Submitted to MFPS 2024

Implicit automata in λ-calculi III: affine planar string-to-string functions[⋆]

Cécilia Pradic[a,1] Ian Price[a,2]

[a] *Department of Computer Science*
Swansea University
Wales

Abstract

We prove a characterization of first-order string-to-string transduction via λ-terms typed in non-commutative affine logic that compute with Church encoding, extending the analogous known characterization of star-free languages. We show that every first-order transduction can be computed by a λ-term using a known Krohn-Rhodes-style decomposition lemma. The converse direction is given by compiling λ-terms into two-way reversible planar transducers. The soundness of this translation involves showing that the transition functions of those transducers live in a monoidal closed category of diagrams in which we can interpret purely affine λ-terms. One challenge is that the unit of the tensor of the category in question is not a terminal object. As a result, our interpretation does not identify β-equivalent terms, but it does turn β-reductions into inequalities in a poset-enrichment of the category of diagrams.

Keywords: non-commutative linear logic, transducers, λ-calculus, automata theory, Church encodings

1 Introduction

The first author and Nguyễn initiated a series of work that compares the expressiveness of *simply-typed affine λ-calculi* (in the sense of linear logic) and *finite-state machine* from automata theory in [28]. This endeavour is very much in the spirit of *implicit computational complexity*, a field where one attempts to capture complexity-theoretic classes of functions (rather than automata-theoretic) via various typed programming languages, whence our borrowing of the term "implicit".

The starting point was to refine Hillebrand and Kanellakis' theorem [19, Theorem 3.4] that states that the simply-typed λ-calculus captures regular languages when computing over Church encodings. Then, it was shown that one can also characterize *star-free languages* via the non-commutative affine λ-calculus ($\lambda\wp$) [28]. $\lambda\wp$ features a function type which constrains arguments to be used at most once and "in order", which restrains the available power. It was conjectured that, when it comes to affine string-to-string functions, $\lambda\wp$ compute exactly *first-order transductions* and its commutative variant the larger class of *regular transductions* [28]. The latter was proven in follow-up work [26,24] and the main contribution of this paper is to tackle the former, extending and generalizing [28, Theorem 1.7].

⋆ We thank Aurore Alcolei, Lê Thành Dũng Nguyễn and Arno Pauly for discussions about this work.

1 Email: c.pradic@swansea.ac.uk
2 Email: 2274761@swansea.ac.uk

Theorem 1.1 *Affine string-to-string $\lambda\wp$ definable functions and first-order string transductions coincide.*

That every first-order transduction is $\lambda\wp$-definable follows from a decomposition lemma that states that all such transductions are a composition of elementary building blocks that can be coded in $\lambda\wp$. Most of this coding was already done in [28, Theorem 4.1]. The more interesting direction is the converse, which is proven using a semantic evaluation argument to compile $\lambda\wp$-definitions into *two-way planar reversible finite transducers* (2PRFTs), a variant of two-way transducers that were recently shown to capture exactly first-order transductions [27]. The semantics in question targets a non-symmetric monoidal-closed category $\mathsf{TransDiag}_\Gamma$ in which transitions of 2PRFTs find a natural home as morphisms.

Much like other semantic evaluation arguments like Hillebrand and Kanellakis' or in higher-order model checking [17,18], a nice aspect is that automata will be computed in a very straightforward way from terms once things are set up, and this computation will even be polynomial-time here provided we are given a normal term. However, one difficulty we are going to run into will have to do with the fact that our calculus is not linear but *affine* and that $\mathsf{TransDiag}_\Gamma$ does not have a terminal object. We will still be able to manage to use it as an interpretation target for $\lambda\wp$ by noticing that it carries a Poset-enriched structure and showing that this is enough to have an interpretation[3] of terms $[\![-]\!]$ such that the $[\![t_{\mathrm{NF}}]\!] \leq [\![t]\!]$ if t evaluates to t_{NF} via β-reduction.

Plan of the paper

In Section 2, we review the standard notions concerning Poset-enriched categories and the non-commutative λ-calculus we will require. We then explain in Section 3 what it means for a string-to-string function to be $\lambda\wp$-definable and what 2PRFTs are. The latter we take as an opportunity to introduce early $\mathsf{TransDiag}_\Gamma$ and define transitions of 2PRFTs as morphisms in those categories. In Section 4, we prove Theorem 1.1. Finally, we conclude with some observations concerning the commutative case and tree transductions that follow from our work in Section 4 before evoking some further research directions that could most probably build on the material presented here.

Related work

For a more comprehensive overview of "implicit automata in λ-calculi", one may consult the introductions of [26,24]. Regarding this paper more specifically, the other most relevant works are the one leading up to the introduction of 2PRFTs in [27], which mostly comes from Hines' suggestion in [20], which itself drew on Girard's geometry of interaction programme [14] and Temperley-Lieb algebras [1,11]. We use categorical automata in the sense of Colcombet and Petrişan [8] for practical purposes similar to [26]. While categorical frameworks are used to give generic results for various classes of automata by, e.g., viewing them as algebras [2,4,16], as coalgebras [30] or as dependent lenses [32], here we simply use a categorical definition of 2PRFTs so that it may be easily be related to the semantics of the $\lambda\wp$-calculus. In particular, we will focus on the categories $\mathsf{TransDiag}_\Gamma$ (for Γ ranging over alphabets) and no other categories for most of the paper. While we are not aware of a source that defines exactly $\mathsf{TransDiag}_\Gamma$, it is likely close matches exist in the literature as it admits a straightforward inductive presentation. A similar construction is the operad of spliced words in [22, Example 1.2], where the more general operad of spliced contours [22, Definition 1.1] is used to analyze and generalize the Chomsky-Schützenberger representation theorem.

2 Background

2.1 Categorical preliminaries

In the rest of this subsection, we list the key definitions related to Poset-enriched strict monoidal categories. In particular, we specialize the definitions from general enriched category [21] to the Poset-enriched case for the convenience of the reader.

For notation, we use $\circ$ for composition, but also ; for composition written in the reverse order ($f;g = g \circ f$) when it is more convenient. We write id_A for the identity at object A and $[A,B]_{\mathcal{C}}$ for the set of

[3] We suspect this can be characterized as an initiality theorem stating that there is a minimal oplax strong monoidal-closed functor from initial affine monoidal-closed categories to Poset-enriched monoidal-closed categories, but we leave this characterization, which would require dealing with tensors in the $\lambda\wp$-calculus, for future work.

morphisms of $\mathcal{C}$ with domain A and codomain B. When the ambient category is clear from context or Set, we sometimes write $f : A \to B$ to mean that f is a morphism from A to B.

Definition 2.1 A *category* $\mathcal{C}$ is said to be *Poset-enriched* if it is enriched in the category of posets and monotone functions, i.e., if for every objects A and B, $[A, B]_{\mathcal{C}}$ is a partially ordered set and composition $[B, C]_{\mathcal{C}} \times [A, B]_{\mathcal{C}} \to [A, C]_{\mathcal{C}}$ is monotone with respect to the product ordering on $[B, C]_{\mathcal{C}} \times [A, B]_{\mathcal{C}}$.

A *functor* $T : \mathcal{C} \to \mathcal{D}$ between Poset-enriched categories is *Poset-enriched* if it is enriched in the category of posets and monotone functions, i.e., for any objects A and B of $\mathcal{C}$, $T_{A,B} : [A, B]_{\mathcal{C}} \to [T(A), T(B)]_{\mathcal{D}}$ is monotone. A *Poset-enriched natural transformation* between Poset-enriched functors is just a natural transformation.

Definition 2.2 A (Poset-enriched) category $\mathcal{C}$ is strict monoidal when we have an (enriched) functor $\otimes : \mathcal{C}^2 \to \mathcal{C}$ and an object $\mathbf{I}$ such that $(\otimes, \mathbf{I})$ and $(\otimes, \mathrm{id}_{\mathbf{I}})$ induce monoid structures on the objects and morphisms of $\mathcal{C}$.

Note that we did not include a symmetry $A \otimes B \cong B \otimes A$ in our definition of monoidal. Although the coming definitions also make sense for non-strict monoidal categories, throughout the rest of the paper, we will consider strict monoidal categories only.

Definition 2.3 A (Poset-enriched) monoidal category $(\mathcal{C}, \mathbf{I}, \otimes)$ is *closed* if for each object X of $\mathcal{C}$, the (enriched) functor $(- \otimes X) : \mathcal{C} \to \mathcal{C}$ has an (enriched) right adjoint $(X \multimap -) : \mathcal{C} \to \mathcal{C}$, i.e., for any triple of objects X, Y, Z we have an natural isomorphism $\Lambda_{X,Y,Z} : [X \otimes Y, Z]_{\mathcal{C}} \cong [X, Y \multimap Z]_{\mathcal{C}}$ which is monotone. We will write $\mathrm{ev}_{Y,Z}$ for the counit of the adjunction [4].

As we are also interested in categories with a dualising structure, it would be natural to ask for an (enriched) compact-closed category. However, to the author's knowledge, there is no clear consensus on the "correct" definition of compact-closed category when the tensor is not symmetric. One such candidate, a restricted version of pivotal category, was put forward by Freyd & Yetter [12] and is appropriate to our needs. The following definitions come from Selinger's survey of graphical languages [31].

Definition 2.4 In a monoidal category, an *exact pairing* between two objects A and B, is given by a pair of maps $\eta : \mathbf{I} \to B \otimes A, \varepsilon : \mathbf{I} \to A \otimes B$, called respectively *cups* and *caps*, such that the following two triangles commute [5]:

$$\begin{array}{ccc} A & \xrightarrow{\mathrm{id}_A \otimes \eta} & A \otimes B \otimes A \\ & \searrow^{\mathrm{id}_A} & \downarrow{\scriptstyle \varepsilon \otimes \mathrm{id}_A} \\ & & A \end{array} \qquad \begin{array}{ccc} B & \xrightarrow{\eta \otimes \mathrm{id}_B} & B \otimes A \otimes B \\ & \searrow^{\mathrm{id}_B} & \downarrow{\scriptstyle \mathrm{id}_B \otimes \varepsilon} \\ & & B \end{array}$$

In an exact pairing, B is called the *right dual* of A and A is called the *left dual of B*.

Definition 2.5 A monoidal category is *left (resp. right) autonomous* if every object A has a left (resp. right) dual, which we denote ${}^*A$ (resp A^*). It is *autonomous* if it is both left and right autonomous.

Any choice of duals A^* and cups and caps ε_A, η_A for every object A in a left autonomous category $\mathcal{C}$ extends $(-)^*$ to a functor $\mathcal{C} \to \mathcal{C}^{\mathrm{op}}$ by setting $f^* = (\eta_A \otimes \mathrm{id}_{B^*}); (\mathrm{id}_{A^*} \otimes f \otimes \mathrm{id}_{B^*}); (\mathrm{id}_{A^*} \otimes \varepsilon_B)$ when $f : A \to B$. We then also have that the chosen cups and caps are natural transformations. Similar definitions can be made for right autonomous categories.

Definition 2.6 A *pivotal category* is a right autonomous category equipped with a monoidal natural transformation $i_A : A \to A^{**}$. We are primarily interested in the case where i_A is the identity, in which case, we refer to it as a *(strict)* pivotal category.

The following lemma shows that pivotal categories allow us treat left and right duals as the same and define closure in terms of duals.

[4] It is equal to $\Lambda^{-1}_{Y,Y,Z}(\mathrm{id}_{Y \multimap Z}) : (Y \multimap Z) \otimes Y \to Z$ by definition and corresponds to an evaluation morphism $(Y \multimap Z) \otimes Y \to Z$ used to interpret function application.

[5] These equations are typically called the "yanking" or "zigzag" equations.

$$\frac{}{\Psi, x : \tau, \Psi'; \Delta \vdash x : \tau} \qquad \frac{}{\Psi; \Delta, x : \tau, \Delta' \vdash x : \tau}$$

$$\frac{\Psi; \Delta, x : \tau \vdash t : \sigma}{\Psi; \Delta \vdash \lambda x.t : \tau \multimap \sigma} \qquad \frac{\Psi; \Delta \vdash t : \tau \multimap \sigma \qquad \Psi; \Delta' \vdash u : \tau}{\Psi; \Delta, \Delta' \vdash t\, u : \sigma}$$

$$\frac{\Psi, x : \tau; \Delta \vdash t : \sigma}{\Psi; \Delta \vdash \lambda^! x.t : \tau \to \sigma} \qquad \frac{\Psi; \Delta \vdash t : \tau \to \sigma \qquad \Psi; \cdot \vdash u : \tau}{\Psi; \Delta \vdash t\, u : \sigma}$$

Figure 1. $\lambda\wp$ and type derivations. The contexts Ψ and Δ are lists of pairs $x : \tau$ containing a variable name x and some type τ. We assume that all variables appearing in a context and under binders are pairwise distinct and that terms and derivations are defined up to α-renaming.

Lemma 2.7 *Pivotal categories are autonomous and closed.*

Proof Since $A^{**} \cong A$ and A^{**} is the right dual of A^*, it follows that A^* is also left dual of A.

To show monoidal closure, define the functor $(B \multimap -) := (- \otimes B^*)$. We can construct prove the adjunction by setting $\Lambda_{A,B,C}(f) = (\mathrm{id}_A \otimes \eta_B); (f \otimes \mathrm{id}_{B^*})$, which has inverse $\Lambda^{-1}_{A,B,C}(g) = (g \otimes \mathrm{id}_B); (\mathrm{id}_C \otimes \varepsilon_B)$. That Λ and Λ^{-1} are inverse is provable thanks to the yanking equations.

□

2.2 The planar λ-calculus λ℘

For most of the paper, we will be working in the non-commutative fragment of the affine λ-calculus that we call $\lambda\wp$. *Types* of $\lambda\wp$, that we typically write with the greek letter τ, σ and κ, are inductively generated by a designated base type $\mathbb{o}$ and two type constructors $\multimap$ and $\to$ corresponding respectively to *affine* and unrestricted function types. We will have the following restrictions for the function spaces built with $\multimap$:

- arguments must be used at most once
- arguments must occur in order in application.

We introduce both the syntax and the typing rules of $\lambda\wp$ (which, in particular, enforce those restrictions) in Figure 1. Throughout, we formally need to manipulate term that come with their type derivations rather than raw terms, but we will often simply write out terms rather than typing judgement for legibility. We call the fragment where types do not contain the non-affine arrow $\to$ *purely affine*.

To make those term compute, we define the capture-avoiding substitution of x by a term u in t by $u[t/x]$ as usual, as well as the relation $\to_\beta$ of *β-reduction* as being the least relation containing satisfying $(\lambda x.\, t)\; u \to_\beta t[u/x]$ for all well-typed expressions (of the same type) and being closed by congruence. Call $\to_\beta^*$ its reflexive transitive closure. An expression of shape $(\lambda x.\, t)\; u$ is called a *β-redex* and a term containing no such redex is called *normal*. *η-equality* The least congruence containing all clauses $t =_\eta \lambda x.\, t\, x$ for every t which has a function type is called η-equivalence. Two terms are called $\beta\eta$-equivalent if they can be related by the least equivalence relation containing $\to_\beta$ and $=_\eta$.

Every rewriting sequence involving $\to_\beta$ and well-typed terms terminates.

Proposition 2.8 (standard argument, see also [28, Proposition 2.3]) *For every* $\Psi;\ \Delta \vdash t : \tau$, *there is a normal term* $\Psi;\ \Delta \vdash t_{\mathrm{NF}} : \tau$ *such* $t \to_\beta^* t_{\mathrm{NF}}$.

3 First-order string-to-string transductions in the planar affine λ-calculus

3.1 Definable string-to-string functions in the planar affine λ-calculus

In order to discuss string functions in $\lambda\wp$, we need to discuss how they are encoded. For that, we use the same framework as in [23,24]. In the pure (i.e. untyped) λ-calculus and its polymorphic typed variants such as System F, the canonical way to encode inductive types is via *Church encodings*. Such encodings are typable in the simply-typed λ-calculus by dropping the prenex universal quantification that comes with them in polymorphic calculi. For instance, for natural numbers and strings over $\{a, b\}$, writing Church(w) for the Church encoding of w, we have $\text{Church}(aab) = \lambda a.\lambda b.\lambda\epsilon.\ \underline{aab} = \lambda a.\lambda b.\lambda\epsilon.\ a\ (a\ (b\ \epsilon))$.

Conversely, a consequence of normalization is that any closed simply typed λ-term "of type string" is $\beta\eta$-equivalent to the Church encoding of some string. In the rest of this paper, we use a type for Church encodings of strings that is finer than usual and not expressible without $\multimap$, first introduced in [13, §5.3.3].

Definition 3.1 Let Σ be an alphabet. We define Str_Σ as $\underbrace{(\mathbb{o} \multimap \mathbb{o}) \to \ldots \to (\mathbb{o} \multimap \mathbb{o})}_{|\Sigma| \text{ times}} \to \mathbb{o} \to \mathbb{o}$.

Definition 3.2 Given an alphabet $\Sigma = \{a_1, \ldots, a_n\}$, define the signature $\underline{\Sigma}$ as $a_1 : \mathbb{o} \multimap \mathbb{o}, \ldots, a_n : \mathbb{o} \multimap \mathbb{o}, \epsilon : \mathbb{o}$. For every word $w \in \Sigma^*$ define the typed term $\underline{\Sigma};\ \cdot \vdash \underline{w} : \mathbb{o}$ and the closed term $\text{Church}(w) : \text{Str}_\Sigma$ by

$$\underline{\epsilon} = \epsilon \qquad \underline{a_i w'} = a_i\ \underline{w} \qquad \text{and} \qquad \text{Church}(w) = \lambda a_1 \ldots \lambda a_n.\lambda\epsilon.\ \underline{w}$$

We can then show the following by inspecting the normal form and using Proposition 2.8.

Lemma 3.3 *For every $\Psi;\ \underline{\Sigma} \vdash t : \mathbb{o}$, t is $\beta\eta$-equivalent to a unique $\underline{w_t}$ and, a fortiori, for every $\Psi;\ \underline{\Sigma} \vdash u : \mathbb{o}$, u is $\beta\eta$-equivalent to a unique* $\text{Church}(w_u)$.

As a consequence, $\lambda\wp$ terms of type $\text{Str}_\Sigma \to \text{Str}_\Gamma$ correspond to functions $\Sigma^* \to \Gamma^*$, but have a limited expressivity. We consider a natural extension of these by allowing to emulate a limited kind of polymorphism via *type substitutions* $\tau[\kappa]$ defined as follows.

$$\mathbb{o}[\kappa] = \kappa \qquad \text{and} \qquad (\tau \multimap \sigma)[\kappa] = \tau[\kappa] \multimap \sigma[\kappa]$$

Type substitutions extends in the obvious way to typing contexts, and even to *typing derivations*, so that $\Psi;\ \Delta \vdash t : \tau$ entails $\Psi[\kappa];\ \Delta[\kappa] \vdash t : \tau[\kappa]$. In particular, it means that a Church encoding $t : \text{Str}_\Sigma$ is also of type $\text{Str}_\Sigma[\kappa]$ for any type κ. This ensures that the following notion of definable string-to-string functions makes sense and is closed under function composition.

Definition 3.4 A function $f : \Sigma^* \to \Gamma^*$ is called *affine $\lambda\wp$-definable* when there exists a *purely affine* type κ together with a λ-term $\mathtt{f} : \text{Str}_\Sigma[\kappa] \multimap \text{Str}_\Gamma$ such that f and $\mathtt{f}$ coincide up to Church encoding; i.e., for every string $t \in \Sigma^*$, $\text{Church}(f(t)) =_{\beta\eta} \mathtt{f}\ \text{Church}(t)$.

Example 3.5 The function $\mathtt{reverse} : \Sigma^* \to \Sigma^*$ that reverses its input is affine $\lambda\wp$-definable. Supposing that we have $\Sigma = \{a_1, \ldots, a_k\}$, one $\lambda\wp$-term that implements it is

$$\lambda s.\lambda a_1.\ldots.\lambda a_k.\lambda\epsilon.\ s\ (\lambda x.\lambda z.x\ (a_1\ z)) \ldots (\lambda x.\ (a_k\ z))\ (\lambda x.x)\ \epsilon\ :\ \text{Str}_\Sigma[\mathbb{o} \multimap \mathbb{o}] \multimap \text{Str}_\Sigma$$

This definition involves terms defined in the full calculus that still requires to work with the $\to$ type constructor that occurs in Str. But we also have an equivalent characterization in terms of purely affine terms. This characterization is obtained by inspecting the normal form of a $\lambda\wp$ definition.

Lemma 3.6 (particular case of [24, Lemma 5.25], easier to prove from Proposition 2.8) *Let $\Sigma = \{a_1, \ldots, a_n\}$ and $\Gamma = \{b_1, \ldots, b_k\}$ be alphabets. Up to $\beta\eta$-equivalence, every term of type $\text{Str}_\Sigma[\kappa] \multimap \text{Str}_\Gamma$ is of the shape $\lambda s.\lambda b_1.\ldots.\lambda b_k.\lambda\epsilon.\ o\ (s\ d_1\ \ldots\ d_n\ d_\epsilon)$ such that o, d_ϵ and the d_is are purely linear $\lambda\wp$-terms with no occurrence of s, that is, terms such as we have typing derivations*

$$\underline{\Gamma}; \cdot \vdash o : \kappa \multimap \mathbb{o} \qquad \underline{\Gamma}; \cdot \vdash d_i : \kappa \multimap \kappa \qquad \underline{\Gamma}; \cdot \vdash d_\epsilon : \kappa$$

This lemma and the fact that $\mathtt{reverse}$ is definable mean that an affine $\lambda\wp$-definable function $\Sigma^* \to \Gamma^*$ can, without loss of generality, be given by a *$\lambda\wp$-transducer*, which we define as follows (see e.g. [29, Definition 2.6] or [26, Definition 3.22] for similar definitions).

Definition 3.7 A *$\lambda\wp$-transducer* with input Σ^* and output Γ^* is given by the following types and terms from the purely affine planar λ-calculus with constants in $\underline{\Gamma}$:

- an iteration type κ,
- for each $a \in \Sigma$, a term $d_a : \kappa \multimap \kappa$ over the signature $\underline{\Gamma}$,
- a term $d_\epsilon : \kappa$
- and a term $o : \kappa \multimap \mathbb{o}$.

The underlying function is then defined by mapping a word $w_0 \dots w_n$ to the word corresponding to the normal form of $o\ (d_{w_n}\ (\dots (d_{w_0}\ d_\epsilon) \dots))$.

3.2 The category of planar diagrams

We will now introduce a category of what we are going to call *planar diagrams*. The idea is that the morphisms may be represented by graphs with (an ordered set of) vertices labelled by polarities $p \in \{-, +\}$ and edges labelled by words over some fixed output alphabet Γ. Also given would be a partition of the vertices into input and outputs, and then the composition would be represented by pasting the diagrams together and concatenating labels, in an order prescribed by the polarities and whether the nodes involved are inputs or outputs. One such diagram is pictured in Figure 2.

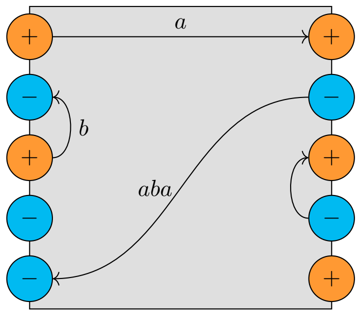


Figure 2. A geometric realization of a morphism from $+ - + - -$ to $+ - + - +$. The edge directions are not part of the definition, but inferred from the polarity labels of the source and targets. When the label is ϵ, we omit it from the picture.

The major restriction that we will put on the diagrams living on our category is that they be *planar*. While we will define these morphisms in a combinatorial way for simplicity, this condition is more intuitive when interpreted geometrically. A geometric interpretation of a diagram can be given by writing out the nodes in order on the boundary of a bounding rectangle (filled in grey in Figure 2), the inputs sitting on the left boundary and outputs on the right boundary, and tracing out the edges within that square. A diagram is then *geometrically planar* when it is possible to do so without making the edges cross.

On the other hand, the combinatorial definition goes as follows.

Definition 3.8 $(V, <, E)$, consisting of an undirected graph (V, E) $(E \subseteq [V]^2)$ and a total order $<$ over V, is called *combinatorially planar* if for every four vertices $a < b < c < d$ then we do not have both edges between a and c and between b and d.

Checking that a combinatorial planar structure can be realized as a geometrical planar structure is relatively straightforward. Proving that conversely a structure with a geometrically planar realization is combinatorially planar can be done using the Jordan curve theorem.

While the diagrams formally do not have a direction, an intended traversal direction is going to be induced by the label of the vertices and whether they are in the input or output sets. More precisely

- if v is an input vertex of polarity $+$ or an output vertex of polarity $-$, then it is an implicit *source* and
- if v is an output vertex of polarity $+$ or an input vertex of polarity $-$, then it is an implicit *target*.

In morphisms, we will restrict edges so that they contain exactly one implicit source and target, so overall they are all orientable. This allows to define the composition $f \circ g$ of two diagrams unambiguously. This can be done for geometrical representations of f and g as follows:

(i) paste the two diagrams together, identifying the output boundary of g with the input boundary of f

(ii) take the new bounding rectangle to be the union of those for f and g; erase the nodes that do not belong to its boundary, as well as the edges that dangle in its interior and loops

(iii) concatenate the labels along the implicit direction of the edges they are labelling

The way we restricted the edges so that they may be oriented makes sure that the last step is well-defined and yields a picture where each interior edge is unambiguously labelled by a word. This process, pictured on an example in Figure 3, can be easily adapted beat-for-beat with the combinatorial definition. However, checking that this yields a diagram which is still planar is more easily done geometrically.

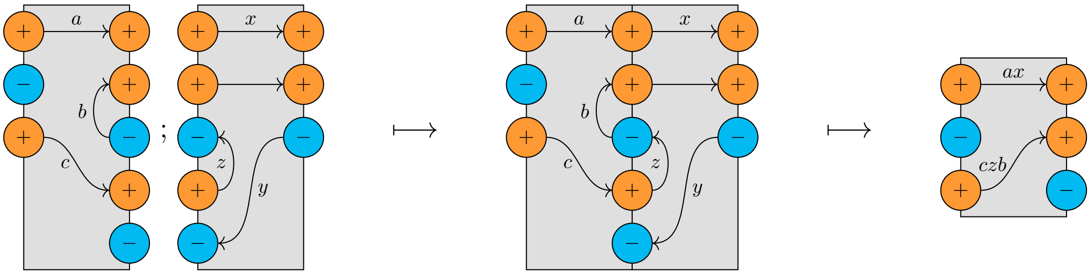


Figure 3. How morphisms compose

Let us summarize what is a legal diagram from a combinatorial standpoint.

Definition 3.9 A *combinatorial planar diagram labelled by a monoid M* is a tuple $(V_{in}, V_{out}, \rho, <, E, \ell)$ where

- V_{in} and V_{out} are disjoint finite sets of vertices
- $<$ is a total order over $V_{in} \cup V_{out}$
- $\rho : V_{in} \cup V_{out} \to \{+, -\}$ assigns polarities to vertices
- E contains subsets of $V_{in} \cup V_{out}$ of size exactly two
- $\ell : E \to M$ assigns labels to edges

subject to the following restrictions, setting $V = V_{in} \cup V_{out}$:

- all vertices in (V, E) must have degree at most one
- $v_{in} < v_{out}$ for every $v_{in} \in V_{in}$ and $v_{out} \in V_{out}$
- $(V, <, E)$ must be planar
- every edge $e \in E$ contains an implicit source as well as a target.

We can now give an official formal definition of categories of diagrams $\mathsf{TransDiag}_\Sigma$ where Σ is going to be the output alphabet. To make the monoidal structure on $\mathsf{TransDiag}_\Sigma$ strict and our lives easier, we will take objects to be words over $\{+, -\}$ rather than labelled sets of inputs and outputs and determine the vertices of the diagrams by positions in the input and output objects.

Definition 3.10 Let Σ be a finite alphabet. The category of planar diagrams over Σ, $\mathsf{TransDiag}_\Sigma$, is defined as follows.

- **Objects** are finite words in $\{+, -\}^*$.
- **Morphisms**, for $A = a_1 \dots a_n$ and $B = b_1 \dots b_m$ a morphism $A \to B$ is a planar combinatorial diagram labelled by Σ^* where:
  - $V_{in} = \{(-1, 1), \dots, (-1, n)\}$
  - $V_{out} = \{(1, 1), \dots, (1, m)\}$
  - $<$ is defined by setting $(i, q) < (j, r)$ if and only if $(i, iq) <_{\text{lex}} (j, jr)$ in the lexicographic order
- **Identities** are given by diagrams where all labels are ϵ and containing all possible edges $\{(-1, k), (1, k)\}$
- **Composition** $h = f; g$ is given by identifying the output vertices $(1, k)$ of g with the input vertices $(-1, k)$ of f and composing the combinatorial diagrams as explained above.

The free monoid structure on objects $\{+,-\}^*$ extends to a strict monoidal structure on $\mathsf{TransDiag}_\Sigma$, i.e., tensoring of objects is concatenation and the unit $\mathbf{I}$ is ϵ. Over morphisms, tensoring can be pictured as putting two diagrams on top of each other as in Figure 4. Note that the planarity of our diagrams mean that this tensor cannot be equipped with a symmetric structure and that $\mathbf{I}$ is not a terminal object.

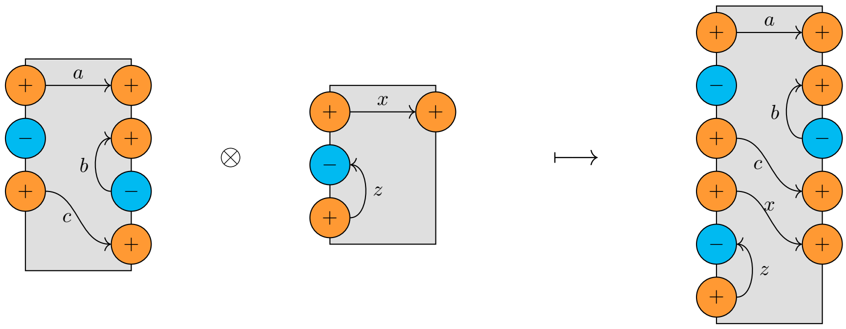
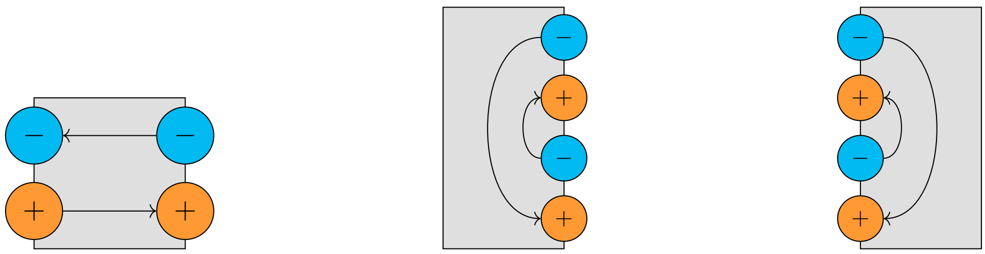


Figure 4. The monoidal product of two morphisms



(a) $\mathrm{id}_{-+} : -+ \to -+$ (b) $\eta_{-+} : \mathbf{I} \to (-+) \otimes (-+)^*$ (c) $\varepsilon_{-+} : (-+)^* \otimes (-+) \to \mathbf{I}$

Figure 5. Identity, cup and cap for the object $-+$

Our category also carries a strict pivotal structure (Definition 2.6). The dual w^* of an object w is obtained by reversing it and flipping the polarities. For instance, $(+--)^*$ is $++-$. Going by this definition, note we also have $(w^*)^* = w$. We also have natural transformations $\eta_A : \mathbf{I} \to A \otimes A^*$ and $\varepsilon_A : A^* \otimes A \to \mathbf{I}$ that we picture in Figure 5. They satisfy the yanking equations, which gives us in particular the closed structure by setting $A \multimap B = B \otimes A^*$, $\mathrm{ev}_{A,B} = \mathrm{id}_B \otimes \varepsilon_A$ and $\Lambda_{A,B,C}(f) = (\mathrm{id}_A \otimes \eta_B);(f \otimes \mathrm{id}_{B^*})$ as per Lemma 2.7.

Finally, observe that we may define a natural order on combinatorial diagrams sharing the same vertices. Given two such diagrams d and d' with respective edge sets E_d and E'_d, we say that $d \le d'$ whenever $E_d \subseteq E'_d$ and their edge labellings coincide over E_d. This gives an order on hom-sets of $\mathsf{TransDiag}_\Sigma$ where composition and tensoring are easily checked to be both monotone. Together with the observation that we have cups and caps that satisfy the yanking equations, we thus have.

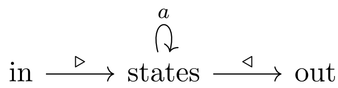
Lemma 3.11 *$\mathsf{TransDiag}_\Sigma$ equipped with the concatenating tensor and inclusion of labelled edges is a strict monoidal-closed poset-enriched category.*

Finally, we note that, for any set of vertices, the bottom element in this order we have defined over diagrams is given by the graph with no edges. Tensoring bottom elements yield bottom elements and $\mathrm{id}_{\mathbf{I}}$ is the bottom element of $[\mathbf{I},\mathbf{I}]_{\mathsf{TransDiag}_\Sigma}$.

3.3 Two-way planar transducers

Following Colcombet and Petrişan [8], we formally define our notion of *two-way planar reversible transducers* (2PRFTs) as being functors whose domain Shape_Σ is category whose morphisms represent infixes of words. In our situation it will mostly have the advantage of concision and making the relationship between 2PRFTs and $\mathsf{TransDiag}$ obvious.

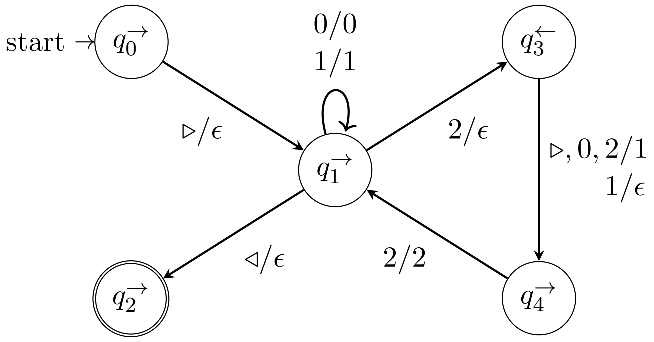
Definition 3.12 For any finite alphabet Σ, there is a three object category Shape_Σ generated by the following finite graph, where there is one morphism for each $a \in \Sigma$.



Morphisms states $\to$ states are identified with words of Σ^* by writing au for $a; u$ and ϵ for $\mathrm{id}_{\text{states}}$ (note that the composition is left-to-right). For any category $\mathcal{C}$ and objects I and O of $\mathcal{C}$, define a $(\mathcal{C}, I, O)$-*automaton with input alphabet* Σ to be a functor $\mathsf{Shape}_\Sigma \to \mathcal{C}$ with $\mathcal{A}(\text{in}) = I$ and $\mathcal{A}(\text{out}) = O$. Given such an automaton $\mathcal{A}$, its semantics is the map $\Sigma^* \to [I, O]_\mathcal{C}$ given by $w \mapsto \mathcal{A}(\triangleright); \mathcal{A}(w); \mathcal{A}(\triangleleft)$.

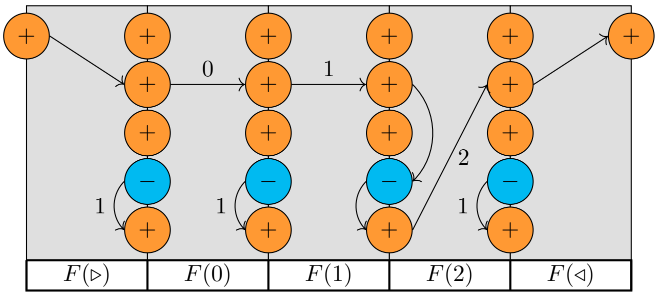
In this framework, we can for instance define deterministics finite automata as $(\mathsf{FinSet}, 1, 2)$-functors and nondeterministic ones as $(\mathsf{FinRel}, 1, 1)$-functors, and check that the semantics computes the languages as we expect it. As the category $\mathsf{TransDiag}_\Sigma$ corresponds to transition profiles as studied in [27], we will use that to define 2PRFTs in an completely equivalent way. In that case, we will pick I and O such that $[I, O]_{\mathsf{TransDiag}_\Sigma} \cong \Sigma^*_\bot$, where $\Sigma^*_\bot$ is the disjoint union of Σ^* with a singleton containing a $\bot$ element; this is required because we will obtain this function by reading off the label of a specific edge of a morphism that may not always exist.

Example 3.13 Let us build a $(\mathsf{TransDiag}_{\{0,1,2\}}, +, +)$-automaton with input alphabet $\{0, 1, 2\}$ that pads any string in $\{0, 1, 2\}^*$ to ensure that every 2 is preceded by a 1 in the output. First, here is a standard automata-theoretic picture of such a device and its transition table:



	$\triangleright$	$\triangleleft$	0	1	2
$q_0^{\to}$	$q_1^{\to}/\triangleright$				
$q_1^{\to}$		$q_2^{\to}/\triangleleft$	$q_1^{\to}/0$	$q_1^{\to}/1$	$q_3^{\leftarrow}/\epsilon$
$q_2^{\to}$					
$q_3^{\leftarrow}$	$q_4^{\to}/1$		$q_4^{\to}/1$	$q_4^{\to}/\epsilon$	$q_4^{\to}/1$
$q_4^{\to}$					$q_1^{\to}/2$

Using the ordering given by the subscripts and assigning $+$ to the forward vertices, i.e., $q_i^{\to}$, and $-$ to the backward vertices, i.e., $q_j^{\leftarrow}$, we obtain the word $F(\text{states}) = + + + - +$. For each letter $a \in \Sigma \sqcup \{\triangleright, \triangleleft\}$ we assign the morphism $F(a)$ of the functor by reading it off the table.



Definition 3.14 A *two-way planar reversible transducer (2PRFT)* $\mathcal{T}$ with input alphabet Σ and output alphabet Γ is a $(\mathsf{TransDiag}_\Gamma, \epsilon, +-)$-automaton with input alphabet Σ.

Writing $\Gamma^*_\bot$ for the disjoint union of Γ^* with a singleton $\{\bot\}$ containing a designated $\bot$ element, the semantics of such a 2PRFT $\mathcal{T}$ induces a function

$$\Sigma^* \xrightarrow{\text{semantics of } \mathcal{T}} [\epsilon, +-]_{\mathsf{TransDiag}_\Gamma} \xrightarrow[\text{($\bot$ if there is no edge)}]{\text{read off the label}} \Gamma^*_\bot$$

$$\frac{x \text{ a variable of } \underline{\Gamma}}{\underline{\Gamma}; \Delta \vdash x : \tau} \quad \longmapsto \quad [\![x]\!] \circ \bot_{[\![\Delta]\!]} : [\![\Delta]\!] \to [\![\tau]\!]$$

$$\frac{}{\underline{\Gamma}; \Delta, x : \tau, \Delta' \vdash x : \tau} \quad \longmapsto \quad \bot_{[\![\Delta]\!]} \otimes \mathrm{id}_{[\![\tau]\!]} \otimes \bot_{[\![\Delta']\!]} : [\![\Delta]\!] \otimes [\![\tau]\!] \otimes [\![\Delta']\!] \to [\![\tau]\!]$$

$$\frac{\underline{\Gamma}; \Delta, x : \tau \vdash t : \sigma}{\underline{\Gamma}; \Delta \vdash \lambda x.t : \tau \multimap \sigma} \quad \longmapsto \quad \frac{[\![t]\!] : [\![\Delta]\!] \otimes [\![\tau]\!] \to [\![\sigma]\!]}{\Lambda_{[\![\Delta]\!],[\![\tau]\!],[\![\sigma]\!]}([\![t]\!]) : [\![\Delta]\!] \to [\![\tau]\!] \multimap [\![\sigma]\!]}$$

$$\frac{\underline{\Gamma}; \Delta \vdash t : \tau \multimap \sigma \qquad \underline{\Gamma}; \Delta' \vdash u : \tau}{\underline{\Gamma}; \Delta, \Delta' \vdash t\ u : \sigma} \quad \longmapsto \quad \frac{[\![t]\!] : [\![\Delta]\!] \to [\![\tau]\!] \multimap [\![\sigma]\!] \qquad [\![u]\!] : [\![\Delta']\!] \to [\![\tau]\!]}{\mathrm{ev}_{[\![\tau]\!],[\![\sigma]\!]} \circ ([\![t]\!] \otimes [\![u]\!]) : [\![\Delta]\!] \otimes [\![\Delta']\!] \to [\![\sigma]\!]}$$

Figure 6. Interpretation of purely affine $\lambda\wp$-terms over $\underline{\Gamma}$ (parameterized by $[\![\mathbb{o}]\!]$ and $[\![x]\!] : \mathbf{I} \to [\![\tau]\!]$ for $x : \tau$ occurring in $\underline{\Gamma}$).

Note that our choice of ϵ and $+-$ and means that by convention, both "initial" and "final" states must occur before the initial and after the final reading of $\triangleleft$, while the convention of [27, Definition 2.1] and in Example 3.13 is slightly different for the initial state. In that version, it should be start reading $\triangleright$, making the 2PRFTs of [27] isomorphic to $(\mathsf{TransDiag}_\Gamma, +, +)$-automata rather than $(\mathsf{TransDiag}_\Gamma, \epsilon, +-)$-automata. But it is not hard to see that both options induce the same class of string-to-string functions. It will turn out that Definition 3.14 matches much more closely λ-transducers, so we favor it out of convenience.

4 Equivalence between planar transducers and $\lambda\wp$ for strings

Now that we have introduced properly our two classes of string-to-string functions, affine $\lambda\wp$-definable functions and first-order transductions, as well as two formalisms that define them, $\lambda\wp$-transducers and 2PRFTs, we will now embark on the proof that they are equivalent.

Theorem 1.1 *Affine string-to-string $\lambda\wp$ definable functions and first-order string transductions coincide.*

To prove that affine $\lambda\wp$-definable functions are first-order transduction, we use the fact that the former class correspond to $\lambda\wp$-transductions and then define a map from $\lambda\wp$-transductions to 2PRFTs that preserves the semantics. To do so, we define an interpretation of purely affine $\lambda\wp$-terms (with duplicable free variables in $\underline{\Gamma}$) in the category $\mathsf{TransDiag}_\Gamma$. One difficulty is that $\mathsf{TransDiag}_\Gamma$ is not *affine* monoidal closed, that is, $\mathbf{I}$ is not a terminal object. So instead of terminal maps we will use $\bot_A \in [A, \mathbf{I}]_{\mathsf{TransDiag}_\Gamma}$ and establish that β-reductions correspond to inequalities in $\mathsf{TransDiag}_\Gamma$ in Subsection 4.1. We will then conclude in Subsection 4.2. Proving the converse, which will amount to a coding exercise and a reference to [28] once the right characterization of first-order transductions as compositions of more basic functions is recalled, will be done in Subsection 4.3.

4.1 Interpreting $\lambda\wp$

All results of this subsection hold for any strict monoidal-closed poset-enriched category $\mathcal{C}$ with a family of least elements $\bot_X \in [X, \mathbf{I}]_\mathcal{C}$ stable under $\otimes$ and with $\bot_\mathbf{I}$, provided we are given an object $[\![\mathbb{o}]\!]$ of $\mathcal{C}$ and, for every constant $x : \tau$ in $\underline{\Gamma}$ a suitable interpretation $[\![x]\!] : \mathbf{I} \to [\![\tau]\!]$, where $[\![\tau]\!]$ is extended inductively over all types by setting for $[\![\tau \multimap \sigma]\!]$ a chosen internal hom $[\![\tau]\!] \multimap [\![\sigma]\!]$. This interpretation also extends to contexts by tensoring as usual by setting $[\![\cdot]\!] = \mathbf{I}$ and $[\![\Delta, x : \tau]\!] = [\![\Delta]\!] \otimes [\![\tau]\!]$. The extension of $[\![-]\!]$ over all purely affine $\lambda\wp$ typing derivations [6] is then given in Figure 6. One thing to note is that the overall interpretation $[\![t]\!]$ of a term t can be carried out in polynomial time in the size of t because type-checking is polynomial-time and composition in $\mathsf{TransDiag}_\Gamma$ can be performed in logarithmic space.

While we will not have that $t =_{\beta\eta} u$ implies $[\![t]\!] = [\![u]\!]$, it will be the case that:

[6] It can actually be shown that the interpretation of a legal typing derivation $\underline{\Gamma}; \Delta \vdash t : \tau$ only depends on the conclusion. But we won't need to make use of that fact.

(a) $[\![a]\!]$ for $a \in \Gamma$ (b) $[\![\epsilon]\!]$

Figure 7. Interpretation of constants as diagrams

- η-expansions $t \to_\eta u$ will be mapped to equalities of morphisms $[\![t]\!] = [\![u]\!]$
- β-reductions $t \to_\beta u$ will be mapped to inequalities $[\![t]\!] \geq [\![u]\!]$

so that, in particular, a normal form t_{NF} of t will always satisfy $[\![t_{\mathrm{NF}}]\!] \leq [\![t]\!]$. Let us now establish that, beginning with η-equivalence.

Lemma 4.1 *When $\underline{\Gamma}; \Delta \vdash t : \tau \multimap \sigma$, we have $[\![\lambda x.\ t\ x]\!] = [\![t]\!]$.*

Proof By definition $[\![\lambda x.\ f\ x]\!] = \Lambda_{[\![\Delta]\!],[\![\tau]\!],[\![\sigma]\!]}(\mathrm{ev}_{[\![\tau]\!],[\![\sigma]\!]} \circ ([\![f]\!] \otimes \mathrm{id}_{[\![\tau]\!]}))$, and the latter is equal to $[\![t]\!]$ by using the universal property of the internal hom. □

Corollary 4.2 *If we have $t \to_\eta u$, then we have that $[\![t]\!] = [\![u]\!]$.*

Proof idea Easy induction using monotonicity of $\circ$ and $\otimes$ together with Lemma 4.1. □

Lemma 4.3 *Suppose we have $\underline{\Gamma}; \Delta, x : \tau, \Delta' \vdash t : \sigma$ and $\underline{\Gamma}; \Delta'' \vdash u : \tau$. Then we have*

$$[\![t[u/x]]\!] \leq [\![t]\!] \circ (\mathrm{id}_{[\![\Delta]\!]} \otimes [\![u]\!] \otimes \mathrm{id}_{[\![\Delta']\!]}) \qquad (\ \ : [\![\Delta, \Delta'', \Delta']\!] \to [\![\tau]\!])$$

Corollary 4.4 *If we have $t \to_\beta u$, then we have that $[\![t]\!] \geq [\![u]\!]$.*

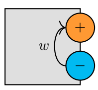
Proof idea Easy induction using monotonicity of $\circ$ and $\otimes$ together with Lemma 4.3. □

We can thus conclude with the only information we will need in the next subsection.

Corollary 4.5 *For any t whose normal form is t_{NF}, we have $[\![t_{\mathrm{NF}}]\!] \leq [\![t]\!]$.*

4.2 From $\lambda\wp$-transducers to 2PRFTs

Now we fix an output alphabet Γ for the $\lambda\wp$-transducer. We shall then use the interpretation from the previous subsection with $\mathcal{C} = \mathsf{TransDiag}_\Gamma$, $[\![\mathbb{o}]\!] = +-$ and the interpretation of the constants of $\underline{\Gamma}$ given in Figure 7.

Lemma 4.6 *For $w \in \Gamma^*$, $\underline{\Gamma}; \cdot \vdash \underline{w} : \mathbb{o}$ is interpreted by the diagram* 

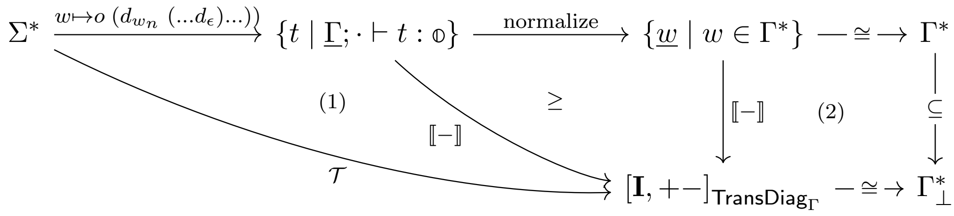
.

Theorem 4.7 *Every $\lambda\wp$-transducer can be converted into an equivalent 2PRFT in polynomial time.*

Proof As per Definition 3.7, assume that we have a purely affine iteration type κ, terms $\underline{\Gamma}; \cdot \vdash d_a : \kappa \multimap \kappa$ for each $a \in \Sigma$, $\underline{\Gamma}; \cdot \vdash o : \kappa \multimap \mathbb{o}$ and $\underline{\Gamma}; \cdot \vdash d_\epsilon : \mathbb{o} \multimap \kappa$. Using the semantic interpretation given above, we obtain morphisms $[\![d_a]\!] : \mathbf{I} \to [\![\kappa]\!] \multimap [\![\kappa]\!]$, $[\![o]\!] : \mathbf{I} \to [\![\kappa]\!] \multimap +-$ $[\![d_\epsilon]\!] : \mathbf{I} \to [\![\kappa]\!]$ respectively. We define the equivalent 2PRFT $\mathcal{T}$ on the generating morphisms of Shape_Σ as follows.

$$\mathcal{T}(a) \quad = \quad \Lambda^{-1}_{\mathbf{I},[\![\kappa]\!],[\![\kappa]\!]}([\![d_a]\!]) \qquad \mathcal{T}(\lhd) \quad = \quad \Lambda^{-1}_{\mathbf{I},[\![\kappa]\!],[\![\mathbb{o}]\!]}([\![o]\!]) \qquad \text{and} \qquad \mathcal{T}(\rhd) = [\![d_\epsilon]\!]$$

To prove that $\mathcal{T}$ compute the same function as the $\lambda\wp$-transducer given, let's consider the diagram below.



By inspecting the definitions, the map defined by the $\lambda\wp$-transducer is obtained by following the topmost maximal path while the map defined by $\mathcal{T}$ is given by the bottommost maximal path, which we must argue define the same map. To do so it actually suffices to show that faces (1) and (2) commute while the central face denotes an inequality between maps; here all nodes are equipped with an order structure by taking the discrete order for the objects on the top row, the order from the enriched structure of $\mathsf{TransDiag}_\Gamma$ for $[\mathbf{I}, +-]_{\mathsf{TransDiag}_\Gamma}$ and by taking for $\Gamma^*_\perp$ the minimal order such that $\perp \leq w$ for $w \in \Gamma^*$. Then the maps are ordered by pointwise ordering. That the inequality "top path $\leq$ bottom path" suffices to derive "top path $=$ bottom path" is because the top path necessarily is a maximal element for the pointwise ordering of maps. This is due to the fact that Γ^* consists of the maximal elements of $\Gamma^*_\perp$.

That (2) commutes is exactly the statement of Lemma 4.6 while the inequality in the central face is Corollary 4.5. All that remains to be proven is that (1) commutes. This is witnessed by the chain of equations below for a fixed input word $w = w_1 \dots w_n \in \Sigma^*$.

$$\begin{aligned}
\mathcal{T}(\triangleright w \triangleleft) &= \mathcal{T}(\triangleleft) \circ \mathcal{T}(w_n) \circ \dots \circ \mathcal{T}(w_1) \circ \mathcal{T}(\triangleright) && \text{(by functoriality)}\\
&= \Lambda^{-1}_{\mathbf{I},[\![\kappa]\!],[\![\mathbb{o}]\!]}([\![o]\!]) \circ \Lambda^{-1}_{\mathbf{I},[\![\kappa]\!],[\![\kappa]\!]}([\![d_{w_n}]\!]) \circ \dots \circ \Lambda^{-1}_{\mathbf{I},[\![\kappa]\!],[\![\kappa]\!]}([\![d_{w_1}]\!]) \circ [\![d_\epsilon]\!] && \text{(by definition of } \mathcal{T})\\
&= \mathrm{ev}_{[\![\kappa]\!],[\![\mathbb{o}]\!]} \circ ([\![o]\!] \otimes \mathrm{id}_{[\![\kappa]\!]}) \circ \mathrm{ev}_{[\![\kappa]\!],[\![\kappa]\!]} \circ ([\![d_{w_n}]\!] \otimes \mathrm{id}_{[\![\kappa]\!]}) \circ \dots \circ \mathrm{ev}_{[\![\kappa]\!],[\![\kappa]\!]} \circ ([\![d_{w_1}]\!] \otimes \mathrm{id}_{[\![\kappa]\!]}) \circ [\![d_\epsilon]\!] \\
& && \text{(because } \Lambda^{-1}_{A,B,C}(f) = \mathrm{ev}_{B,C} \circ (f \otimes \mathrm{id}_B))\\
&= \mathrm{ev}_{[\![\kappa]\!],[\![\mathbb{o}]\!]} \circ ([\![o]\!] \otimes (\mathrm{ev}_{[\![\kappa]\!],[\![\kappa]\!]} \circ ([\![d_{w_n}]\!] \otimes \dots \mathrm{ev}_{[\![\kappa]\!],[\![\kappa]\!]} \circ ([\![d_{w_1}]\!] \otimes [\![d_\epsilon]\!]) \dots))) \\
& && \text{(by functoriality of } \otimes \text{ and } A \otimes \mathbf{I} = A)\\
&= [\![o\ (d_{w_n}\ \dots (d_{w_1}\ d_\epsilon) \dots)]\!] && \text{(by definition of } [\![-]\!])
\end{aligned}$$

□

4.3 From first-order transductions to λ℘

Now we wish to prove the converse direction of Theorem 1.1, that is that every FO-transduction can be encoded in $\lambda\wp$. Much like in [28,27], we rely on the fact that affine $\lambda\wp$-definable string-to-string functions are closed under composition. Using this and the seminal Krohn-Rhodes decomposition theorem, In previous work, it was already shown that affine $\lambda\wp$-definable functions include all *sequential functions* [28, Theorem 5.4]. We thus rely on the same strategy that is used in [27] to show that 2PRFTs compute all first-order transductions.

Lemma 4.8 (rephrasing of [6, Lemma 4.8], see also [27, Lemma 4.3]) *Every first-order transduction can be decomposed as $f \circ$ `reverse` $\circ\, g \circ$ `reverse` $\circ\, h$ where f is computed by a* monotone register transducer *and the functions g and h are aperiodic sequential.*

Example 3.5 already shows that `reverse` is affine $\lambda\wp$-definable. Now it remains to show that fuctions computed by monotone register transducers [3] are affine $\lambda\wp$-definable. Those machines go through their inputs in a single left-to-right pass, storing infixes of their outputs in registers that they may update by performing concatenations of previously stored values and constants. *Monotone* here corresponds to the further restrictions that those machines have no control states, that the output corresponds to a single register and that the register updates satisfy a monotonicity condition in addition to being copyless.

First, let us define the notion of update our machine can use. For simplicity, throughout the rest of this section we assume a fixed output alphabet Γ, disjoint from the set of natural numbers, and a fixed input alphabet Σ.

Definition 4.9 The set of *copyless monotone register update* from n registers to k registers, which we write $\mathrm{RegUp}(n,k)$, is the subset consisting of those k-uples $(w_0,\ldots,w_{k-1})$ of words over $\Gamma \cup \{0,\ldots,n-1\}$ such that:

- every index $i < n$ occurs at most once in the overall tuple (copylessness/affineness)
- if we have that $i \le j < n$ occurring in w_{ℓ_i} and w_{ℓ_j} respectively, then we have either that $\ell_i < \ell_j$, or $\ell_i = \ell_j$ and i occurs before j in w_{ℓ_i}. (monotonicity/planarity)

Given $\sigma \in \mathrm{RegUp}(k,\ell)$ and $\sigma' \in \mathrm{RegUp}(n,k)$, the composition $\sigma \circ \sigma' \in \mathrm{RegUp}(n,\ell)$ is defined by substituting each index $i < k$ in σ by the ith component of σ' (this preserves copylessness and monotonicity).

At the intuitive level, an element of $\mathrm{RegUp}(n,k)$ encodes a function $(\Gamma^*)^n \to (\Gamma^*)^k$ that can operate by concatenating together the components of its inputs, subject to restrictions that match affineness and planarity[7]. For the sequel, write $\pi_\ell \in \mathrm{RegUp}(k,1)$ for $\ell < k$ for the obvious projections, ϵ^k for the updates of $\mathrm{RegUp}(0,k)$ that initialize every register with the empty word and RegContent for the canonical isomorphism $\mathrm{RegContent} : \mathrm{RegUp}(0,1) \cong \Gamma^*$. With this in hand, we give a working definition of monotone register transducers.

Definition 4.10 A *monotone register transducer* consists of the following:

- a number of register n
- for each input letter $a \in \Sigma$, a copyless monotone register update $\sigma_a : x^n \to x^n$.

It computes the function
$$\begin{array}{ccc} \Sigma^* & \longrightarrow & \Gamma^* \\ a_1 \ldots a_n & \longmapsto & \mathrm{RegContent}(\pi_0 \circ \sigma_{a_n} \circ \ldots \circ \sigma_{a_1} \circ \epsilon^k) \end{array}$$

Now we will argue that for every monotone register transducer with n registers, we can produce an equivalent $\lambda\wp$-transducer with some iteration type $\kappa_n \multimap \mathbb{o}$. The intuition behind the definition of κ_n is that a register holding a string that support concatenations can be encoded using the type $\mathbb{o} \multimap \mathbb{o}$ and composition. As we need n copies of those, we thus set

$$\kappa_n = \underbrace{(\mathbb{o} \multimap \mathbb{o}) \multimap \ldots \multimap (\mathbb{o} \multimap \mathbb{o})}_{n\text{-fold}} \multimap \mathbb{o}$$

so that $\kappa_n \multimap \mathbb{o}$ is a sufficiently expressive stand-in for the n-fold tensor of $\mathbb{o} \multimap \mathbb{o}$.

Lemma 4.11 *Every $\sigma \in \mathrm{RegUp}(k,n)$ maps to a $\lambda\wp$-term $\underline{\Gamma}; \cdot \vdash \underline{\sigma} : \kappa_n \to \kappa_k$ in a way that is compatible with composition, that is $\underline{\sigma \circ \sigma'} =_{\beta\eta} \lambda z.\ \underline{\sigma'}\ (\underline{\sigma}\ z)$. Finally, if $\sigma \in \mathrm{RegUp}(0,1)$, we have $\underline{\mathrm{RegContent}(\sigma)} =_{\beta\eta} \underline{\sigma}\ (\lambda x.x)$.*

Proof idea For $\sigma = (w_1,\ldots,w_n) \in \mathrm{RegUp}(k,n)$, define $\underline{\sigma} : \kappa_n \to \kappa_k$ to be the term $\lambda F\ f_1 \ldots f_k.F\ t_1 \ldots t_n$ where t_i is obtained by recursion over w_i, starting with the identity and postcomposing with

- the appropriate constant from $\underline{\Gamma}$ when we encounter a letter of Γ
- f_k if we encounter the index k

This is typable in $\lambda\wp$ specifically because the transitions are monotone and copyless. Then it is relatively straightforward to check that we have the advertised equations. □

Then the $\lambda\wp$-terms corresponding to transitions will essentially precompose the suitable terms $\underline{\sigma}$ defined in Lemma 4.11. This corresponds to applying the exponentiation operation, defined over terms of type

[7] This could have alternatively been defined as a free affine strict monoidal category with a monoid object and generators for the letters of Γ.

$\tau \multimap \sigma$ by $t \multimap \mathbb{o} = \lambda X.\lambda z.\ X\ (t\ z)$. This operation is compatible with composition, i.e. we have $(t \multimap \mathbb{o}) \circ (u \multimap \mathbb{o}) =_{\beta\eta} (u \circ t) \multimap \mathbb{o}$ for arbitrary terms t and u which make those expressions typecheck.

Lemma 4.12 *Every function definable by a monotone register transducer is $\lambda\wp$ definable.*

Proof idea Suppose we are given such a transducer with n registers and transitions $(\sigma_a)_{a\in\Sigma}$ and let us build terms as per Definition 3.7. We take for iteration type $\kappa_n \multimap \mathbb{o}$, $d_a = \underline{\sigma_a} \multimap \mathbb{o}$, $d_\epsilon = \lambda Z.Z\ (\lambda x.x)\ \ldots\ (\lambda x.x)$ and $o = \lambda K.(K \circ \underline{\pi_0})\ (\lambda Z.Z\ (\lambda x.x))$. Then using Lemma 4.11, we can check step-by-step we have the desired equations. ☐

5 Conclusion

We have now proven that affine $\lambda\wp$-definable string-to-string functions correspond exactly to first-order transductions. One key aspect of the proof was to use a semantic interpretation of purely affine λ-terms as planar diagrams to compile $\lambda\wp$-transducers to 2PRFTs. This result essentially closes the open questions raised in [28] and provides an alternative, less syntactic, proof for the soundness part of its main theorem.

We will now discuss further results that could be derived by adapting the material we have developed in the previous section. We will then list some questions that arise because of, or could be solved using, the interpretation of terms as (planar) diagrams.

5.1 Discussion on variations: dropping planarity, regular transductions & tree languages

A natural variation on $\mathsf{TransDiag}_\Gamma$ is to drop the planarity requirement on the morphisms so that wires may cross in the geometric realizations of diagrams. If we do so, the tensor product becomes *symmetric*, that is we have a natural isomorphisms $\gamma_{A,B} : A \otimes B \to B \otimes A$ such that $\gamma_{A,B} = \gamma_{B,A}^{-1}$ and $\gamma_{A,\mathbf{I}} = \mathrm{id}_A$ [8], while still keeping a poset-enriched autonomous structure. This change makes the order of nodes in diagrams irrelevant, and objects with the same number of $+$ and $-$ occurring isomorphic [9] This allows to model the commutative variation of $\lambda\wp$, which we call λa, where we include the exchange rule:

$$\frac{\Gamma;\ \Delta, y : \tau_2, x : \tau_1, \Delta' \vdash t : \sigma}{\Gamma;\ \Delta, x : \tau_1, y : \tau_2, \Delta' \vdash t : \sigma}$$

If we define what are (affine) λa-definability and λa-transducers in a manner analogous to $\lambda\wp$-definability and $\lambda\wp$-transducers, as well as the notion of (not necessarily planar) *two-way reversible finite transducers* (2RFTs, which closely match the notion in [9] and thus capture all *regular* transductions) we have the following.

Theorem 5.1 *Affine λa-definable functions and regular transductions coincide:*

- *λa-transducers can be translated into equivalent 2RFTs in polynomial time*
- *regular transductions are λa-definable*

Proof idea The first point is obtained by an easy adaptation the arguments of Subsections 4.1 (where we add the interpretation of the exchange rule using the symmetry γ) and 4.2. The second point is also obtained by an argument similar to 4.3: Lemma 4.8 holds if we replace "aperiodic sequential" by "sequential" and "first-order transduction" by "regular transduction". We then need to know that all sequential functions are λa-definable, which is true by [28, Theorem 5.4]. ☐

---

[8] This is of course for the strict monoidal product.

[9] Quotienting sensibly yields a (poset-enriched) category isomorphic to the one computed by applying the Int construction [?, §4] to the category whose objects are natural numbers regarded as finite sets and morphisms from n to k are the subsets of $n \times \Gamma^* \times k$ that induce partial injections from n to k. The composition is then defined by $f \circ g = \{(i, uv, j) \mid \exists \ell.\ (i, u, \ell) \in f \wedge (\ell, v, j) \in g\}$ and then the traced monoidal structure is defined analogously to that of the category of partial injections.

This statement should be contrasted with [26, Theorem 1.1] which states that regular string-to-string transductions coincide with functions definable in a variant of λa which is augmented with additives [10]. There, λ-terms defining string-to-string functions are compiled into streaming string transducer (SSTs). But this translation can yield a machine that has a state-space whose size is non-elementary in terms of the size of an input λa-transduction free of additives connectives. Since the translations between 2RFTs and SSTs is Elementary [9], the translation we offer here is more efficient. On the other hand, the second point improves on [26] by compiling first-order transductions in a smaller λ-calculus at the cost of employing Lemma 4.8 that relies on the powerful and relatively complex technique of Krohn-Rhodes decomposition instead of a direct polynomial-time compilation of SSTs.

While we have only investigated functions that take as string inputs in this paper, the tools we have introduced can be used to study functions that take ranked trees as input (and still output strings). Indeed, ranked trees, that are parameterized by finite ranked alphabet can be represented by Church encodings and given precise affine typing (c.f. [26, §2.3]). In that case, λa terms get compiled to what amounts to *reversible tree-walking transducers with string output* (or simply reversible tree-walking automata if we take the output alphabet to be empty) as defined by restricting Definitions 3.5 and 3.8 of [29] to string outputs. As a result, we can give a new proof of the following theorem, which is also a consequence of [29, Theorem 1.4] [11].

Theorem 5.2 *Every* λa *tree-to-string transducer can be turned into an equivalent reversible tree-walking transducer.*

This result means the affine λ-calculus without additives cannot recognize all regular tree languages [5], whereas allowing additives captures all regular tree transductions [26, Theorem 1.2].

5.2 Perspectives

A natural question is whether Theorem 5.2 admits a converse: is every language recognized by a reversible tree-walking automaton also recognized by some λa-term? Another natural question is "what are the tree languages recognized by λ℘-terms?". Clearly, they should be recognized by tree-walking automata that are not only *reversible*, but also *planar* in the obvious sense. This is an actual restriction, as a non-planar tree-walking automaton could count the number of leaves modulo 2, which a planar device could not. So we can also ask the question: is every language recognized by a planar reversible tree-walking automaton also recognized by some λ℘-term? These questions might be challenging since we are currently not aware of a convenient tool similar to the Krohn-Rhodes theorem or [7, Theorem 3.4] that would allow to decompose tree-walking transducers in elemental functions. A first step might be to check that those transducers, as well as their planar variant, are closed under composition. This would require considering tree-to-tree transductions as discussed in [29], which would naturally lead to extending our diagrammatic constructions so that they may depend on a ranked alphabet, much like the categories of register updates considered in [26]. A variant of the operad of spliced arrows specified in [22, Definition 1.1] could be of use.

We have treated only *affine* λ℘-definable functions in this paper. The next question is whether we can also get a characterization of λ℘-definable functions implemented by terms of type $\mathrm{Str}_\Sigma[\kappa] \to \mathrm{Str}_\Gamma$. It is plausible they correspond to *first-order polyblind* functions alluded to in [25] [12], which are obtained by closing first-order transductions under *compositions by substitution* [25, Definition 4.1]. Our hope is that this correspondence can be established using a similar strategy as [24, §5.3].

References

[1] Abramsky, S., *Temperley–Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics*, in: G. Chen, L. Kauffman and S. Lomonaco, editors, *Mathematics of Quantum Computation and Quantum Technology*,

[10] And also linear instead of affine, however in the presence of additives, this distinction is not very important (see [15, §1.2.1] for a discussion).

[11] Both arguments essentially appeal to Girard's geometry of interaction, but theirs is based on an compiling executions of an abstract machine evaluating λ-terms while we focus on a semantic interpretation of linear logic.

[12] In which they were called first-order comparison-free. We follow the terminological change introduced in [10].

volume 20074453, pages 515–558, Chapman and Hall/CRC (2007), ISBN 978-1-58488-899-4 978-1-58488-900-7.
https://doi.org/10.1201/9781584889007.ch15

[2] Adamek, J. and V. Trnkova, *Automata and Algebras in Categories*, Kluwer Academic Publishers, USA, 1st edition (1990), ISBN 0792300106.

[3] Alur, R. and P. Černý, *Expressiveness of streaming string transducers*, in: K. Lodaya and M. Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010).
https://doi.org/10.4230/LIPIcs.FSTTCS.2010.1

[4] Arbib, M. A. and E. G. Manes, *Adjoint machines, state-behavior machines, and duality*, Journal of Pure and Applied Algebra **6**, pages 313–344 (1975), ISSN 0022-4049.
https://doi.org/https://doi.org/10.1016/0022-4049(75)90028-6

[5] Bojańczyk, M. and T. Colcombet, *Tree-walking automata do not recognize all regular languages*, SIAM Journal on Computing **38**, pages 658–701 (2008).
https://doi.org/10.1137/050645427

[6] Bojańczyk, M., L. Daviaud and S. N. Krishna, *Regular and First-Order List Functions*, in: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, pages 125–134, ACM Press, Oxford, United Kingdom (2018), ISBN 978-1-4503-5583-4.
https://doi.org/10.1145/3209108.3209163

[7] Bojańczyk, M. and A. Doumane, *First-order tree-to-tree functions*, in: H. Hermanns, L. Zhang, N. Kobayashi and D. Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany (online conference), July 8-11, 2020*, pages 252–265, ACM (2020).
https://doi.org/10.1145/3373718.3394785

[8] Colcombet, T. and D. Petrişan, *Automata Minimization: a Functorial Approach*, Logical Methods in Computer Science **16** (2020).
https://doi.org/10.23638/LMCS-16(1:32)2020

[9] Dartois, L., P. Fournier, I. Jecker and N. Lhote, *On reversible transducers*, in: I. Chatzigiannakis, P. Indyk, F. Kuhn and A. Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 113:1–113:12, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017).
https://doi.org/10.4230/LIPIcs.ICALP.2017.113

[10] Douéneau-Tabot, G., *Hiding pebbles when the output alphabet is unary*, in: M. Bojanczyk, E. Merelli and D. P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 120:1–120:17, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022).
https://doi.org/10.4230/LIPICS.ICALP.2022.120

[11] East, J., *Presentations for Temperley–Lieb Algebras*, The Quarterly Journal of Mathematics **72**, pages 1253–1269 (2021), ISSN 0033-5606.
https://doi.org/10.1093/qmath/haab001

[12] Freyd, P. J. and D. N. Yetter, *Braided compact closed categories with applications to low dimensional topology*, Advances in Mathematics **77**, pages 156–182 (1989), ISSN 0001-8708.
https://doi.org/https://doi.org/10.1016/0001-8708(89)90018-2

[13] Girard, J.-Y., *Linear logic*, Theoretical Computer Science **50**, pages 1–101 (1987), ISSN 0304-3975.
https://doi.org/10.1016/0304-3975(87)90045-4

[14] Girard, J.-Y., *Towards a geometry of interaction*, in: J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 69–108, American Mathematical Society, Providence, RI (1989). Proceedings of a Summer Research Conference held June 14–20, 1987.
https://doi.org/10.1090/conm/092/1003197

[15] Girard, J.-Y., *Linear logic: its syntax and semantics*, in: J.-Y. Girard, Y. Lafont and L. Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 1–42, Cambridge University Press (1995).
https://doi.org/10.1017/CBO9780511629150.002

[16] Goguen, J. A., *Minimal realization of machines in closed categories*, Bull. Amer. Math. Soc. **78**, pages 777–783 (1972).
http://dml.mathdoc.fr/item/1183533991

[17] Grellois, C., *Semantics of linear logic and higher-order model-checking*, Ph.D. thesis, Université Paris 7 (2016).
https://tel.archives-ouvertes.fr/tel-01311150/

[18] Grellois, C. and P.-A. Melliès, *Finitary semantics of linear logic and higher-order model-checking*, in: *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015*, pages 256–268 (2015).
https://doi.org/10.1007/978-3-662-48057-1_20

[19] Hillebrand, G. G. and P. C. Kanellakis, *On the Expressive Power of Simply Typed and Let-Polymorphic Lambda Calculi*, in: *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 253–263, IEEE Computer Society (1996), ISBN 978-0-8186-7463-1.
https://doi.org/10.1109/LICS.1996.561337

[20] Hines, P., *Temperley-Lieb Algebras as two-way automata*, http://www.dcs.gla.ac.uk/~simon/qnet/talks/Hines.pdf (2006). Slides of a talk given at the QNET Workshop 2006.

[21] Kelly, G. M., *Basic concepts of enriched category theory*, Reprints in Theory and Applications of Categories **10**, pages 1–136 (2005).
http://www.tac.mta.ca/tac/reprints/articles/10/tr10.pdf

[22] Melliès, P.-A. and N. Zeilberger, *The categorical contours of the Chomsky-Schützenberger representation theorem* (2023). This is a thoroughly revised and expanded version of a paper with a similar title (hal-03702762, arXiv:2212.09060) presented at the 38th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2022). 62 pages, including a 13 page Addendum on "gCFLs as initial models of gCFGs", and a table of contents.
https://hal.science/hal-04399404

[23] Moreau, V. and L. T. D. Nguyễn, *Syntactically and semantically regular languages of lambda-terms coincide through logical relations* (2023). 2308.00198.

[24] Nguyễn, L. T. D., *Implicit automata in linear logic and categorical transducer theory*, Ph.D. thesis, Université Paris XIII (Sorbonne Paris Nord) (2021).
https://theses.hal.science/tel-04132636

[25] Nguyễn, L. T. D., C. Noûs and C. Pradic, *Comparison-free polyregular functions*, in: N. Bansal, E. Merelli and J. Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 139:1–139:20, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021).
https://doi.org/10.4230/LIPICS.ICALP.2021.139

[26] Nguyễn, L. T. D., C. Noûs and C. Pradic, *Implicit automata in typed λ-calculi II: streaming transducers vs categorical semantics*, CoRR **abs/2008.01050** (2020). 2008.01050.

[27] Nguyễn, L. T. D., C. Noûs and C. Pradic, *Two-way automata and transducers with planar behaviours are aperiodic* (2023). 2307.11057.

[28] Nguyễn, L. T. D. and C. Pradic, *Implicit automata in typed λ-calculi I: aperiodicity in a non-commutative logic*, in: A. Czumaj, A. Dawar and E. Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 135:1–135:20, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020).
https://doi.org/10.4230/LIPIcs.ICALP.2020.135

[29] Nguyễn, L. T. D. and G. Vanoni, *(almost) affine higher-order tree transducers*, CoRR **abs/2402.05854** (2024). 2402.05854.
https://doi.org/10.48550/ARXIV.2402.05854

[30] Rutten, J., *Universal coalgebra: a theory of systems*, Theoretical Computer Science **249**, pages 3–80 (2000), ISSN 0304-3975. Modern Algebra.
https://doi.org/https://doi.org/10.1016/S0304-3975(00)00056-6

[31] Selinger, P., *A Survey of Graphical Languages for Monoidal Categories*, page 289–355, Springer Berlin Heidelberg (2010), ISBN 9783642128219.
https://doi.org/10.1007/978-3-642-12821-9_4

[32] Spivak, D. I., *Poly: An abundant categorical setting for mode-dependent dynamics* (2020). 2005.01894.

A $(\mathsf{TransDiag}_\Gamma, +, +)$-automata vs 2PRFTs

Let us detail here why $(\mathsf{TransDiag}_\Gamma, +, +)$-automata and 2PRFTs with output alphabet Γ define the same string-to-string functions.

First, let us note that both options define an output in $\Gamma^*_\bot$ by examining, when it exists, the label of the single possible edge of $[+,+]_{\mathsf{TransDiag}_\Gamma}$ and $[\mathbf{I}, +-]_{\mathsf{TransDiag}_\Gamma}$. This operation commutes with the isomorphism

$$\begin{array}{ccc} [+,+]_{\mathsf{TransDiag}_\Gamma} & \cong & [\mathbf{I}, +-]_{\mathsf{TransDiag}_\Gamma} \\ f & \longmapsto & \varepsilon_+; (f \otimes \mathrm{id}_-) \\ (g \otimes \mathrm{id}_+); (\mathrm{id}_+ \otimes \eta_+) & \longleftarrow & g \end{array}$$

For the purpose of this discussion, fix an input alphabet Σ. First assume we are given a $(\mathsf{TransDiag}_\Gamma, +, +)$-automaton $\mathcal{A}$. We define an equivalent 2PRT (i.e. a $(\mathsf{TransDiag}_\Gamma, \epsilon, +-)$-automaton) $\mathcal{A}'$ by setting

$$\mathcal{A}'(\mathrm{states}) = \mathcal{A}(\mathrm{states}) \otimes - \qquad \mathcal{A}'(\triangleright) = \varepsilon_+; (\mathcal{A}(\triangleright) \otimes \mathrm{id}_-) \qquad \text{and otherwise} \qquad \mathcal{A}'(f) = \mathcal{A}(f) \otimes \mathrm{id}_-$$

which is easily checked to be functorial and is equivalent to $\mathcal{A}$ as we have

$$\begin{array}{rcll} (\mathcal{A}'(\triangleright w \triangleleft) \otimes \mathrm{id}_+); (\mathrm{id}_+ \otimes \eta_+) & = & ((\mathcal{A}'(\triangleright); \mathcal{A}'(w\triangleleft)) \otimes \mathrm{id}_+); (\mathrm{id}_+ \otimes \eta_+) & \text{by functoriality of } \mathcal{A}' \\ & = & ((\varepsilon_+; (\mathcal{A}(\triangleright) \otimes \mathrm{id}_-); (\mathcal{A}(w\triangleleft) \otimes \mathrm{id}_-)) \otimes \mathrm{id}_+); (\mathrm{id}_+ \otimes \eta_+) & \text{by definition of } \mathcal{A}' \\ & = & (\varepsilon_+ \otimes \mathrm{id}_+); ((\mathcal{A}(\triangleright); \mathcal{A}(w\triangleleft)) \otimes \mathrm{id}_{-+}); (\mathrm{id}_+ \otimes \eta_+) & \text{by functoriality of } \otimes \\ & = & (\varepsilon_+ \otimes \mathrm{id}_+); (\mathrm{id}_+ \otimes \eta_+); (\mathcal{A}(\triangleright); \mathcal{A}(w\triangleleft)) & \text{by naturality of } \eta \\ & = & \mathcal{A}(\triangleright); \mathcal{A}(w\triangleleft) & \text{by a zigzag equation} \\ & = & \mathcal{A}(\triangleright w \triangleleft) & \text{by functoriality} \end{array}$$

Conversely, if we have a 2PRFT $\mathcal{T}$, we can turn it into an equivalent $(\mathsf{TransDiag}_\Gamma, +, +)$-automaton $\mathcal{T}'$ by setting

$$\mathcal{T}'(\mathrm{states}) = \mathcal{T}(\mathrm{states}) \otimes + \qquad \mathcal{T}'(\triangleleft) = \mathcal{T}(\triangleleft) \otimes \eta_+ \qquad \text{and otherwise} \qquad \mathcal{T}'(f) = \mathcal{A}(f) \otimes \mathrm{id}_+$$

The proof that it is equivalent to $\mathcal{T}$ is similar to the one above, exploiting the naturality of ε and the other zigzag equation.

B Proof of Lemma 4.3

Lemma B.1 *Suppose we have $\underline{\Gamma}; \Delta, x : \tau, \Delta' \vdash t : \sigma$ and $\underline{\Gamma}; \Delta'' \vdash u : \tau$. Then we have*

$$[\![t[u/x]]\!] \le [\![t]\!] \circ (\mathrm{id}_{[\![\Delta]\!]} \otimes [\![u]\!] \otimes \mathrm{id}_{[\![\Delta']\!]}) \qquad (\ : [\![\Delta, \Delta'', \Delta']\!] \to [\![\tau]\!])$$

Proof The proof is by induction over the typing derivation of t. We will use throughout that $\circ$ and $\otimes$ are monotone, that $\bot_A \otimes \bot_B = \bot_{A \otimes B}$ as well as $\mathrm{id}_{\mathbf{I}} = \bot_{\mathbf{I}}$ and that $\bot_A \le f$ for any $f : \mathbf{I} \to A$ without calling explictly attention to it.

- If t is the variable x, then both sides are equal to $(\bot_{[\![\Delta]\!]} \otimes [\![u]\!] \otimes \bot_{[\![\Delta']\!]})$.
- If t a variable other than x from the linear part of the context, say y from Δ such that we have $\Delta = \Theta, y : \sigma, \Theta'$ (the case where y is from Δ' is treated analogously), we derive the following using that

$\bot_{\llbracket\Delta''\rrbracket} \leq \bot_{\llbracket\sigma\rrbracket} \circ \llbracket u \rrbracket$:

$$\begin{aligned}
\llbracket y[u/x] \rrbracket &= \bot_{\llbracket\Theta\rrbracket} \otimes \mathrm{id}_{\llbracket\sigma\rrbracket} \otimes \bot_{\llbracket\Theta',\Delta'',\Delta'\rrbracket} \\
&= \bot_{\llbracket\Theta\rrbracket} \otimes \mathrm{id}_{\llbracket\sigma\rrbracket} \otimes \bot_{\llbracket\Theta'\rrbracket} \otimes \bot_{\Delta''} \otimes \bot_{\llbracket\Delta'\rrbracket} \\
&\leq \bot_{\llbracket\Theta\rrbracket} \otimes \mathrm{id}_{\llbracket\sigma\rrbracket} \otimes \bot_{\llbracket\Theta'\rrbracket} \otimes (\bot_{\llbracket\sigma\rrbracket} \circ \llbracket u \rrbracket) \otimes \bot_{\llbracket\Delta'\rrbracket} \\
&= \bot_{\llbracket\Theta\rrbracket} \otimes \mathrm{id}_{\llbracket\sigma\rrbracket} \otimes \bot_{\llbracket\Theta'\rrbracket} \otimes (\bot_{\llbracket\sigma\rrbracket} \circ \llbracket u \rrbracket) \otimes \bot_{\llbracket\Delta'\rrbracket} \\
&= (\bot_{\llbracket\Theta\rrbracket} \otimes \mathrm{id}_{\llbracket\sigma\rrbracket} \otimes \bot_{\llbracket\Theta'\rrbracket} \otimes \bot_{\llbracket\sigma\rrbracket} \otimes \bot_{\llbracket\Delta'\rrbracket}) \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket}) \\
&= \llbracket y \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket})
\end{aligned}$$

- If t is a variable of $\underline{\Gamma}$, the desired inequality follows from $\bot_{\llbracket\Delta,\Delta'',\Delta'\rrbracket} \leq \bot_{\llbracket\Delta\rrbracket\otimes\llbracket\tau\rrbracket\otimes\llbracket\Delta'\rrbracket} \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket})$.
- If $t = f\ g$ for $f : \tau \multimap \sigma$, $g : \tau$, we have two subcases according to which context x appears in.
  - · Suppose x appears in the context of f so that we have, $\Delta' = \Delta'_f, \Delta'_g$ and judgements

$$\underline{\Gamma}; \Delta, x : \tau, \Delta'_f \vdash f : \tau \multimap \sigma \qquad \text{and} \qquad \underline{\Gamma}; \Delta'_g \vdash t : \tau$$

By the induction hypothesis, we have $\llbracket f[u/x] \rrbracket \leq \llbracket f \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'_f\rrbracket})$, which allows to derive

$$\begin{aligned}
\llbracket t[u/x] \rrbracket &= \llbracket f[u/x]\ g \rrbracket \\
&= \mathrm{ev}_{\llbracket\tau\rrbracket,\llbracket\sigma\rrbracket} \circ (\llbracket f[u/x] \rrbracket \otimes \llbracket g \rrbracket) \\
&\leq \mathrm{ev}_{\llbracket\tau\rrbracket,\llbracket\sigma\rrbracket} \circ \Big(\Big(\llbracket f \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'_f\rrbracket})\Big) \otimes \llbracket g \rrbracket\Big) \\
&= \mathrm{ev}_{\llbracket\tau\rrbracket,\llbracket\sigma\rrbracket} \circ \Big(\Big(\llbracket f \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \mathrm{id}_{\llbracket\tau\rrbracket} \otimes \mathrm{id}_{\llbracket\Delta'_f\rrbracket})\Big) \otimes \llbracket g \rrbracket\Big) \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'_f\rrbracket} \otimes \mathrm{id}_{\llbracket\Delta'_g\rrbracket}) \\
&= \mathrm{ev}_{\llbracket\tau\rrbracket,\llbracket\sigma\rrbracket} \circ \Big(\llbracket f \rrbracket \circ (\mathrm{id}_{\llbracket\Delta,x:\tau,\Delta'_f\rrbracket} \otimes \llbracket g \rrbracket)\Big) \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'_f\rrbracket} \otimes \mathrm{id}_{\llbracket\Delta'_g\rrbracket}) \\
&= \llbracket f\ g \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'_f\rrbracket} \otimes \mathrm{id}_{\llbracket\Delta'_g\rrbracket})
\end{aligned}$$

  - · The case when x appears in the context of g is very similar and left to the reader.
- If $t[u/x] = \lambda y.\ t'[u/x]$ with $y \neq x$, then the premise of the rule under consideration is $\underline{\Gamma}; \Delta, x : \tau, \Delta', y : \tau' \vdash t : \sigma$ and the induction hypothesis thus is

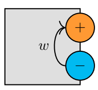
$$\llbracket t'[u/x] \rrbracket \leq \llbracket t' \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket} \otimes \mathrm{id}_{\llbracket\tau'\rrbracket})$$

So the result is then derived as follows, using the monotonicity of Λ and that we have $\Lambda_{A,B,C}(h \circ (\ell \otimes \mathrm{id}_B)) = \Lambda_{A,B,C}(h) \circ \ell$ in monoidal closed categories:

$$\begin{aligned}
\llbracket t[u/x] \rrbracket &= \llbracket \lambda y.\ t'[u/x] \rrbracket \\
&= \Lambda_{\llbracket\Delta\rrbracket\otimes\llbracket\tau\rrbracket\otimes\llbracket\Delta'\rrbracket,\llbracket\tau'\rrbracket,\llbracket\sigma\rrbracket}(\llbracket t'[u/x] \rrbracket) \\
&\leq \Lambda_{\llbracket\Delta\rrbracket\otimes\llbracket\tau\rrbracket\otimes\llbracket\Delta'\rrbracket,\llbracket\tau'\rrbracket,\llbracket\sigma\rrbracket}(\llbracket t' \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket} \otimes \mathrm{id}_{\llbracket\tau'\rrbracket})) \\
&= \Lambda_{\llbracket\Delta\rrbracket\otimes\llbracket\tau\rrbracket\otimes\llbracket\Delta'\rrbracket,\llbracket\tau'\rrbracket,\llbracket\sigma\rrbracket}(\llbracket t' \rrbracket) \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket})) \\
&= \llbracket \lambda y.\ t' \rrbracket \circ (\mathrm{id}_{\llbracket\Delta\rrbracket} \otimes \llbracket u \rrbracket \otimes \mathrm{id}_{\llbracket\Delta'\rrbracket}))
\end{aligned}$$

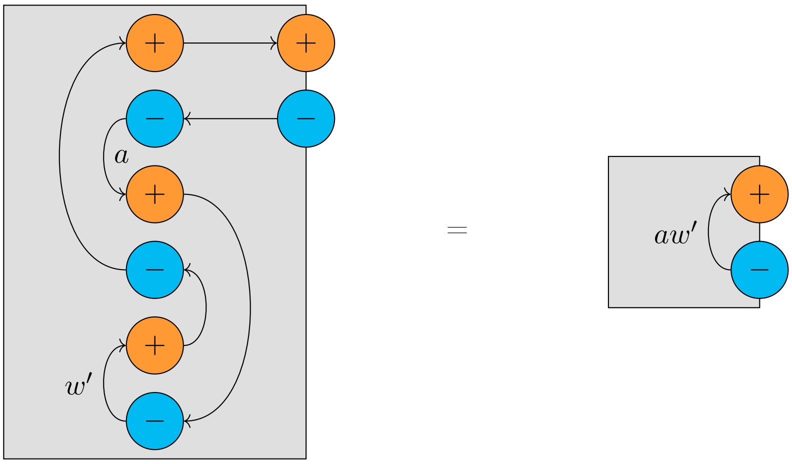
□

C Proof of Lemma 4.6

Lemma C.1 *For $w \in \Gamma^*$, $\underline{\Gamma}; \cdot \vdash \underline{w} : \mathbb{o}$ is interpreted by the diagram* 

.

Proof This is done by an induction over w. When $w = \epsilon$, this is obvious. When $w = aw'$, we have $[\![\underline{aw'}]\!] = [\![a\ \underline{w'}]\!] = ([\![a]\!] \otimes [\![\underline{w'}]\!]); \mathrm{ev}_{[\![o]\!],[\![o]\!]}$. Applying the induction hypothesis and drawing out the picture of this composition, we can conclude by chasing the path.



$([\![a]\!] \otimes [\![\underline{w'}]\!])\ ;\ \mathrm{ev}_{+-,+-}$

□