A categorical perspective on regular functions

Mikołaj Bojańczyk, Lê Thành Dũng Nguyễn

February 3, 2023

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 regular string-to-string functions. This is a fundamental class of functions, which has many equivalent descriptions: deterministic two-way automata [**?**, Note 4], copyless streaming string transducers (SST) [**?**, Section 3] (or the earlier and very similar single use restricted macro tree transducers [**?**, Section 5]), MSO transductions [**?**, Theorem 13], combinators [**?**, Section 2], a functional programming language [**?**, Section 6], λ-calculus with linear types [**?**, Theorem 3] (see also [**?**, Claim 6.2] and [**?**, Theorem 1.2.3]), decompositions *à la* Krohn–Rhodes [**?**, Theorem 18, item 4], etc.

The present paper adds a new characterization to the list, which uses minimal syntax, and refers only to basic concepts from algebra and category theory. We prove that a string-to-string function of type $\Sigma^* \to \Gamma^*$ is regular if and only if it can be decomposed as

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

where F is some finiteness-preserving functor from the category of semigroups to itself, h is some semigroup homomorphism, and the output function out_{Γ^*} is a natural transformation.

Tito — oops, forgot to specify that out goes from semigroups to sets

This result (thm:regular-functions) 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 warm-up theorems

In this section, we define the model that is studied in this paper, namely transducer semigroups. The purpose of this model is to recognize *string-to-string* functions, which are defined to be functions of type $\Sigma^* \to \Gamma^*$, for some finite alphabets. Some results will work in the slightly more general case where the input or output is a semigroup that is not necessarily a finitely generated free monoid, but we focus on the string-to-string case for the sake of concreteness.

The model is defined using terminology based in category theory. However, we do not assume that the reader has a background in category theory, beyond the two most basic notions of category and functor. Recall that a *category* consists of objects with morphisms between them, such that the morphisms can be composed and each object has an identity morphism to itself. 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.

Recall that a functor between to categories consists of two maps: one map assigns to each object A in the source category a new object in the target category, and another map assigns to each morphism $f : A \to B$ a morhpism $\mathsf{F}f : \mathsf{F}A \to \mathsf{F}B$. These maps need to be consistent with composition of morphisms, and the identity must go to the identity. An 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. The forgetful functor is an example of a semigroup-to-set functor, which goes from the category of semigroups to the category of sets.

Here are some examples of semigroup-to-semigroup functors, which can be seen as semigroup constructions.

Tuples. This functor maps a semigroup A to its square A^2, with the semigroup operation 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. This functor maps a semigroup A to the semigroup where the underlying set is the same, but multiplication is reversed, i.e. the product of a and b in the new semigroup is the product b and a in the old semigroup. Morphisms are not changed by the functor: they retain the homomorphism property despite the change in the multiplication operation.

Non-empty lists. This functor maps a semigroup A to the free semigroup A^+ which consists of non-empty lists (or strings) over the alphabet A equipped with concatenation. On morphisms, the functor is defined element-wise (or letter-wise).

Powerset. This (covariant) powerset functor maps A to the powerset semigroup $\mathsf{P}A$, whose underlying set is the family of all subsets of A, endowed with the operation

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

Variants of the powerset functor require the subsets to be nonempty, or finite, or both.

We now present the central definition of this paper.

A *transducer semigroup* consists of a semigroup-to-semigroup functor F, together with an *output mechanism*, which associates to each semigroup A a function of type $\mathsf{F}A \to A$, called the *output function for* A. The output function does not need to be a semigroup homomorphism. The output mechanism is required to be *natural*, which means that the diagram

$$\begin{tikzcd}
\mathsf{F}A \arrow[r, "\mathsf{F}h"] \arrow[d, "\text{output function for } A"'] & \mathsf{F}B \arrow[d, "\text{output function for } B"] \\
A \arrow[r, "h"'] & B
\end{tikzcd}$$

commutes for every semigroup homomorphism $h : A \to B$.

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

$$A \xrightarrow{\quad h \quad} \mathsf{F}B \xrightarrow{\text{output function for } B} B \qquad \text{for some semigroup homomorphism } h.$$

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

$$\begin{tikzcd}
\text{Semigroups} \arrow[r, bend left, "\text{apply } \mathsf{F} \text{ and return underlying set}"] \arrow[r, bend right, "\text{return underlying set}"'] \arrow[r, phantom, "\Downarrow"] & \text{Sets.}
\end{tikzcd}$$

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 from A to B.

Consider the transducer semigroup in which the functor is the identity, and the output function for A is $a \in A \mapsto aa \in A$. (This output function is not a semigroup homomorphism.) The functions of type $A \to B$ that are recognized by this transducer semigroup are exactly those of the form $a \mapsto h(a)h(a)$ where h is some homomorphism. In particular, if h is the identity on the monoid Σ^*, which is also a semigroup, then we get the duplicating function on strings over the alphabet Σ.

Consider the reversing functor from ex:functors. Define the output mechanism to be the identity. Using this transducer semigroup, we can recognize the string reversal function.

Consider the functor $A \mapsto A^*$, which is similar to the nonempty list functor from Example **??**, except that it allows empty lists, and consider an output function

$$[a_1, \ldots, a_n] \in A^+ \mapsto \underbrace{(a_1 \cdots a_n) \cdots (a_1 \cdots a_n)}_{n \text{ times}} \in A.$$

This transducer semigroup recognizes the squaring function $w \in \Sigma^* \mapsto w^{|w|} \in \Sigma^*$ that is illustrated in the following example: 123 $\mapsto$ 123123123.

2.1 Two simple characterizations

We begin with two simple theorems, which use transducer semigroups to describe two classes of string-to-string functions: all functions (Theorem **??**) and functions that reflect recognizability (Theorem **??**). In Section **??**, 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.

Every string-to-string function is recognized by some transducer semigroup. We prove a slightly stronger result, namely that every function between two semigroups A and B is recognized by some transducer semigroup. Consider some semigroup A. We define a transducer semigroup that recognizes all functions from A to every other semigroup. The functor is defined by

$\mathsf{F}B = A \times$ (set of all functions of type $A \to B$, not necessarily recognizable).

The semigroup operation in $\mathsf{F}B$ is defined as follows: on the first coordinate, we inherit the operation from A, while on the second coordinate, we use the trivial *left zero* semigroup structure, which means that the multiplication of two functions is simply the first one (this is a trivial way of equipping every set with a semigroup structure). The functor is defined on morphisms as in the tuple construction from Example **??**: 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 easily seen to be a functor. The output mechanism, which is easily seen to be natural, is function application i.e. $(a, f) \mapsto f(a)$. Every function $f : A \to B$ is recognized by this transducer semigroup. The appropriate homomorphism is $a \in A \mapsto (a, f)$.

Recognizability reflecting functions. We now characterize functions which have the property that inverse images of recognizable languages are also recognizable. We use a slightly more general setup, where instead of languages we use functions into finite sets (languages can be seen as the special case of functions into a two-element set). We say that a function from a possibly infinite semigroup A to some finite set X is *recognizable* if it factors through some semigroup homomorphism from A to some finite semigroup. A function $f : B \to A$ between semigroups, not necessarily a semigroup homomorphism, is called *recognizability reflecting* if for every recognizable function $g : A \to X$, the composition $g \circ f$ is recognizable.

The following example shows that there are many recognizability reflecting functions.

[Factorials] Consider the semigroup $(\mathbb{N}, +)$ of natural numbers with addition. In this semigroup, the recognizable languages are ultimately periodic subsets. A corollary is that every recognizable language gives the same answer to all factorials $\{1!, 2!, \ldots\}$, with finitely many exceptions. Take any function $f : \mathbb{N} \to \mathbb{N}$ such that (a) every output number arises from at most finitely many input numbers; (b) every output number is a factorial. The composition of f with any recognizable language will be a language that gives the same answer to all numbers with finitely many exceptions; such languages are necessarily recognizable. A function with condition (a) and (b) can be chosen in uncountably many ways, even if we require that it has linear growth.

In light of the above example, there are too many recognizability reflecting functions to allow a machin model, or some other effective syntax. A (non-effective) syntax is given in the following theorem, which is proved the same way as Theorem **??**.

The following conditions are equivalent for a string-to-string function:

1. it is recognizability reflecting.
2. it is recognized by a transducer semigroup such that for every finite semigroup C, the corresponding output function of type $\mathsf{F}C \to C$ is recognizable.

3 The regular functions

The two straightforward constructions in Theorems **??** and **??** amount to little more than symbol pushing. In this section, we present a more advanced characterization, which is the main result of this paper.

In Theorem **??**, the condition on the output mechanism is that if a semigroup A is finite, then the output function $\mathrm{out}_A : \mathsf{F}A \to A$ is recognizable. In this section, we strengthen the conclusion to saying that functor is *finiteness preserving*, which means that if A is finite, then the same is true for $\mathsf{F}A$. This implies that the output mechanism $\mathsf{F}A \to A$ is recognizable, since every function from a finite semigroup is recognizable. However, the condition is strictly stronger, as witnessed by Example **??**, which is recognizability reflecting but not finiteness preserving. As we will see, the stronger condition will characterize exactly the regular string-to-string functions.

The following example shows that naturality of the output mechanism interacts with the condition that the functor is finiteness preserving in a non-trivial way.

Consider the powerset functor $\mathsf{P}A$ from Example **??**. This is a finiteness preserving functor, since the powerset of a finite semigroup is also finite. One could imagine that using powersets, we could construct a transducer semigroup that recognizes functions that are not regular, e.g. because they have exponential growth (unlike regular functions, which have linear growth). It turns out that this is impossible, because there is no possible output mechanism, i.e. no natural transformation of type $\mathsf{P}A \to A$, as we explain below.

The issue is that the naturality condition disallows choosing elements from a subset. To see why, consider a semigroup A with two elements, with the left zero semigroup operation defined by $ab = a$. For this semigroup, the output mechanism of type $\mathsf{P}A \to A$ would need to choose some element $a \in A$ when given as input the full set $A \in \mathsf{P}A$. However, none of the two choices is right, because swapping the two elements of A is an automorphism of the semigroup A, which maps the full set to itself, but does not map any element to itself.

We now state the main theorem of this paper.

The following conditions are equivalent for every string-to-string function:

1. it regular, i.e. recognized by a streaming string transducer;
2. it is recognized by a transducer semigroup in which the functor is finiteness preserving.

Here is the plan for the rest of this section:

Section ?? gives a formal definition of regular functions

Section ?? proves the easy implication in the theorem, namely (**??**) $\Rightarrow$ (**??**)

Section ?? proves the hard implication in the theorem, namely (**??**) $\Leftarrow$ (**??**)

Before continuing, we remark on one advantage of the characterization, namely a straightforward proof of closure under composition. This is in contrast with some other models, where closure under composition is a non-trivial construction, such as two-way transducers [**?**, Theorem 2] or copyless SST [**?**, Theorem 1]. (For other models, such as MSO transductions, closure under composition is straightforward.)

To see why the functions from item (**??**) in Theorem **??** are closed under composition, consider the following diagram, where the upper path describes the composition of two functions recognized transducer semigroups $(\mathsf{F}, \mathrm{out})$ and $(\mathsf{F}', \mathrm{out}')$, respectively:

$$\begin{tikzcd}
\Sigma^* \arrow[r, "h"] & \mathsf{F}(\Gamma^*) \arrow[r, "\mathrm{out}_{\Gamma^*}"] \arrow[d, "\mathsf{F}h'"] & \Gamma^* \arrow[d, "h'"] & \\
& \mathsf{F}\mathsf{F}'(\Pi^*) \arrow[r, "\mathrm{out}_{\mathsf{F}'(\Pi^*)}"] & \mathsf{F}'(\Pi^*) \arrow[r, "\mathrm{out}'_{(\Pi^*)}"] & \Pi^*
\end{tikzcd}$$

The rectangle in the middle commutes by naturality, and therefore the upper path is equal to the lower path. The lower path corresponds to a transducer semigroup that is obtained by composing the two functors F and F', and the two corresponding output mechanisms.

3.1 Defininition of streaming string transducers

In this section, we formally describe the regular functions, using a model based on streaming string transducers. We will prove Theorem **??** in a slightly more general case, namely for string-to-semigroup functions instead of only string-to-string functions. Here, a string-to-semigroup function is any function of type $\Sigma^* \to A$ where Σ

is a finite alphabet and A is an arbitrary semigroup. This will make notation easier, since the fact that the output semigroup consists of strings will not play any role in our proof. To handle string-to-semigroup functions, we work with a mild extension of streaming string transducers, in which the inputs are strings over a finite alphabet, but the output is an abstract semigroup A.

The model is based on registers which store elements of some semigroup, so we begin by describing notation for registers and their updates. Suppose that R is a finite set of *register names*, and A is a semigroup called the *output semigroup*. We consider two sets

$$\underbrace{R \to A}_{\text{the set of } \textit{register valuations}} \qquad\qquad \underbrace{R \to (A+R)^+.}_{\text{the set of } \textit{register updates}}$$

The main operation on these sets is *application*

$$v \in \text{register valuations} \quad u \in \text{register updates} \quad \mapsto \quad vu \in \text{register valuations},$$

which substitutes all register names in the register update u with their values in the register valuation v. A register update is called *copyless* if, after concatenating all right hand sides (theser are the strings in the image of the update), the resulting string contains each every register name at most once.

In our model of streaming string tranducers, the registers will be updated by a stream of register updates that is produced by a rational function, defined as follows. Intuitively speaking, a rational function corresponds to an automaton that produces one output letter for each input position, with the output letter depending on regular properties of the input position within the input string. More formally, a *rational function*, is a length-preserving string-to-string function which has the following property: there is a recognizable function

$$f : (\{\text{current, not current}\} \times (\text{input alphabet}))^+ \to \text{output alphabet},$$

such that for every input string, the i-th output letter is obtained by applying the function to the string that is obtained from the input string by setting the first coordinate to "current" for the i-th position, and "not current" for the remaining positions. In a rational function, the output of label of the i-th position is allowed to depend on letters of the input string that are to the right of the i-th input position; this corresponds to regular lookahead in a streaming string transducer. Such lookahead can be eliminated without affecting the expressive power, see [**?**, Section 12.3], but allowing lookahead will be convenient later in the proof.

Having defined register updates and rational functions, we are ready to define the variant of streaming string transducers that is used in this paper.

The syntax of a streaming string transducer is given by:

- A finite *input alphabet* Σ and an *output semigroup* A.
- A finite set R of *register names*. All register valuations and updates below use R and A.

- A designated *initial register valuation*, and a designated *final register.*
- An update oracle, which is a rational letter-to-letter function of type

$$\Sigma^* \to (\text{copyless register updates})^*.$$

The semantics of the transducer is a function of type $\Sigma^* \to A$ that is defined as follows. When given an input string, the transducer begins in the designated initial register valuation. Next, it applies all updates produced by the update oracle, in left-to-right order. Finally, the output of the transducer is obtained by returning the semigroup element stored in the designated final register.

The model described above is easily seen to be equivalent to streaming string transducers with regular look-ahead, which are one of the equivalent models defining the regular string-to-string functions, see [**?**, Section 12].

3.2 From a regular function to a transducer semigroup

Having defined the transducer model, we prove the easy implication in thm:regular-functions. Suppose that a function $f : \Sigma^* \to A$ is computed by some streaming string transducer. In the proof below, when refering to register valuations and register updates, we refer to those that use the registers and output semigroup of the fixed transducer. We say that a register update is in *normal form* if, in every right hand side, one cannot find two consecutive letters from the semigroup A. Every register update can be normalized, i.e. converted into one that is in normal form, by using the semigroup operation to merge consecutive elements of the output semigroup in right hand sides. The register updates before and after normalization act in the same way on register valuations. The crucial property of being copyless is that if a register update is copyless and in normal form, then the combined length of all right hand sides is at most three times the number of registers. Therefore, if a semigroup is finite, then the set of register updates in normal form – call this set $\mathsf{S}A$ – is also finite. (This would not be true for register updates that are not copyless.) The set $\mathsf{S}A$ of register updates in normal form can be equipped with a composition operation

$$u_1, u_2 \in \mathsf{S}A \quad \mapsto \quad u_1 u_2 \in \mathsf{S}A,$$

which is defined in the same way as applying a register update to a register valuation, except that we normalize at the end. This composition operation is associative, and compatible with applying register updates to register valuations, in the sense that $(vu_1)u_2 = v(u_1u_2)$ holds for every register valuation v and register updates u_1 and u_2. Therefore, $A \mapsto \mathsf{S}A$ is semigroup construction, which is finiteness preserving. We can also extend S to morphisms, i.e. view it as a functor from semigroups to semigroups, by applying a semigroup homomorphism to every semigroup element that appears in an update.

We define below a transducer semigroup which uses a functor F that is based on the functor S and the update oracle of the streaming string transducer defining f. Let

$$h : (\{\text{current, not current}\} \times \Sigma)^* \to B,$$

be a homomorphism into a finite semigroup such that the i-th letter produced by the update oracle depends only on the result of applying this homomorphism to the string obtained from the input in the way that was described in the definition of rational functions. Without loss of generality we assume that B is a monoid. The semigroup-to-semigroup functor F is defined as follows. If the input semigroup is A, then the underlying set of the output semigroup $\mathsf{F}A$ is

$$B \quad \times \quad \underbrace{(B \times B) \to \mathsf{S}A}_{\substack{\text{functions of this kind} \\ \text{are called \textit{conditional}} \\ \text{\textit{register updates}}}} \quad \times \quad \underbrace{R \to A}_{\substack{\text{register} \\ \text{valuations}}}.$$

The semigroup operation is defined as follows. On the third coordinate, we use the trivial left zero semigroup structure. On the first two coordinates, the semigroup structure is defined[1] so that the product of two pairs (b_1, φ_1) and (b_2, φ_2) is the pair consisting of $b_1 b_2$ and the function

$$(c_1, c_2) \mapsto \varphi_1(b_1, c_2 b_2) \cdot \varphi_2(b_1 c_1, b_2).$$

The construction F is extended to morphisms in the same way as S.

We now define the output mechanism. When given $(b, \varphi, v) \in \mathsf{F}A$, the output mechanism returns the element of the semigroup A that is obtained as follows: (1) apply φ to the pair consisting of the neutral elements in the monoid B, yielding a register update in $\mathsf{S}A$; then (2) apply this register update to the register valuation v, yielding some new register valuation; and then (3) from the resulting register valuation, return the semigroup element stored in the distinguished output register. Checking the naturality condition is left to the reader.

Using the transducer semigroup defined above, we can recognize the function computed by our streaming string transducer.

3.3 From a transducer semigroup to a regular function

We now turn to the difficult implication (**??**) $\Rightarrow$ (**??**) in Theorem **??**. The assumption of the implication uses an abstract model (transducer semigroups), while the conclusion uses a concrete operational model (streaming string transducers). To bridge the gap, we will use an intermediate model, which is similar to streaming string transducers, but a bit more abstract. The abstraction will be obtained by using polynomial functors instead of registers, as described in Section **??**.

3.3.1 Functorial streaming string transducers

Define a *polynomial functor* to be a semigroup-to-set functor of the form

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

[1] This definition coincides with the two-sided semidirect product of monoids from [**?**, Section 6], when applied to the monoids B and $\mathsf{S}A$.

where Q is some possibly infinite set, called the *components*, with each component having an associated *dimension* in $\{0, 1, \ldots\}$. The symbol $\coprod$ stands for disjoint union of sets. This functor does not take into account the semigroup structure of the input semigroup, since the output is seen only as a set. On morphisms, the functor works in the expected way, i.e. coordinate-wise.

A *finite polynomial functor* is one that has finitely many components. A finite polynomial functor can be seen as a mild generalization of the construction which maps a semigroup A to the set A^R of register valuations for some fixed set R of register names. In the generalization, we allow a variable number of registers, depending on some finite information (the component).

Having defined a more abstract notion of "register valuations", we now define a more abstract notion of "register updates". The first condition for such updates is that they do not look inside the register contents; this condition is captured by naturality as described in the following definition.

[Natural functions] Let F and G be polynomial functors, let A be a semigroup. A function[2] $f : \mathsf{F}A \to \mathsf{G}A$ is called *natural* if it can be extended to natural transformation of type $\mathsf{F} \Rightarrow \mathsf{G}$. This means that there is a family of functions, with one function

$$f_A : \mathsf{F}A \to \mathsf{G}A$$

for each semigroup A, such that $f = f_A$, and the the diagram

$$\begin{tikzcd}
\mathsf{F}A \arrow[r, "\mathsf{F}h"] \arrow[d, "f_A"'] & \mathsf{F}B \arrow[d, "f_B"] \\
\mathsf{G}A \arrow[r, "h"] & \mathsf{G}B
\end{tikzcd}$$

commutes for every semigroup homomorphism h.

Consider the polynomial functors

$$\mathsf{F}A = A^* = \coprod_{q \in \mathbb{N}} A^q \qquad \mathsf{G}A = A + 1,$$

where 1 represents the singleton set A^0. An example of a natural transformation between these two functors is the function which maps a nonempty list in A^* to the product of its elements, and which maps the empty list to the unique element of 1. A non-example is the function that returns the leftmost element in the input list that is an idempotent in the semigroup, and returns 1 if such an element does not exist. The reason why the non-example is not natural is that a semigroup homomorphism can map a non-idempotent to an idempotent.

Apart from naturality, we will want our register updates to be copyless. This can be formalized in several ways; we choose to use the following semantic definition.

[2] This function is not necessarily a semigroup homomorphism. In fact, it would not even make sense call it a homomorphism, since the functors F and G produce sets and not semigroups.

[Copyless natural function] A natural function $f : \mathsf{F}A \to \mathsf{G}A$ is called *copyless* if it arises from some natural transformation with the following property: when instantiated to the semigroup[3] $(\mathbb{N}, +)$, the corresponding function of type $\mathsf{F}\mathbb{N} \to \mathsf{G}\mathbb{N}$ does not increase the norm. Here, the norm of an element in a polynomial functor $\mathsf{F}\mathbb{N}$ or $\mathsf{G}\mathbb{N}$ is defined to be the sum of numbers that appear in it.

Having defined functions that are natural and copyless, we now describe the more abstract model of streaming string transducers that will be used in our proof. The main difference is that instead of register valuations and updates that are given by some finite set of register names, we have two abstract polynomial functors, together with an explicitly given application function. Another minor difference is that we allow the model to define partial functions; this will be useful in the proof.

The syntax of a functorial streaming string transducer is given by:

- A finite *input alphabet* Σ and an *output semigroup* A.
- Two finite polynomial functor R and U, called the *register* and *update* functors, together with an *application function* of type $\mathsf{R}A \times \mathsf{U}A \to \mathsf{R}A$, which is natural and copyless.
- A distinguished *initial register valuation* in $\mathsf{R}A$.
- A *final function* of type $\mathsf{R}A \to A + 1$, which is natural and copyless.
- An *update oracle*, which is a rational function of type $\Sigma^* \to (\mathsf{R}A)^*$.

The semantics of the transducer is a partial function of type $\Sigma^* \to A$ that is defined as follows. As in Definition **??**, for every input string we use the initial register valuation, the application function and the update oracle to define a sequence of register valuations in $\mathsf{F}A$. We then apply the final function to the last register valuation, yielding a result in $A+1$. If this result is in the 1 part, then the output of the transducer is undefined, otherwise the output of the transducer is the semigroup element stored in the A part. We will care about transducers that compute total functions, which corresponds to the property that for every input string, the last register valuation is in the A part of $A + 1$.

The models defined in Definition **??** and **??** define the same (total) string-to-semigroup functions. An SST as in Definition **??** can be seen as a special case of an SST as in Definition **??**, because the sets of register valuations and register updates are constructed using finite polynomial functors, and the application operation is natural and copyless.

(TODO complete)

[3] The choice of the semigroup $(\mathbb{N}, +)$ in the def:copyless is not particularly important. For example, the same notion of copylessness would arise if instead of $(\mathbb{N}, +)$, we used the semigroup $\{0, 1\}$ with addition up to threshold 1 (i.e. the only way to get zero is to add two zeros). In the appendix, we present a more syntactic characterization of copyless natural transformations, which will be used later on when proving equivalence with streaming string transducers.

3.3.2 Coproducts and views

Apart from the more abstract transducer model from Definition **??**, the other ingredient used in the proof of the hard implication in Theorem **??** will be coproducts of semigroups, and some basic operations on them, as described in this section.

We write 1 for the semigroup that has one element. This semigroup is unique up to isomorphism and it is a *terminal object* in the category of semigroups, which means that it admits a unique homomorphism from every other semigroup A. This unique homomorphism will be denoted by $! : A \to 1$. It has no connection with the factorial function on numbers.

The *coproduct* of two semigroups A and B, which is denoted by $A \oplus B$, is the semigroup that is defined as follows. Elements of this semigroup are nonempty words over an alphabet that is the disjoint union of A and B, restricted to words that are *alternating* in the sense that two consecutive letters cannot belong to the same semigroup. The semigroup operation is defined in the expected way. We draw elements of a coproduct using coloured boxes:

$$\boxed{aba} \cdot \boxed{b} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{abba} .$$

The picture above shows an element of the coproduct of two copies of the semigroup $\{a, b\}^+$, with the copies distinguished using the colours red and blue. One can also have a coproduct of more than two semigroups; in the pictures this would correspond to more colours for the boxes, subject to the condition that every two consecutive boxes have different colours.

The polynomial functors that we use in our proof will arise using coproducts with the singleton semigroup 1. Consider the semigroup-to-set functor $A \mapsto A \oplus 1$, which maps a semigroup to the underlying set of its coproduct with the singleton semigroup. Although not defined as a polynomial functor, this functor is isomorphic to a polynomial functor. This is because for every semigroup A there is a bijective correspondence between the sets

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

where the dimension of q is defined to be the number of times that the first copy of 1 appears in q. Furthermore, the bijection in (**??**) is natural, and therefore there is a natural bijection between the functor $(-) \oplus 1$ 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 (**??**) is unique up to renaming of components. By uniqueness, we will simply speak of $A \oplus 1$ as being a polynomial functor. In a similar way, functors such as $A \mapsto A \oplus 1 \oplus A$ are also polynomial.

The crucial property of semigroups that will be used in our proof is described in Lemma **??** below, which says that a coproduct can be reconstructed based on certain partial information. This partial information is described using the following operations on coproducts.

1. **Merging**. Consider a coproduct $A_1 \oplus \cdots \oplus A_n$, such that the same semigroup A appears on all coordinates from a subset $I \subseteq \{1, \ldots, n\}$, and possibly on other coordinates as well. Define *merging the parts from* I to be the function of type

$$A_1 \oplus \cdots \oplus A_n \to A \oplus \bigoplus_{i \notin I} A_i$$

that is defined in the expected way, and explaine in the following picture. In the picture, merging is applied to a coproduct of three copies of the semigroup $\{a, b\}^+$, indicated using colours red, black and blue, and the merged coordinates are red and blue:

$$\boxed{aba} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{abba} \cdot \boxed{b} \quad \mapsto \quad \underbrace{\boxed{abab} \cdot \boxed{aa} \cdot \boxed{baaabba} \cdot \boxed{b}}_{\text{the merge of red and blue is drawn in violet}}.$$

2. **Shape.** Define the *shape operation* to be the function of type

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

obtained by applying ! on every coordinate. The shape says how many alternating blocks there are, and which semigroups they come from, as explained in the following picture:

$$\boxed{aba} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{abba} \cdot \boxed{b} \quad \mapsto \quad \boxed{1} \cdot \boxed{1} \cdot \boxed{1} \cdot \boxed{1} \cdot \boxed{1} \cdot \boxed{1} \cdot \boxed{1}.$$

3. **Views.** The final operation is the i-th view

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

This operation applies ! to all coordinates other than i, and then it merges all those coordinates. Here is a picture, in which we take the view of the blue coordinate:

$$\boxed{aba} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{b} \cdot \boxed{aa} \cdot \boxed{abba} \cdot \boxed{b} \quad \mapsto \quad \boxed{aba} \cdot \boxed{1} \cdot \boxed{aa} \cdot \boxed{1}.$$

The key observation is that an element of a coproduct can be reconstructed from its shape and views, as stated in the following lemma.

Let $A_1, \ldots, A_n$ be semigroups. The function of type

$$A_1 \oplus \cdots \oplus A_n \to (1 \oplus A_1) \times \cdots \times (1 \oplus A_n) \times (1 \oplus \cdots \oplus 1),$$

which is obtained by combining the views for all $i \in \{1, \ldots, n\}$ and the shape, is injective. The input can be reconstructed from the output as follows. Start with the shape, and replace the entries from 1 with the semigroup elements used in the views.

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 for which 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).

Tito — I wonder if the important thing is not more simply that $A \oplus B$ is a polynomial bifunctor

Interesting!

3.4 Proof of thm:regular-functions

We have now collected all necessary ingredients to prove the implication (**??**) ⇒ (**??**) in Theorem **??**. Consider some transducer semigroup, with the functor being F, and fix a string-to-semigroup function that is recognized by this transducer semigroup, i.e. a function $f : \Sigma^* \to A$ which is a composition of some semigroup homomorphism $h : \Sigma^* \to \mathsf{F}A$ followed by the output mechanism of the transducer semigroup. We will show that the function f is computed by some functorial SST as in Definition **??**.

The main idea behind the proof is that, using coproducts, we will be able to identify the origin semantics of the function f, which means that we will now which parts of the input string are responsible for which parts of the output semigroup. This will be done using coproducts, as described below.

For semigroups $A_1, \ldots, A_n$, define the *vectorial output mechanism* to be the function

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

that is obtained by composing the three functions described below

$$\begin{tikzcd}
\mathsf{F}A_1 \times \cdots \times \mathsf{F}A_n \arrow[d, "\mathsf{F}(\text{co-projection}) \times \cdots \times \mathsf{F}(\text{co-projection})"] \\
\mathsf{F}(A_1 \oplus \cdots \oplus A_n) \times \cdots \times \mathsf{F}(A_1 \oplus \cdots \oplus A_n) \arrow[d, "\text{semigroup operation}"] \\
\mathsf{F}(A_1 \oplus \cdots \oplus A_n) \arrow[d, "\text{output mechamism for } A_1 \oplus \cdots \oplus A_n"] \\
A_1 \oplus \cdots \oplus A_n .
\end{tikzcd}$$

To illustrate the definitions in this section, we use the transducer semigroup from ex:duplicator for the duplicating functions. In this transducer semigroup, the functor is the identity $\mathsf{F}A = A$, and the output mechanism is $a \mapsto aa$. The duplicating function on $\{a, b\}^*$ is obtained by composing the identity homomorphism

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

with the output mechanism. Here is an example of the vectorial output mechanism, with the two semigroups being 1 and $\{a, b\}^*$:

$$(1, abbb) \in \mathsf{F}1 \times \mathsf{F}\{a, b\}^* \qquad \mapsto \qquad \boxed{1}\,\boxed{abbb}\,\boxed{1}\,\boxed{abbb} \in 1 \oplus \{a, b\}^* .$$

The vectorial output mechanism is natural, in the sense that the diagram

$$\begin{tikzcd}
\mathsf{F}A_1 \times \cdots \times \mathsf{F}A_n \arrow[r, "\text{factorized output}"] \arrow[d, "\mathsf{F}h_1 \times \cdots \times \mathsf{F}h_n"'] & A_1 \oplus \cdots \oplus A_n \arrow[d, "h_1 \oplus \cdots \oplus h_n"] \\
\mathsf{F}B_1 \times \cdots \times \mathsf{F}B_n \arrow[r, "\text{factorized output}"'] & B_1 \oplus \cdots \oplus B_n
\end{tikzcd}$$

commutes for every semigroup homomorphisms $h_1, \ldots, h_n$. This is because each of the three steps in the definition of the vectorial output mechanism is itself a natural transformation, and natural transformations compose. Naturality of the first two steps is easy to check, while for the last step we use the assumption that the (non-vectorial) output mechanism is natural.

Using the vectorial output mechanism, we will be able to track the origins in the output of the function f, with respect to some partition of the input string into several nonempty parts. For strings $w_1, \ldots, w_n \in \Sigma^*$, define the corresponding *factorized output*, denoted by

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

to be the result of first applying h to all the strings, then applying the factorized output function, and finally removing the elements of the output co-product that correspond to input coordinates $i \in \{1, \ldots, n\}$ in which the string w_i was the empty string ε.

In our running example, we have

$$\langle abbbbb | \varepsilon | bbabaaa \rangle = \boxed{abbbbb}\,\boxed{bbabaaa}\,\boxed{abbbbb}\,\boxed{bbabaaa} \in \{a,b\}^+ \oplus \{a,b\}^* \oplus \{a,b\}^*.$$

Here, we use colours to distinguish which of the three parts of the input is used; the empty middle part has black colour which is not used in the output.

We also use a similar notation but with some strings underlined. In the underlined case, before applying the vectorial output mechanism, we use h for the non-underlined strings we apply h, and

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

for the underlined strings. (As before, the empty input strings are removed from the output.) In our running example, we have

$$\langle \underline{abbbbb} | \varepsilon | bbabaaa \rangle = \boxed{1}\,\boxed{bbabaaa}\,\boxed{1}\,\boxed{bbabaaa}.$$

The following lemma is the key part of our construction. As discussed in Section **??**, we consider $A \mapsto A \oplus 1$ and $A \mapsto 1 \oplus A \oplus 1$ as a polynomial semigroup-to-set functors, which enables us to talk about natural and copyless functions that operate on them. There is a copyless natural function

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

such that every strings $w, v \in \Sigma^*$ and letter $a \in \Sigma$, one obtains $\langle wa|\underline{v}\rangle$ by applying δ to the pair consisting of $\langle w|\underline{av}\rangle$ and $\langle \underline{w}|a|\underline{v}\rangle$. We use the following claim, which is proved using naturality of the output mechanism. $\langle wa|\underline{v}\rangle$ is obtained from $\langle w|a|\underline{v}\rangle$ by merging the first two parts.

Since merging the first two parts is a copyless natural function, the above claim shows that the factorized output $\langle wa|\underline{v}\rangle$ is obtained from $\langle w|a|\underline{v}\rangle$ by a copyless natural function. To complete the proof of the lemma, we will show that latter value $\langle w|a|\underline{v}\rangle$ can also be obtained by applying some copyless natural function to the pair consisting of $\langle wa|\underline{v}\rangle$ and $\langle \underline{w}|a|\underline{v}\rangle$. This will be done using (an extension of) Lemma **??**. Consider the function of type

$$A \oplus A \oplus 1 \to \underbrace{(1 \oplus A)}_{\substack{\text{first}\\ \text{view}}} \times \underbrace{(1 \oplus A)}_{\substack{\text{second}\\ \text{view}}} \times \underbrace{(1 \oplus 1)}_{\substack{\text{third}\\ \text{view}}} \times \underbrace{(1 \oplus 1 \oplus 1)}_{\text{shape}},$$

which is the injective function from Lemma **??** in the special case of the coproduct $A \oplus A \oplus 1$. We use the name *deconstruction* for this function. By the same proof as in Lemma **??**, this function is not only injective, but it also has a one-sided inverse, i.e a function of type

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

which we call *reconstruction*, such that deconstruction followed by reconstruction is the identity on $A \oplus A \oplus 1$. Furthermore, reconstruction is natural and copyless.

By the above observations, one can obtain the factorized output $\langle w|a|\underline{v}\rangle$ by applying reconstruction to the following four items (the equalities below are proved using Claim **??**):

1. First view of $\langle w|a|\underline{v}\rangle$, which is equal to $\langle w|\underline{av}\rangle$.

2. Second view of $\langle w|a|\underline{v}\rangle$, which is obtained by merging the first and third parts in $\langle \underline{w}|a|\underline{v}\rangle$.

3. Third view of $\langle w|a|\underline{v}\rangle$, which is equal to $\langle \underline{wa}|\underline{v}\rangle$.

4. Shape of $\langle w|a|\underline{v}\rangle$, which is equal to $\langle \underline{w}|\underline{a}|\underline{v}\rangle$.

To complete the proof of the lemma, it remains to justify that the last three items can be obtained from $\langle \underline{w}|a|\underline{v}\rangle$ by applying some copyless natural function. Each item is obtained separately by applying a natural function. Furthermore, the second item is obtained in a copyless way, while the last two items do not use A at all, and therefore they are obtained in a copyless way for trivial reasons, even when combined with the second item.

Using the above lemma, we can design a device that recognizes our desired function $w \mapsto \langle w\rangle = f(w)$, and which is almost a functorial SST as in Definition **??**. 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 the (infinite) polynomial functors

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

As mentioned above, these are not a finite polynomial functors; we will resolve this problem shortly. Beyond that, the construction is immediate. Consider an input string $a_1 \cdots a_n$. The device begins its computatin with the initial register value

$$\langle \varepsilon | \underline{a_1 \cdots a_n} \rangle \in A \oplus 1.$$

This value does not depend on the input string, since it is always equal to the unique element of $1 \oplus A$ that does not use A. The rational function in the transducer is defined so that the i-th lettter of its output string is

$$\langle \underline{a_1 \cdots a_{i-1}} | a_i | \underline{a_{i+1} \cdots a_n} \rangle \in 1 \oplus A \oplus 1$$

We will explain shortly how these letters can be computed by a rational function. Thanks to Lemma **??**, after applying all the register updates produced by this rational function to the initial register valuation, the register valuation at the end is

$$\langle a_1 \cdots a_n | \varepsilon \rangle \in A \oplus 1,$$

which is the same as the output when viewed as an element of $A \oplus 1$, as required in Definition **??** for representing the output of a partial function.

We are left with proving that the update oracle is a rational letter-to-letter function, and resolving the issue that the two functors R and S are not finite polynomial functors.

To see why the update oracle is a rational letter-to-letter function, we observe that

$$\langle \underline{a_1 \cdots a_{i-1}} | a_i | \underline{a_{i+1} \cdots a_n} \rangle \in 1 \oplus A \oplus 1$$

depends only the letter a_i, as well as the images of the words $a_1 \cdots a_{i-1}$ and $a_{i+1} \cdots a_n$ under the semigroup homomorphism obtained by composing h with $\mathsf{F}! : \mathsf{F}A \to \mathsf{F}1$. Since the target semigroup $\mathsf{F}1$ of this homomorphism is a finite, by the assumption that the functor is finiteness preserving, it follows that the update oracle is a rational letter-to-letter function.

We now explain how to turn R and S into finite polynomial functors. The key observation is 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 bijection

$$A \oplus 1 \quad \cong \quad \coprod_{q \in 1 \oplus 1} A^{\dim q}$$

that was discussed in Section **??**. If we apply this bijection to a factorized output $\langle w | \underline{v} \rangle \in A \oplus 1$, 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 $A \oplus 1$ 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 applies to the update functor $\mathsf{S}A$.