e. g. the set of unordered pairs of atoms: $\{\{a, b\} \mid a, b \in \mathbb{A},\ a \neq b\}$.

$$\text{single-use} \overset{\text{first letter appears again [6, Exercise 91]}}{\subsetneq} \text{one-way multiple-use} \overset{\text{some letter appears twice [13, Example 11]}}{\subsetneq} \text{two-way multiple-use.}$$

Two-way multiple-use automata have an undecidable emptiness problem [15, Theorem 5.3]. For one-way (even multiple-use) automata, emptiness is decidable and even tractable in a suitable parametrised understanding [6, Corollary 9.12]. We leave open the following question: given a one-way multiple-use automaton, can one decide if there is an equivalent automaton that is single-use (by Theorem 6, it does not matter whether one-way or two-way)?

3.1 From two-way automata to orbit-finite monoids

In this section, we show the implication 2 $\Rightarrow$ 3 of Theorem 6. (This is the only proof presented in the conference version of the paper – we chose it, because it illustrates the importance of the single-use restriction). The implication states that the language of every single-use two-way automaton can also be recognised by an equivariant homomorphism into an orbit-finite monoid. In the proof, we use the Shepherdson construction for two-way automata [16] and show that, thanks to the single-use restriction, it produces monoids which are orbit-finite.

Consider a two-way single-use automaton, with k registers and let Q be the set of its states. For a string over the input alphabet (extended with endmarkers), define its *Shepherdson profile* to be the function of the type

$$\overbrace{Q \times (\mathbb{A} + \bot)^k}^{\substack{\text{state and register}\\ \text{valuation at the}\\ \text{start of the run}}} \times \overbrace{\{\leftarrow, \rightarrow\}}^{\substack{\text{does the run}\\ \text{enter from the}\\ \text{left or right}}} \quad \to \quad \{\text{accept, loop}\} + (\overbrace{Q \times (\mathbb{A} + \bot)^k}^{\substack{\text{state and register}\\ \text{valuation at the}\\ \text{end of the run}}} \times \overbrace{\{\leftarrow, \rightarrow\}}^{\substack{\text{does the run}\\ \text{exit from the}\\ \text{left or right}}})$$

that describes runs of the automaton in the natural way (see [16, Proof of Theorem 2]). The run is taken until the automaton either exits the string from either side, accepts, or enters an infinite loop. By the same reasoning as in Shepherdson's proof, one can equip the set of Shepherdson profiles with a monoid structure so that the function which maps a word to its Shepherdson profile becomes a monoid homomorphism. We use the name Shepherdson monoid for the resulting monoid (it only contains the "achievable" profiles – the image of Σ^*). It is easy to see that whether a word is accepted depends only on an equivariant property of its Shepherdson profile, and therefore the language recognised by the automaton is also recognised by the Shepherdson monoid.

It remains to show that the Shepherdson monoid is orbit-finite, which is the main part of the proof. Unlike the arguments so far, this part of the proof relies on the single-use restriction. To illustrate this, we give an example of a one-way automaton that is not single-use and whose Shepherdson monoid is not orbit-finite.

▶ **Example 7.** Consider the language over $\mathbb{A}$ of words whose first letter appears again. This language is not recognised by any orbit-finite monoid [6, Exercise 91], but it is recognised by a multiple-use one-way automaton, which stores the first letter in a register, and then compares this register with all remaining letters of the input word. For this automaton, the Shepherdson profile needs to remember all of the distinct letters that appear in the word. In particular, if two words have different numbers of distinct letters, then their Shepherdson profiles cannot be in the same orbit. Since input strings can contain arbitrarily many distinct letters, the Shepherdson monoid of this automaton is not orbit-finite.

▶ **Lemma 8.** *For every single-use two-way automaton there is some $N \in \mathbb{N}$ such that every Shepherdson profile is supported by at most N atoms.*

Before proving the lemma, we use it to show that the Shepherdson monoid is orbit-finite. In the full version of the paper, we show that if an equivariant set consists of functions from one orbit-finite set to another orbit-finite set (as is the case for the underling set in the Shepherdson monoid) and all functions in the set have supports of bounded size (as is the case thanks to Lemma 8), then the set is orbit-finite. This leaves us with proving Lemma 8.

Proof. Define a *transition* in a run to be a pair of consecutive configurations. Each transition has a corresponding question and action. A transition in a run is called *important* if its question or action involves a register that has not appeared in any action or question of the run. The number of important transitions is bounded by k – the number of registers. The crucial observation, which relies on the single-use restriction, is that if the input word, head position, and state are fixed (but not the register valuation), then the sequence of actions in the corresponding run depends only on the answers to the questions in the important transitions. This is described in more detail below.

Fix a choice of the following parameters: (a) a string over the input alphabet that might contain endmarkers; (b) an entry point of the automaton – either the left or the right end of the word; (c) a state of the automaton. We do not fix the register valuation. For a register valuation η, define $\rho(\eta)$ to be the run which begins in the configuration described by the parameters (abc) together with η, and which is maximal, i.e. it ends when the automaton either accepts, rejects, or tries to leave the fixed string. For $i \in \{0, 1, \ldots, k\}$ define $\alpha_i(\eta)$ to be the sequence of actions that are performed in the maximal prefix of the run $\rho(\eta)$ which uses at most i important transitions. The crucial observation that was stated at the beginning of this proof is that once the parameters (abc) are fixed, then the sequence of actions $\alpha_i(\eta)$ depends only on the answers to the questions asked in the first i important transitions. In particular, the function α_i has at most 2^i possible values. Furthermore, by a simple induction on i, one can show the following claim.

▷ Claim 9. The function α_i is supported by at most 2^{i+1} atoms.

Since there are at most k important transitions in a run, the above claim implies that, for every fixed choice of parameters (abc), at most 2^{k+1} atoms are needed to support the function which maps η to the sequence of actions in the run $\rho(\eta)$. In the arguments for the Shepherdson profile for a fixed word w, parameter (b) can have two values (first or last position) and parameter (c) can have at most $|Q|$ values. Therefore, at most $2|Q|2^{k+1}$ atoms are needed to support the function which takes an argument as in the Shepherdson profile, and returns the sequence of actions in the corresponding run. The lemma follows. ◀

4 A Krohn-Rhodes decomposition of one-way transducers with atoms

In this section, we present a decomposition result for single-use one-way transducers, which is a version of the celebrated Krohn-Rhodes Theorem [14, p. 454]. We think that this result gives further evidence for the good structure of single-use models. In the next section, we give a similar decomposition result for two-way single-use transducers which will be used to prove the equivalence of several other characterisations of the two-way model.

We begin by describing the classical Krohn-Rhodes Theorem. A *Mealy machine* is a deterministic one-way length-preserving transducer, which is obtained from a deterministic finite automaton by labelling transitions with output letters and ignoring accepting states.

The Krohn-Rhodes Theorem says that every function computed by a Mealy machine is a composition of functions computed by certain prime Mealy machines (which are called *reversible* and *reset* in [1, Chapter 6]). In this section, we prove a version of this theorem for orbit-finite alphabets; this version relies crucially on the single-use restriction. To distinguish the original model of Mealy machines from the single-use model described below, we will use the name *classical Mealy machine* for the Mealy machines in the original Krohn-Rhodes Theorem, i.e. the alphabets and state spaces are finite.

Define a *single-use Mealy machine* to have the same syntax as in Definition 3, with the following differences: there are no "next/previous" actions from item 2c, but the output action from item 2b has the side effect of moving the head to the next position. A consequence is that a Mealy machine is length-preserving, i.e. it outputs exactly one letter for each input position. Furthermore, there are no endmarkers and no "accept" or "reject" actions from item 2d; the automaton begins in the first input position and accepts immediately once its head leaves the input word from the right.

▶ **Example 10.** Define *atom propagation* to be the following length-preserving function. The input alphabet is $\mathbb{A} + \{\epsilon, \downarrow\}$ and the output alphabet is $\mathbb{A} + \bot$. If a position i in the input string has label $\downarrow$ and there is some (necessarily unique) position $j < i$ with an atom label such that all positions strictly between j and i have label ϵ, then the output label of position i is the atom in input position j. For all other input positions, the output label is $\bot$. Here is an example of atom propagation:

input	1	2	ϵ	ϵ	$\downarrow$	$\downarrow$	3	ϵ	ϵ	$\downarrow$	ϵ	$\downarrow$
output	$\bot$	$\bot$	$\bot$	$\bot$	2	$\bot$	$\bot$	$\bot$	$\bot$	3	$\bot$	$\bot$

Atom propagation is computed by a single-use Mealy machine, which stores the most recently seen atom in a register, and outputs the register at the nearest appearance of $\downarrow$.

The following example illustrates some of the technical difficulties with single-use Mealy machines: It is often useful to consider a Mealy machine that computes the run of another Mealy machine i. e. decorate every input position with the state and the register valuation that the Mealy machine will have after reading the input up to (but not including) that position. As shown in the following example the single-use restriction makes this construction impossible.

▶ **Example 11.** Consider the single-use Mealy machine that implements the atom propagation function from Example 10. This machine has only one register. Every time it sees an atom value, it stores the value in the register and every time it sees $\downarrow$, and the register is non-empty, the machine outputs the register's content. We claim that the run of this machine cannot be computed by a Mealy machine. If it could, we would be able to use it to construct a Mealy machine that given a word over $\mathbb{A}$, equips every position with the atom from the first position. This would easily lead to a construction of a single-use automaton for the language "the first letter appears again" (from Example 7) which, as we already know, is impossible.

The Krohn-Rhodes Theorem, both in its original version and in our orbit-finite version below, says that every Mealy machine can be decomposed using two types of composition:

$$\frac{\Sigma^* \xrightarrow{f} \Gamma^* \quad \Gamma^* \xrightarrow{g} \Delta^*}{\Sigma^* \xrightarrow{g \circ f} \Delta^*} \text{ sequential} \qquad \frac{\Sigma_1^* \xrightarrow{f_1} \Gamma_1^* \quad \Sigma_2^* \xrightarrow{f_2} \Gamma_2^*}{(\Sigma_1 \times \Sigma_2)^* \xrightarrow{f_1 | f_2} (\Gamma_1 \times \Gamma_2)^*} \text{ parallel}$$

The sequential composition is simply function composition. The parallel composition – which only makes sense for length preserving functions – applies the function f_i to the i-th projection of the input string.

▶ **Theorem 12.** *Every total function computed by a single-use Mealy machine can be obtained, using sequential and parallel composition, from the following* prime functions*:*

1. Length-preserving homomorphisms. *Any function of type* $\Sigma^* \to \Gamma^*$*, where* Σ *and* Γ *are polynomial orbit-finite, obtained by lifting to strings an equivariant function of type* $\Sigma \to \Gamma$*.*
2. Classic Mealy machines. *Any function computed by a classical Mealy machine.*
3. Atom propagation. *The atom propagation function from Example 10.*

By the original Krohn-Rhodes theorem, classical Mealy machines can be further decomposed.

Define a *composition of primes* to be any function that can be obtained from the prime functions by using a parallel and sequential composition. In this terminology, Theorem 12 says that every function computed by a single-use Mealy machine is a composition of primes. In the full version of the paper we show that the converse is also true: every prime function is computed by a single-use Mealy machine, and single-use Mealy machines are closed under both kinds of composition.

Decomposition of single-use one-way transducers. A Mealy machine is the special case of a single-use one-way transducer which is length preserving, and does not see an endmarker. A corollary of Theorem 12 is that, in order to generate all total functions computed by single-use one-way transducers, it is enough to add two items to the list of prime functions from Theorem 12: (a) a function $w \mapsto w\dashv$ which appends an endmarker[5], and (b) equivariant homomorphisms over polynomial orbit-finite alphabets that are not necessarily length-preserving.

5 Two-way single-use transducers

In this section, we turn to two-way single-use transducers. For them, we show three other equivalent models: (a) compositions of certain two-way prime functions; (b) an atom variant of the streaming string transducer (SST) model of Alur and Černý from [2]; and (c) an atom variant of the regular string functions from [7]. We believe that the atom variants of items (b) and (c), as described in this section, are the natural atom extensions of the original models; and the fact that these extensions are all equivalent to single-use two-way transducers is a further validation of the single-use restriction.

We illustrate the transducer models using the functions from the following example.

▶ **Example 13.** Consider some polynomial orbit-finite alphabet Σ. The input and output alphabets are the same, namely Σ extended with a separator symbol $|$. Define *map reverse* (respectively, *map duplicate*) to be the function which reverses (respectively, duplicates) every string between consecutive separators, as in the following examples:

$$\underbrace{12||345|678|9 \;\mapsto\; 21||543|876|9}_{\text{map reverse}} \qquad \underbrace{12||345|678|9 \;\mapsto\; 1212||345345|678678|99}_{\text{map duplicate}}$$

Both functions can be computed by single-use two-way transducers. These functions will be included in the prime functions for two-way single-use transducers, as discussed in item (a) at the beginning of this section.

[5] This function accounts for the fact that a one-way transducer (contrary to a Mealy machine) may perform some computation and produce some output at the end of the input word.

Streaming string transducers with atoms. A streaming string transducer with atoms has two types of registers: atom registers $r, s, \ldots$ which are the same as in Definition 3, and string registers $A, B, C, \ldots$ which are used to store strings over the output alphabet. Both kinds of registers are subject to the single-use restriction, which is highlighted in bold in the following definition.

▶ **Definition 14** (Streaming string transducer with atoms). *Define the syntax of a* streaming string transducer (SST) with atoms *in the same way as a one-way single-use transducer (variant of Definition 3), except that the model is additionally equipped with a finite set of* string registers*, with a designated* output string register*. The actions are the same as for one-way single-use transducers except that the output action is replaced by two kinds of actions:*

1. *Apply an equivariant function $f : \mathbb{A}^k \to \Gamma$ to the contents of distinct registers $r_1, \ldots, r_k \in R$, and put the result into string register A (overwriting its previous contents). If any of these registers is undefined, then the run immediately stops and rejects.* ***This action has the side effect of setting the values of $r_1, r_2, \ldots, r_{k_j}$ to $\bot$.***
2. *Concatenate string registers A and B, and put the result into string register C.* ***This action has the side effect of setting A and B to the empty string.***

The output of a streaming string transducer is defined to be the contents of the designated output register when the "accept" action is performed. In the atomless case, when no atom registers are allowed and the input and output alphabets are finite, the above definition is equivalent to the original definition of streaming string transducers from [2].

▶ **Example 15.** Consider the map reverse function from Example 13, with alphabet $\mathbb{A}$. To compute it, we use two string registers A and B, with the output register being B. When reading an atom $a \in \mathbb{A}$, the transducer executes an action $A := aA$. (This action needs to be broken into simpler actions as in Definition 14 and requires auxiliary registers). When reading a separator symbol, the automaton executes action $B := B|A$, which erases the content of register A. Similar idea works for map duplicate – it uses two copies of register A.

Regular list functions with atoms. Our last model is based on the regular list functions from [7]. Originally, the regular list functions were introduced to characterise two-way transducers (over finite alphabets), in terms of simple prime functions and combinators [7, Theorem 6.1]. The following definition extends the original definition[6] in only two ways: we add an extra datatype $\mathbb{A}$ and an equality test $\mathsf{eq} : \mathbb{A}^2 \to \{\text{yes,no}\}$.

▶ **Definition 16** (Regular list functions with atoms). *Define the* datatypes *to be sets which can be obtained from $\mathbb{A}$ and singleton sets, by applying constructors for products $\tau \times \sigma$, co-products $\tau + \sigma$ and lists τ^*. The class of* regular list functions with atoms *is the least class which:*

1. *contains all equivariant constant functions;*
2. *contains all functions from Figure 2, and an equality test $\mathsf{eq} : \mathbb{A}^2 \to \{yes,no\}$;*
3. *is closed under applying the following combinators:*
   a. *`comp` function composition $(f, g) \mapsto f \circ g$;*
   b. *`pair` function pairing $(f_0, f_1) \mapsto (x \mapsto (f_0(x), f_1(x)))$;*
   c. *`cases` function co-pairing $(f_0, f_1) \mapsto ((i, a) \mapsto f_i(a))$;*
   d. *`map` lifting functions to lists $f \mapsto ([a_1, \ldots, a_n] \mapsto [f(a_1), \ldots, f(a_n)])$.*

[6] In [7], the group product operation has output type G^*, while this paper uses $(G \times \sigma)^*$. This difference is due to an error in [7].

$\texttt{project}_i$	:	$(\sigma_0 \times \sigma_1) \to \sigma_i$ projection $(a_0, a_1) \mapsto a_i$
$\texttt{coproject}_i$	:	$\sigma_i \to (\sigma_0 + \sigma_1)$ coprojection $a_i \mapsto (i, a_i)$
distr	:	$(\sigma_1 + \sigma_2) \times \tau \to (\sigma_1 \times \tau) + (\sigma_2 \times \tau)$ distribution $((i, a), b) \mapsto (i, (a, b))$
reverse	:	$\sigma^* \to \sigma^*$ list reverse $[a_1, \ldots, a_n] \mapsto [a_n, \ldots, a_1]$
concat	:	$(\sigma^*)^* \to \sigma^*$ list concatenation, defined by $[] \mapsto []$ and $[a] \cdot l \mapsto a \cdot \mathit{concat}(l)$
append	:	$(\sigma \times \sigma^*) \to \sigma^*$ append, defined by $(a, l) \mapsto [a] \cdot l$
coappend	:	$\sigma \to (\sigma \times \sigma^*) + \bot$ the opposite of append, defined by $[] \mapsto (1, \bot)$ and $[a] \cdot l \mapsto (0, (a, l))$
block	:	$(\sigma + \tau)^* \to (\sigma^* + \tau^*)^*$ group the list into maximal connected blocks from σ^* or τ^*
group	:	$(G \times \sigma)^* \to (G \times \sigma)^*$ $[(g_1, a_1), \ldots, (g_n, a_n)] \mapsto [(1, a_1), (g_1, a_2), (g_1 g_2, a_3), \ldots, (g_1 \cdots g_{n-1}, a_n)]$

Figure 2 For every datatypes $\tau, \tau_0, \tau_1, \sigma$, every finite group G, and every $i \in \{0, 1\}$ the above functions are regular list functions with atoms.

Every polynomial orbit-finite set is a datatype (actually, polynomial orbit-finite sets are exactly the datatypes that do not use lists), and therefore it makes sense to talk about regular list functions with atoms that describe string-to-string functions with input and output alphabets that are polynomial orbit-finite sets. Also, one can consider string-to-boolean functions – they describe languages, and are the model mentioned in item 5 of Theorem 6.

▶ **Example 17.** We show that map reverse from Example 13 is a regular list function with atoms. Consider an input string, say

$$[1, 2, |, |, 3, 4, 5, |, 6, 7, 8, |, 9] \in (\mathbb{A} + |)^*.$$

Apply the prime `block` function, yielding

$$[[1, 2], [|, |], [3, 4, 5], [|], [6, 7, 8], [|], [9]] \in (\mathbb{A}^* + |^*)^*.$$

Using the `cases` and `map` combinators, apply `reverse` to all list items, yielding

$$[[2, 1], [|, |], [5, 4, 3], [|], [8, 7, 6], [|], [9]] \in (\mathbb{A}^* + |^*)^*.$$

To get the final output, apply `concat`. A similar idea works for map duplicate, except we need to derive the string duplication function:

$$w \overset{\texttt{pair}(\ldots)}{\mapsto} (w, [w]) \overset{\texttt{append}}{\mapsto} [w, w] \overset{\texttt{concat}}{\mapsto} ww$$

Equivalence of the models. The main result of this section is that all models described above are equivalent, and furthermore admit a decomposition into prime functions in the spirit of the Krohn-Rhodes theorem. Since the functions discussed in this section are no longer length-preserving, the Krohn-Rhodes decomposition uses only sequential composition.

▶ **Theorem 18.** *The following conditions are equivalent for every total function $f : \Sigma^* \to \Gamma^*$, where Σ and Γ are polynomial orbit-finite sets:*

1. *f is computed by a two-way single-use transducer;*
2. *f is computed by a streaming string transducer with atoms;*
3. *f is a regular list function with atoms;*
4. *f is a sequential composition of functions of the following kinds:*
   a. *single-use Mealy machines;*
   b. *equivariant homomorphisms that are not necessarily length-preserving;*
   c. *map reverse and map duplicate functions from Example 13.*

In the future, we plan to extend the above theorem with one more item, namely a variant of MSO transductions based on rigidly guarded MSO$^{\sim}$. The models in items 3 and 4 are closed under sequential composition, and therefore the same is true for the models in items 1 and 2; we do not know any direct proof of composition closure for items 1 and 2, which contrasts the classical case without atoms [8, Theorem 2]. The Krohn-Rhodes decomposition from item 4, in the case without atoms, was present implicitly in [7]; in this paper we make the decomposition explicit, extend it to atoms, and leverage it to get a relatively simple proof of Theorem 18. Even for the reader interested in transducers but not atoms, our Krohn-Rhodes-based proof of Theorem 18 might be of some independent interest.

Here are some immediate corollaries of Theorem 18:

1. Every function in item 4 is computed by a two-way single-use transducer which is reversible in the sense of [10, p. 2]; hence two-way single-use transducers can be translated into reversible ones.
2. Since the equivalence in Theorem 18 also works for functions with yes/no outputs, it follows that items 2 and 5 in Theorem 6 are equivalent.
3. If f is a transducer from the class described in Theorem 18, then the language class described in Theorem 6 is preserved under inverse images of f.

All conversions between the models in Theorem 18 are effective. Our last result concerns the equivalence problem for these models, which is checking if two transducers compute the same function. Using a reduction to the equivalence problem for *copyful* streaming string transducers without atoms [12, p. 81], we prove the following result:

▶ **Theorem 19.** *Equivalence is decidable for streaming string transducers with atoms (and therefore also for every other of the equivalent models from Theorem 18).*

References

1 Ginzburg Abraham. *Algebraic Theory of Automata*. Elsevier, 1968.

2 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

3 Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4):702–715, January 2010.

4 Mikołaj Bojańczyk. Automata for Data Words and Data Trees. In Christopher Lynch, editor, *Rewriting Techniques and Applications, RTA, Edinburgh, Scottland, UK*, volume 6 of *LIPIcs*, pages 1–4. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

5 Mikołaj Bojańczyk. Nominal Monoids. *Theory Comput. Syst.*, 53(2):194–222, 2013.

6 Mikołaj Bojańczyk. Slightly infinite sets, 2019. URL: `https://www.mimuw.edu.pl/~bojan/paper/atom-book` [cited version of September 11, 2019].

7 Mikołaj Bojańczyk, Laure Daviaud, and Shankara Narayanan Krishna. Regular and First-Order List Functions. In *Logic in Computer Science, LICS, Oxford, UK*, pages 125–134. ACM, 2018.

8 Michal Chytil and Vojtech Jákl. Serial Composition of 2-Way Finite-State Transducers and Simple Programs on Strings. In *International Colloquium on Automata, Languages and Programming, ICALP, Turku, Finland*, volume 52 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 1977.

9 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. Logics with rigidly guarded data tests. *Logical Methods in Computer Science*, 11(3), 2015.

10 Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On reversible transducers. In *International Colloquium on Automata, Languages and Programming, ICALP, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 113:1–113:12. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.

11 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *SIGLOG News*, 3(3):4–19, 2016.

12 Emmanuel Filiot and Pierre-Alain Reynier. Copyful Streaming String Transducers. In Matthew Hague and Igor Potapov, editors, *Reachability Problems, RP , London, UK*, volume 10506 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 2017.

13 Michael Kaminski and Nissim Francez. Finite-Memory Automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

14 Kenneth Krohn and John Rhodes. Algebraic theory of machines. i. prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–450, 1965.

15 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

16 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, April 1959.

17 Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990.