A categorical perspective on regular functions

Mikołaj Bojańczyk, Lê Thành Dũng Nguyễn

November 23, 2022

Abstract

We consider regular string-to-string functions, i.e. functions that are recognized by copyless streaming string transducers, or any of their equivalent models, such as deterministic two-way automata. We give yet another characterization: functors from the category of semigroups together to itself, together with a certain output function that is a natural transformation.

1 Introduction

The purpose of this paper is to give a characterization of the (linear) regular string-to-string functions. This is a fundamental class of functions, which has many equivalent descriptions, including deterministic two-way automata [She59, Note 4], copyless streaming string transducers [AČ10, Section 3], MSO transductions [EH01, Theorem 13], combinators [AFR14, Section 2], a functional programming language [BDK18, Section 6] or decompositions *à la* Krohn–Rhodes [BS20, Theorem 18, item 4].

The present paper adds a new characterization to the list, which uses minimal syntax, and refers only to basic concepts, such as semigroups and natural transformations. We prove that a string-to-string function of type $\Sigma^+ \to \Gamma^+$ is regular, in the sense that it is recognized by any of the equivalent models described in the previous paragraph, if and only if it can be decomposed as

$$\Sigma^+ \xrightarrow{h} \mathsf{F}(\Gamma^+) \xrightarrow{\mathrm{out}} \Gamma^+$$

where F is some finiteness preserving functor from the category of semigroups to itself, h is some semigroup homomorphism, and the output function out is natural in the sense of natural transformations. Here, a finiteness preserving functor is one that maps finite semigroups to finite semigroups.

The theorem also extends to some other algebraic structures, such as trees modelled via forest algebra. However, our proof uses properties of the underlying algebraic structure which seem to fail for some structures such as groups or algebras corresponding to weighted automata; we do not know if our proof can be extended to these structures, or even if the theorem itself is true.

2 Transducer semigroups and the main theorem

We use some basic notions from category theory, such as functors or natural transformations. We do not assume that the reader has a background in category theory, beyond the two most basic notions: category (objects together with morphisms between them, such that morphisms can be composed) and functor (a map between two categories, which maps objects of the first category to objects of the second category, and likewise for morphisms, which is consistent with composition of morphisms). In this paper, we will be working mainly with two categories:

Sets. Objects are sets, morphisms are functions between them.

Semigroups. Objects are semigroups, morphisms are semigroup homomorphisms.

One example of a functor is the *forgetful functor* from the category of semigroups to the category of sets, which maps a semigroup to its underlying set, and a semigroup homomorphism to the corresponding function on sets.

Example 1. In this paper, we will mainly be studying functors from the category of semigroups to itself; such functors can be seen as semigroup constructions. Several such constructions are listed in the following example.

- **Tuples.** The functor maps a semigroup A to its square A^2, with the semigroup operation being defined coordinate-wise. The functor extends to morphisms in the expected way. This functor also makes sense for higher powers, including infinite powers, such as A^ω.
- **Reverse.** The functor maps a semigroup A to the semigroup where the underlying set is the same, but the multiplication is reversed, i.e. the product of a and b in the new semigroup is the product of b and a in the old semigroup. Morphisms are not changed by the functor; it is easy to see that they retain the homomorphism property despite the change in the multiplication operation.
- **Lists.** The functor maps a semigroup A to the free semigroup A^+ which consists of non-empty strings over the alphabet A equipped with string concatenation. On morphisms, the functor is defined coordinate-wise.
- **Powerset.** The (covariant) powerset functor maps a semigroup A to the powerset semigroup $\mathsf{P}A$, where the underlying set is the family of all subsets of A, and the semigroup operation is defined by

$$(A_1, A_2) \quad \mapsto \quad \{a_1 a_2 \mid \ a_1 \in A_1 \text{ and } a_2 \in A_2 \ \}.$$

  There are several variants of this construction: we could, for example, require the subsets to be nonempty, or finite, or both.

□

We now present the central definition of this paper.

Definition 2.1 *A* transducer semigroup *is defined to be a functor* F *from the category of semigroups to itself, together with an output mechanism defined as follows. For each semigroup* A*, there is an output function*

$$\underbrace{\mathrm{out}_A : \mathsf{F}A \to A,}_{\substack{\text{a function between two sets, that} \\ \text{is not necessarily a semigroup homomorphism}}}$$

and this family of functions is natural in the sense that the following diagram commutes for every semigroup homomorphism h*:*

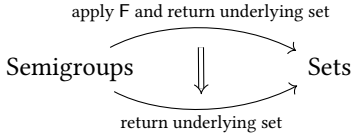
$$\begin{array}{ccc} \mathsf{F}A & \xrightarrow{\mathsf{F}h} & \mathsf{F}B \\ {\scriptstyle \mathrm{out}_A}\downarrow & & \downarrow{\scriptstyle \mathrm{out}_B} \\ Awhich & \xrightarrow[h]{} & B \end{array}$$

A function between two semigroups $f : A \to B$*, not necessarily a semigroup homomorphism, is* recognized *by a transducer semigroup if it can be decomposed as*

$$A \xrightarrow{h} \mathsf{F}B \xrightarrow{\mathrm{out}_B} B$$

for some semigroup homomorphism h*.*

In the language of category theory, the naturality condition from the above definition says that the output mechanism is a natural transformation of type



Example 2. Consider the transducer semigroup in which the functor is the identity, and the output mechanism is also the identity. The functions of type $A \to B$ that are recognized by this transducer semigroup are exactly the semigroup homomorphisms. □

Example 3. Consider the transducer semigroup in which the functor is the identity, and the output mechanism is the function

$$\begin{aligned} A &\to A \\ a &\mapsto aa. \end{aligned}$$

The functions of type $A \to B$ that are recognized by this transducer semigroup are exactly the functions of the form $a \mapsto h(a)h(a)$ where h is some semigroup homomorphism. In particular, if h is the identity on Σ^+, then we get the duplicating function on strings over alphabet Σ. □

Example 4. Consider the reversing functor, which maps a semigroup A to the semigroup $\mathsf{F}A$ in which the underlying set is the same, the identity is the same, but the semigroup operation is reversed:

$$\underbrace{ab}_{\text{in } \mathsf{F}A} \quad \stackrel{\text{def}}{=} \quad \underbrace{ba}_{\text{in } A}.$$

This is easily seen to be a functor, with semigroup homomorphisms being mapped to the same functions. Define the output function to be the identity. Using this transducer semigroup, we can recognize the reversing function $f : \Sigma^+ \to \Sigma^+$. More generally, the functions recognized by this transducer semigroup are of the form: reverse, followed by some semigroup homomorphism. □

Example 5. Consider the list functor $A \mapsto A^+$ described in Example 1. The output semigroup is the free semigroup with generators A, and the functor is defined coordinate-wise on morphisms. Consider the following output mechanism

$$\begin{aligned} A^+ &\to A \\ [a_1, \ldots, a_n] &\mapsto \underbrace{(a_1 \cdots a_n) \cdots (a_1 \cdots a_n)}_{n \text{ times}}. \end{aligned}$$

This transducer semigroup recognizes the squaring function $\Sigma^+ \to \Sigma^+$ that is illustrated in the following example

$$123 \mapsto 123123123.$$

□

Example 6. Here is a generalization of the previous example. The functor continues to be A^+. The output mechanism $A^+ \to A$ is given by a sequence of strings

$$w_1 \in \{1\}^+, \quad w_2 \in \{1,2\}^+, \quad w_3 \in \{1,2,3\}^+, \quad \ldots.$$

When applied to lists of length n, the output mechanism is

$$A^n \xrightarrow{f \mapsto f \text{ applied to } w_n} A^+ \xrightarrow{\text{semigroup operation}} A.$$

□

2.1 Two simple characterizations

We begin with two simple theorems that describe two classes of string-to-string functions in terms of the transducer semigroups that can be used to recognize them: all functions (Theorem 2.2) and functions that reflect recognizability (Theorem 2.3). These two theorems have simple proofs. In the next Section 3 we present a third, more interesting, theorem about regular functions.

All functions. The first theorem shows that every function between two semigroups is recognized by some transducer semigroup.

Theorem 2.2 *Every function between two semigroups, not necessarily a semigroup homomorphism, is recognized by some transducer semigroup.*

Proof
Consider some semigroup A. Based on this semigroup we will define a transducer semigroup which will recognize all functions from A to some other semigroup. Define a functor as follows:

$$\mathsf{F}B = A \times \underbrace{(A \to B)}_{\substack{\text{the set of all functions, viewed as a semigroup} \\ \text{with the trivial semigroup operation } xy = x}}.$$

The functor is defined on morphisms as follows: the first coordinate, corresponding to A, is not changed, and the second coordinate, corresponding to the set of functions, is transformed coordinate-wise, when viewed as a tuple indexed by A. (This is similar to the tuple construction in Example 1, except that the semigroup structure of $A \to B$ is not defined coordinate-wise.) This is easily seen to be a functor. The output mechanism, which is easily seen to be natural, is function application i.e.

$$(a, f) \mapsto f(a).$$

The transducer semigroup defined above recognizes the function $f : A \to B$. The appropriate homomorphism is

$$a \in A \mapsto (a, f).$$

□

Recognizability reflecting functions. Recall that a language

$$A \xrightarrow{L} \{\text{yes, no}\}$$

is called *recognizable* if it factors through a homomorphism from A to some *finite* semigroup. More generally, recognizable maps from the semigroup A to some set, not necessarily with two elements, are those that factor through some homomorphism into a finite semigroup. A function is called *recognizability reflecting* if inverse images of recognizable languages are also recognizable. The following example shows that there are a lot of recognizability reflecting functions.

Example 7. Consider the semigroup $\mathbb{N}$ of natural numbers with addition. In this semigroup, the recognizable languages are ultimately periodic subsets. We will show that there are uncountably many recognizability preserving functions of type $\mathbb{N} \to \mathbb{N}$. To see this, consider any non-decreasing function

$$g : \mathbb{N} \to \mathbb{N},$$

and define f to be the composition of g with the factorial operation

$$f : n \mapsto g(n)!$$

The property of the factorial operation is that every ultimately periodic subset of $\mathbb{N}$ contains all or no factorials, up to finitely many exceptions. Therefore, the inverse image of any ultimately periodic set under f will be either finite or co-finite, and therefore also ultimately periodic. $\square$

We now present a second characterization, which concerns functions between semigroups that are recognizability reflecting.

Theorem 2.3 *The following conditions are equivalent for a function $f : A \to B$, which is not necessarily a semigroup homomorphism:*

1. *f is recognizability reflecting, which means for every recognizable language*

$$B \xrightarrow{L} \{\textit{yes, no}\}$$

   the language $f; L$ is also recognizable.

2. *f is recognized by a transducer semigroup such that the output mechanism*

$$\mathrm{out}_B : \mathsf{F}B \to B$$

   is recognizable for every finite semigroup B.

Proof
For the implication $1 \Rightarrow 2$ we use a similar construction as in the proof of Theorem 2.2. Consider some function $f : A \to B$, which is recognizability reflecting. Define a functor as follows:

$$\mathsf{F}C = A \times \underbrace{\mathrm{Hom}(B, C)}_{\substack{\text{the set of all semigroup homomorphisms, viewed} \\ \text{as a semigroup with the trivial semigroup operation } xy = x}}.$$

On morphisms, the functor is defined as in the proof of Theorem 2.2, and the output mechanism is function application with f inserted as an interface i.e.

$$\mathrm{out} : (a, g) \mapsto g(f(a)).$$

By the assumption that f is recognizability reflecting, the output mechanism out_C is recognizable for finite C [**(Tito) not sure this holds when B isn't finitely generated?**]. The transducer semigroup defined above recognizes the function f via the homomorphism

$$a \in A \mapsto (a, \mathrm{id}).$$

Consider now the converse implication $1 \Leftarrow 2$. Take a function $f : A \to B$ between two semigroups that satisfies 2, i.e. it is a composition

$$A \xrightarrow{h} \mathsf{F}B \xrightarrow{\mathrm{out}_B} B$$

where h is some homomorphism. We want to show that f is recognizability reflecting. To prove this, consider some recognizable language over the output semigroup, i.e. a composition of some homomorphism from B into a finite semigroup, followed by an arbitrary boolean-valued function

$$B \xrightarrow[g]{} C \xrightarrow[\text{accepting set}]{} \{\text{yes,no}\}$$

We want to show that is inverse image of the language under f is also recognizable. Consider the following diagram.

$$\begin{array}{ccccc} A & \xrightarrow{h} & \mathsf{F}B & \xrightarrow{\text{out}_B} & B \\ & & \downarrow{\scriptstyle \mathsf{F}g} & & \downarrow{\scriptstyle g} \\ & & \mathsf{F}C & \xrightarrow[\text{out}_A]{} C \xrightarrow[\text{accepting set}]{} & \{\text{yes, no}\} \end{array}$$

The upper path from A to $\{\text{yes, no}\}$ describes the inverse image under f. The middle rectangle commutes by naturality of the output mechanism, and therefore the upper path describes the same function as the lower path. The lower path is a recognizable function, since the first three arrows on it are semigroup homomorphisms. □

3 The regular functions

The two constructions in Theorems 2.2 and 2.3 were rather straightforward, and amounted to little more than symbol pushing. In this section, we present a more advanced characterization, which concerns the regular functions, i.e. string-to-string functions recognized by streaming string transducers and their equivalent models. In the characterization, we require that the functor is finiteness preserving, i.e. it maps finite semigroups to finite semigroups. It turns out that the naturality of the output mechanism interacts with the condition that the functor is finiteness preserving, resulting in a strong restriction on the expressive power.

Example 8. Consider the powerset functor $\mathsf{P}A$ from Example 1. This is a finiteness preserving functor, since the powerset of a finite set is also finite. One could imagine that using the powerset functor we could construct some transducer semigroup which recognizes functions that are not regular, e.g. because they have exponential growth. It turns out that this is impossible, because there is no possible output mechanism, i.e. no natural transformation

$$\mathsf{P}A \xrightarrow{\text{out}_A} A.$$

There reason why there is no such natural transformation is that it require some kind of choice, which would contradict naturality. More formally, let A be a semigroup with two elements, with a trivial semigroup operation defined by $xy = x$. The output mechanism needs to choose some element $a \in A$ to the full set $A \in \mathsf{P}A$. However, none of the two choices is right, because if we take any semigroup homomorphism

$f : A \to A$ such that $f(A) = A$, then naturality of the output mechanism implies that $a = f(a)$. If f is the homomorphism that swaps the two elements, then we get a contradiction. $\square$

We now state the main theorem of this paper. Unlike the previous characterizations, the statement is stated in terms of functions between free semigroups Σ^+ and Γ^+, because the models defining regular functions are defined for string-to-string functions, and not transformations of abstract semigroups. (Some of the models, such as streaming string transducers or two-way automata, easily make sense when the output is an abstract semigroup, but the string structure of the input semigroup seems to be essential for all the models.)

Theorem 3.1 *Let Σ and Γ be finite alphabets. The following conditions are equivalent for a function $f : \Sigma^+ \to \Gamma^+$, which is not necessarily a semigroup homomorphism:*

1. *f is a regular function, as defined in Section 3.1.*
2. *f is recognized by a transducer semigroup $(\mathsf{F}, \mathrm{out})$ such that for every finite semigroup A, the semigroup $\mathsf{F}A$ is also finite.*

Before proceeding with the proof, we comment on the role of empty strings. Regular functions are usually defined for possibly empty strings, i.e. functions of type $\Sigma^* \to \Gamma^*$. We use nonempty strings, because it will be more convenient to work with semigroups, and the free semigroup construction produces nonempty strings. To extend the construction to functions that can output possibly empty strings, while still working with semigroups, we could modify the type of the output mechanism to be

$$\mathrm{out}_A : \mathsf{F}A \to \underbrace{A + 1}_{\substack{\text{disjoint union of underlying set of } A \\ \text{with one extra element representing the empty word}}},$$

under this modification the same proof as presented below would give us exactly the regular functions with possibly empty outputs. The empty string as an input is less important, since we can always extend the source code of a transducer that inputs nonempty strings with an extra line which says how to handle an empty input string.

The left-to-right implication is relatively straightforward, and presented in Section 3.1, together with the definition of streaming string transducers. The main part of the proof is devoted to the right-to-left implication. The proof is presented in a way which, if sometimes slightly verbose, makes it easier to see how it can be adapted to other algebraic structures instead of semigroups (such as forest algebras).

3.1 From a regular function to a transducer semigroup

(TODO fill in)

3.2 Term operations and natural transformations

We now turn to proving the right-to-left implication in Theorem 3.1.

Let us begin with some notation. We write 1 for the semigroup that has exactly one element; it is a *terminal object*, that is, it admits a unique homomorphism from every other semigroup A, which will be denoted by $! : A \to 1$ (the notation has no connection with the factorial function on numbers). Another construction that will be used heavily in the proof is the *coproduct* of two semigroups A and B, which is denoted by $A \oplus B$. This is a semigroup which consists of words over an alphabet that is the disjoint union of A and B, restricted to words which are nonempty and alternating in the sense that two consecutive elements cannot belong to the same semigroup. The semigroup operation is defined in the expected way. The coproduct deserves its name due to the following universal property: for every pair of semigroup homomorphisms

$$f : A \to C \qquad \text{and} \qquad g : B \to C$$

there is a unique semigroup homomorphism

$$f \text{ or } g : A \oplus B \to C$$

that coincides with f (resp. g) on the subsemigroup of $A \oplus B$ consisting of words with a single letter from A (resp. B).

Next, we introduce some terminology that will be used in the proof, concerning polynomial functors, and copyless operations between them. They will be used as the register structure for an SST in our proof.

3.2.1 Polynomial functors

Define a *polynomial functor* to be a functor from the category of semigroups to the category of sets, which is of the form

$$A \quad \mapsto \quad \coprod_{q \in Q} A^{\text{dimension of } q},$$

where Q is some possibly infinite set, called the *components*, with each component having an associated *dimension* in $\{0, 1, \ldots\}$. On morphisms, the functor works in the expected way, i.e. coordinate-wise. A *finite polynomial functor* is one that has finitely many components.

Example 9. A crucial property that will be used in our proof is that the functor

$$A \mapsto \text{underlying set of } \underbrace{1 \oplus A}_{\text{coproduct with the trivial semigroup}}$$

is in fact a polynomial functor (but not a finite polynomial functor). This is because for every semigroup A there is a bijective correspondence

$$1 \oplus A \quad \simeq \quad \coprod_{q \in 1 \oplus 1} A^{\text{dimension of } q}, \tag{1}$$

where the dimension of q is defined to be the number of times that the second copy of 1 appears in q. Furthermore, the bijective correspondence in (1) is natural in A, and therefore there is a natural bijection between the functor $1 \oplus A$ and some polynomial functor. Also, if two polynomial functors are connected by a natural bijection, then they are the same, up to renaming of the components, and therefore the representation in (1) is unique up to renaming of components. By uniqueness, we will simply speak of $1 \oplus A$ as being a polynomial functor. $\square$

3.2.2 Copyless natural transformations.

Among all natural transformations between polynomial functors, we will be interested mainly in those that are *copyless*. To define this notion, we will first observe that every natural transformation between polynomial functors arises from some syntactic description, and within this syntactic description, the copyless restriction can easily be phrased.

We begin with *monomial functors*, i.e. polynomial functors with one component. Consider two monomial functors

$$\mathsf{F}A = A^k \qquad \mathsf{G}A = A^\ell \qquad \text{where } k, \ell \in \{0, 1, \ldots\}.$$

What is the possible form of a natural transformation between these functors? One way to create such a natural transformation is to take a function of type

$$\{1, \ldots, \ell\} \to \{1, \ldots, k\}^+,$$

which will be called the *syntactic description* of the natural transformation, and to define the natural transformation as follows: for a semigroup A the natural transformations gives the function that inputs $\bar{a} \in A^k$ and outputs the following tuple A^ℓ:

$$\{1, \ldots, \ell\} \xrightarrow{\text{syntactic description}} \{1, \ldots, k\}^+ \xrightarrow{\text{substitute } \bar{a}} A^+ \xrightarrow{\text{semigroup operation}} A.$$

Every natural transformation between monomial functors arises this way. To see this, the syntactic description is recovered by using the natural transformation for the free semigroup $A = \{1, \ldots, k\}^+$, and applying it for the identity valuation

$$x \in \{1, \ldots, k\} \quad \mapsto \quad [x] \in \{1, \ldots, k\}^+.$$

The advantage of the syntactic description, which is unique, is that it allows us to define the *copyless restriction*: (*) we say that a syntactic description

$$\alpha : \{1, \ldots, \ell\} \to \{1, \ldots, k\}^+$$

is copyless if every letter from $\{1, \ldots, k\}$ appears in at most one word $\alpha(x)$, and in that word it appears at most once. An equivalent condition can be phrased semantically: (**) if we use the natural transformation in the semigroup $A = \mathbb{N}$, then the corresponding function $\mathbb{N}^k \to \mathbb{N}^\ell$ is non-expansive, i.e. the norm of its output is at most the norm of its input, where the norm of a vector is the sum of its coordinates.

We now define what it means to be copyless for a natural transformation between arbitrary polynomial functors

$$\mathsf{F}A = \coprod_{q \in Q} A^{\dim q} \qquad \mathsf{G}A = \coprod_{p \in P} A^{\dim p},$$

which are not necessarily monomial. Such natural transformations also admit syntactic descriptions: for every input component q, there is some designated output component p, and a natural transformation $A^{\dim q} \to A^{\dim p}$. The set of possible syntactic descriptions is

$$\prod_{q \in Q} \coprod_{p \in P} \dim p \to (\dim q)^+.$$

Again, one can show that all natural transformations arise this way. The natural transformation is called copyless if for every q, the corresponding natural transformation between monomial functors is copyless.

Example 10. Consider the functor $1 \oplus A$ that was discussed in Example 9, and shown to be a polynomial functor. Consider the natural transformation

$$(1 \oplus A) \times (1 \oplus A) \to 1 \oplus A$$

which describes the semigroup operation in the coproduct semigroup $1 \oplus A$. This natural transformation is copyless. □

3.2.3 Views

We now describe a crucial property of the coproduct of semigroups, which is behind the proof of Theorem 3.1. The idea is that an element of a coproduct can be uniquely defined from its views onto the individual coordinates, as defined below. For semigroups $A_1, \ldots, A_n$ and an index $i \in \{1, \ldots, n\}$, define the *i-th view* operation to be the function

$$A_1 \oplus \cdots \oplus A_n \to 1 \oplus A_i$$

obtained by applying the cases construction to the following homomorphisms:

$$\underbrace{A_i \xrightarrow{\text{co-projection}} 1 \oplus A_i}_{\text{homomorphism for } i} \qquad \underbrace{A_j \xrightarrow{!} 1 \xrightarrow{\text{co-projection}} 1 \oplus A_i}_{\text{homomorphism for } j \neq i}.$$

For fixed n and i, this is as a natural transformation. The important property of views is that they give complete information about the coproduct, i.e. if we have all views then we can reconstruct an element of the coproduct; furthermore this reconstruction is copyless. This is stated in the following lemma.

Lemma 3.2 *For every $n \in \{1, 2, \ldots\}$ there is a natural transformation* combine*, which is copyless, such that for every semigroups $A_1, \ldots, A_n$, the following diagram commutes*

$$\begin{array}{ccc} A_1 \oplus \cdots \oplus A_n & \xrightarrow{\text{view}_1 \times \cdots \times \text{view}_n} & (1 \oplus A_1) \times \cdots \times (1 \oplus A_n) \\ \| & \swarrow_{\text{combine}} & \\ A_1 \oplus \cdots \oplus A_n. & & \end{array}$$

Proof
Straightforward. **TODO: wrong (except $n = 2$); formulate a suitable corrected version** □

This lemma seems to contain the essential property of semigroups that makes the construction work. Our theorem will also be true for other algebraic structures in the lemma is true, such as forest algebras. However, the lemma seems to fail for certain algebraic structures, such as groups, even if we allow 1 to be replaced by some fixed finite group. Another example where the lemma seems to fail is the monad of weighted sums of words (i.e. this monad corresponds to weighted automata).

3.2.4 Functorial streaming string transducers

We now describe the last ingredient in our proof, which is a more abstract variant of streaming string transducers (SST) that is described in Definition 3.3.

Before presenting the abstract definition, we discuss how it differs from the usual of SST. The first difference, which is least important, there is some abstract output semigroup A instead of a free semigroup Γ^+; this generalization is only meant to have cleaner notation. The second, and more important, difference is that, instead of having a fixed number of registers, we allow the register structure to be a finite polynomial functor such as

$$\mathsf{R}A = A^3 + A^2 + A^2 + A + 1.$$

The idea is that the register structure already contains the states; with the states corresponding to components in the disjoint union, and with different states using different numbers of registers. The final difference is that the transducer is allowed to have a look at regular properties of the string on both sides of the head: when the head is over some position in the input string, then the way in which the registers are is decided based on some recognizable property of

$$\underbrace{\Sigma^*}_{\substack{\text{before}\\ \text{head}}} \times \underbrace{\Sigma}_{\substack{\text{under}\\ \text{head}}} \times \underbrace{\Sigma^*}_{\substack{\text{after}\\ \text{head}}}.$$

(As usual, the register update must be copyless.) By a recognizable property of the above we mean a function that inputs elements of the above set of triples and outputs elements of some set X, which can be be decomposed as

$$\Sigma^* \times \Sigma \times \Sigma^* \xrightarrow{h \times \text{id} \times h} M \times \Sigma \times M \xrightarrow{f} X$$

for some homomorphism h into a finite monoid and some function f. In a sense, this model has two features that replace states: the disjoint unions in the register structure, and the recognizable property. Each of these features alone would be enough, but for our intended application having both features will give a cleaner construction.

Here is the formal definition of our SST model.

Definition 3.3 *An functorial SST is defined by:*

- *a finite input alphabet Σ;*
- *a (not necessarily finite) output semigroup A;*
- *two a finite polynomial functor $\mathsf{R}A, \mathsf{S}A$, called the* register functor *and the* update functors*, along with three copyless natural transformations*

$$\vdash\colon 1 \to \mathsf{R}A \qquad \delta : \mathsf{R}A \times \mathsf{S}A \to \mathsf{R}A \qquad \underbrace{\dashv\colon \mathsf{R}A \to A}_{\text{can be partial}}$$

- *an* oracle*, which is a recognizable function*

$$o : \Sigma^* \times \Sigma \times \Sigma^* \to \mathsf{S}A.$$

The function computed by a functorial SST is the partial function of type

$$\Sigma^+ \to A$$

that is defined as follows. Consider some input string in Σ^+. The machine moves its head along all input positions, and computes for each one a register valuation in $\mathsf{R}A$. In the initial register valuation corresponding to $i = 0$, when no positions were processed yet, the register valuation is obtained by applying $\vdash$ to the unique element 1. For $i > 0$, the i-th register valuation is obtained by applying δ to the pair which consists of the $(i-1)$-st register valuation and the result of applying the oracle to input string with the i-th position distinguished. Finally, the output of the SST is obtained by applying the output term operation $\dashv$ to the last register valuation.

Lemma 3.4 *A function of type $\Sigma^+ \to A$ is regular, in the standard sense, if and only if it is computed by a functorial SST.*

Proof
A normal SST with states Q and k registers can be seen as a functorial SST with a register functor of the form

$$\mathsf{R}A = \coprod_{q \in Q} (A + 1)^k.$$

The oracle function does not look at any properties of the input string (all appropriate information is remembered in the implicit state from the register functor), and it

simply outputs all elements of the output semigroup that might potentially be used in an update, with sufficient copies

$$\mathsf{S}A = A^\ell$$

to make the function δ copyless.

Consider now the other implication in the lemma, which is the one that we use in this proof. In the case when the register functor is A^k for some k, a functorial SST is a special case of an SST with regular lookahead; and regular lookahead can be eliminated [BC, Lemma 13.6]. To accommodate more general polynomial functors as register functors, we observe that the component in a polynomial functor can be stored in the state of an SST, and the register values can be stored in A^k for sufficiently large k. □

3.3 Proof

We now present proof of the right-to-left implication in Theorem 3.1. Consider some transducer semigroup, with the functor being F and the output transformation being out. We will show that for every finite alphabet Σ, every semigroup A (not necessarily a free semigroup Γ^+), the composition

$$\Sigma^+ \xrightarrow{h} \mathsf{F}A \xrightarrow{\mathrm{out}_A} A$$

is recognized by a functorial SST.

We begin by introducing some notation, which allows us to track which parts of an output come from which parts of an input. The new notation will be explained using the running example of the duplicating function

$$f : \{a, b\}^+ \to \{a, b\}^+$$

which is recognized by the transducer semigroup in which the functor F is the identity, the output mechanism is duplication, and the homomorphism

$$h : \{a, b\}^+ \to \mathsf{F}\{a, b\}^+ = \{a, b\}^+$$

is the identity.

For semigroups $A_1, \ldots, A_n$, define the *factorized output function* to be the function of type

$$\mathsf{F}A_1 \times \cdots \times \mathsf{F}A_n \to A_1 \oplus \cdots \oplus A_n$$

that is obtained by composing the four functions described below

$$\begin{array}{c}
\mathsf{F}A_1 \times \cdots \times \mathsf{F}A_n \\
\downarrow \text{co-projection} \times \cdots \times \text{co-projection} \\
(\mathsf{F}A_1 \oplus \cdots \oplus \mathsf{F}A_n) \times \cdots \times (\mathsf{F}A_1 \oplus \cdots \oplus \mathsf{F}A_n) \\
\downarrow \text{semigroup operation in the co-product} \\
\mathsf{F}A_1 \oplus \cdots \oplus \mathsf{F}A_n \\
\downarrow \mathsf{F}(\text{co-projection}) \oplus \cdots \oplus \mathsf{F}(\text{co-projection}) \\
\mathsf{F}(A_1 \oplus \cdots \oplus A_n) \oplus \cdots \oplus \mathsf{F}(A_1 \oplus \cdots \oplus A_n) \\
\downarrow \text{out}_{A_1 \oplus \cdots \oplus A_n} \text{ or } \ldots \text{ or } \text{out}_{A_1 \oplus \cdots \oplus A_n} \\
A_1 \oplus \cdots \oplus A_n.
\end{array}$$

Let us illustrate the factorized output function on our running example, with

$$A_1 = 1 \qquad A_2 = \{a, b\}^+.$$

If we apply the factorized output function to

$$(1, abbb) \in \mathsf{F}A_1 \times \mathsf{F}A_2 = A_1 \times A_2$$

then the output will be

$$1abbb1abbb.$$

If we fix n, then the factorized output function is a natural in the sense that the following diagram commutes

$$\begin{array}{ccc}
\mathsf{F}A_1 \times \cdots \times \mathsf{F}A_n & \xrightarrow{\text{factorized output}} & A_1 \oplus \cdots \oplus A_n \\
\downarrow \mathsf{F}h_1 \times \cdots \times \mathsf{F}h_n & & \downarrow h_1 \oplus \cdots \oplus h_n \\
\mathsf{F}B_1 \times \cdots \times \mathsf{F}B_n & \xrightarrow[\text{factorized output}]{} & B_1 \oplus \cdots \oplus B_n
\end{array}$$

for every semigroup homomorphisms $h_i : A_i \to B_i$. In other words, the factorized output function is a natural transformation of type

$$\text{Semigroups}^n \;\Rightarrow\; \text{Sets}$$

with the upper functor $(A_1, \ldots, A_n) \mapsto$ underlying set of $\mathsf{F}A_1 \times \cdots \times \mathsf{F}A_n$ and the lower functor $(A_1, \ldots, A_n) \mapsto$ underlying set of $A_1 \oplus \cdots \oplus A_n$.

The reason for naturality is that each of the four steps in the definition of the factorized output is itself a natural transformation, and natural transformations compose.

For input strings $w_1, \ldots, w_n \in \Sigma^+$ let use write

$$\langle w_1| \cdots |w_n\rangle \in \underbrace{A \oplus \cdots \oplus A}_{n \text{ times}}$$

to be the result of first applying h to all the strings, and then applying the factorized output function. By abuse of notation, we allow some, but not all of the input strings to be empty, in which case the appropriate values in the co-product are ignored, but the output type is not changed. In our running example, we have

$$\langle abbbbb|\varepsilon|bbabaaa\rangle = \underbrace{\underline{abbbbb}}_{\substack{\text{in first}\\ \text{copy of}\\ \{a,b\}^+}} \underbrace{\underline{bbabaaa}}_{\substack{\text{in third}\\ \text{copy of}\\ \{a,b\}^+}} \underbrace{\underline{abbbbb}}_{\substack{\text{in first}\\ \text{copy of}\\ \{a,b\}^+}} \underbrace{\underline{bbabaaa}}_{\substack{\text{in third}\\ \text{copy of}\\ \{a,b\}^+}}.$$

We also use a similar notation but with some strings underlined; in the underlined case, to the non-underlined strings we apply h, and to the underlined strings we apply

$$\Sigma^+ \xrightarrow{h} \mathsf{F}A \xrightarrow{\mathsf{F}!} \mathsf{F}1.$$

The following lemma is the key part of our construction. The lemma speaks of a copyless natural transformation of type

$$(1 \oplus A) \times (1 \oplus A) \times (1 \oplus 1) \to 1 \oplus A.$$

By copyless we mean that the natural transformation is copyless if its input and outputs are seen as polynomial functors, in the sense that is explained in Example **??**.

Lemma 3.5 *There exists a copyless natural transformation*

$$\delta : (1 \oplus A) \times (1 \oplus A) \times (1 \oplus 1) \to 1 \oplus A$$

and some function

$$o : \Sigma^* \times \Sigma \times \Sigma^* \to (1 \oplus A) \times (1 \oplus 1)$$

which is recognizable in the sense that was defined in Section 3.2.4, such that for every strings $w, v \in \Sigma^*$ and every letter $a \in \Sigma$ we have

$$\langle wa|\underline{v}\rangle = u(\langle w|\underline{av}\rangle, g(w, a, v)).$$

Proof
We begin with the following claim, which shows that there is no difference if we merge parts before or after applying the factorized output operation.

Claim 3.6 *The following diagram commutes*

$$\begin{array}{ccc} \Sigma^* \times \Sigma \times \Sigma^* & \xrightarrow{(w,a,v)\mapsto\langle w|a|\underline{v}\rangle} & A \oplus A \oplus 1 \\ \downarrow{\scriptstyle (w,a,v)\mapsto(wa,v)} & & \downarrow{\scriptstyle \text{merge first two coordinates}} \\ \Sigma^+ \times \Sigma^* & \xrightarrow[(wa,v)\mapsto\langle wa|\underline{v}\rangle]{} & A \oplus 1 \end{array}$$

Thanks to the above claim, we know that the desired value $\langle wa|\underline{v}\rangle$ is obtained from $\langle w|a|\underline{v}\rangle$ by applying a copyless natural transformation, namely merging the first two coordinates. By Lemma 3.2, we know that $\langle w|a|\underline{v}\rangle$ is obtained by applying some copyless natural transformation to its three views

$$\underbrace{\mathrm{view}_1(\langle w|a|\underline{v}\rangle)}_{\text{in } 1\oplus A} \qquad \underbrace{\mathrm{view}_2(\langle w|a|\underline{v}\rangle)}_{\text{in } 1\oplus A} \qquad \underbrace{\mathrm{view}_3(\langle w|a|\underline{v}\rangle)}_{\text{in } 1\oplus 1}.$$

By the same argument as in Claim 3.6, the three views above are equal to, respectively

$$\langle w|\underline{av}\rangle \qquad \underbrace{\mathrm{merge}(\langle \underline{w}|a|\underline{v}\rangle)}_{\substack{\text{here merging is the operation}\\ \text{that merges the two copies of } 1}} \qquad \langle \underline{wa}|\underline{v}\rangle. \tag{2}$$

Summing up, we have shown that the desired value $\langle wa|\underline{v}\rangle$ is obtained by applying some copyless natural transformation to the three values in (2). To complete the proof of the lemma, we observe that second and third values in (2) depend only on the letter a and the images of w and v under h; $\mathsf{F}!$ and therefore can be obtained from (w, a, v) in a recognizable way. □

Thanks to the above lemma, we can construct a device which is almost a functorial SST as described in Section 3.2.4 and which recognizes our desired function $w \mapsto \langle w\rangle$. We say "almost", because the device will use register and update functors that are infinite polynomial functors; this construction will be later improved so that it becomes finite. The register and update functors are

$$\mathsf{R}A = 1 \oplus A \qquad \mathsf{S}A = (1 \oplus A) \times (1 \oplus 1).$$

As mentioned above, these are not a finite polynomial functors; we will resolve this problem shortly. Beyond that, there construction is immediate: if we take the register update term δ and the oracle o as in the above lemma, and we define the initial register value to be

$$\underbrace{\langle \varepsilon|\underline{a_1 \cdots a_n}\rangle}_{\substack{\text{this does not depend on the input string } a_1\cdots a_n,\\ \text{it is equal to the unique element of } 1\oplus A \text{ that does not use } A}}$$

then the resulting machine will have the property that if the input string is $a_1 \cdots a_n$, then its i-th configuration is

$$\langle a_1 \cdots a_i|\underline{a_{i+1} \cdots a_n}\rangle.$$

In particular, the last configuration is $\langle a_1 \cdots a_n|\varepsilon\rangle$, which is the same as the output when viewed as an element of $A \oplus 1$. Therefore, to get the output, we should apply the partial copyless natural transformation of type $A \oplus 1 \to A$ which is the one-sided inverse of the embedding of A in $A \oplus 1$.

The only remaining issue with our construction is that the two functors, i.e. the register functor and the update functors, are not finite polynomial functors. This

problem is overcome by observing that not all of $1 \oplus A$ need be used for the register values, only a small part of it, and likewise for the update functor. More formally, consider the natural isomorphism

$$1 \oplus A \quad \simeq \quad \underbrace{\coprod_{q \in 1 \oplus 1} A^{\dim q}}_{\text{call this the } \textit{register representation} \text{ of } 1 \oplus A}$$

that was discussed in Example **??**. If we apply this isomorphism to an element of the form

$$\langle w | \underline{v} \rangle \in A \oplus 1,$$

then the corresponding component will be $\langle \underline{w} | \underline{v} \rangle$. Since the latter depends only on $\underline{w}$ and $\underline{v}$, and these take values in the finite semigroup $\mathsf{F}1$, it follows that there are only finitely many components of $1 \oplus A$ that will be used to represent values from of the form $\langle w | \underline{v} \rangle$. Therefore, instead of using $\mathsf{R}A$ to be all of $A \oplus 1$, we can restrict it to those finitely many components, giving thus a finite polynomial functor. The same argument can be applied to the update functor $\mathsf{S}A$.

References

[AČ10] 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.

[AFR14] Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *Computer Science Logic and Logic in Computer Science, CSL-LICS 2014, Vienna, Austria,*, pages 1–10. ACM, 2014.

[BC] Mikołaj Bojańczyk and Wojciech Czerwiński. An Automata Toolbox.

[BDK18] 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.

[BS20] Mikołaj Bojańczyk and Rafał Stefański. Single-Use Automata and Transducers for Infinite Alphabets. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113:1–113:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO Definable String Transductions and Two-way Finite-state Transducers. *ACM Trans. Comput. Logic*, 2(2):216–254, 2001.

[She59] 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.