IMPLICIT AUTOMATA IN TYPED λ-CALCULI II: STREAMING TRANSDUCERS VS CATEGORICAL SEMANTICS

LÊ THÀNH DŨNG (TITO) NGUYỄN, CAMILLE NOÛS, AND CÉCILIA PRADIC

Laboratoire d'informatique de Paris Nord, Villetaneuse, France
e-mail address: nltd@nguyentito.eu

Laboratoire Cogitamus
URL: https://www.cogitamus.fr/camilleen.html

Department of Computer Science, University of Oxford, United Kingdom

ABSTRACT. We characterize regular string transductions as programs in a linear λ-calculus with additives. One direction of this equivalence is proved by encoding copyless streaming string transducers (SSTs), which compute regular functions, into our λ-calculus. For the converse, we consider a categorical framework for defining automata and transducers over words, which allows us to relate register updates in SSTs to the semantics of the linear λ-calculus in a suitable monoidal closed category.

To illustrate the relevance of monoidal closure to automata theory, we leverage this notion to give abstract generalizations of the arguments showing that copyless SSTs may be determinized and that the composition of two regular functions may be implemented by a copyless SST.

Our main result is then generalized from strings to trees using a similar approach. In doing so, we exhibit a connection between a feature of streaming tree transducers and the multiplicative/additive distinction of linear logic.

1. INTRODUCTION

We recently initiated [NP20] a series of works at the interface of programming language theory and automata. As the title suggests, the present paper is the second installment; it starts with an introduction to this research programme, meant to be accessible with a general computer science background (Section 1.1). After stating a main theorem, we shall argue, in two mostly independent subsections, that these connections between two fields that we investigate:

- are relevant to natural questions on the λ-calculus (Section 1.2);
- provide new conceptual insights into automata theory (Section 1.3).

Section 1.4 exposes some of our key technical ideas, stressing the role of category theory as a mediating language. A table of contents is provided after this introduction.

1.1. **What is this all about?**

1.1.1. *From proofs-as-programs to implicit complexity.* One of the central principles in the contemporary theory of programming languages is a close relationship between constructive logics and statically typed functional programming, known as the *proofs-as-programs corre-spondence*, also known as the *formulae-as-types* or *Curry–Howard* correspondence. The idea is that, in certain logical systems, proofs admit a "normalization procedure" that can be seen as the execution of a program. According to this analogy, a proof is thus a program, and the *formula* that it proves is the *type* of the program. A remarkable empirical fact is that this manifests as several concrete isomorphisms between (theoretical) languages and proof systems that were designed independently.

An important point is that *termination* on the programming side is highly desirable in this context since it entails *consistency* on the logical side. Take for instance the *untyped λ-calculus*, one of the models of computation that led to the birth of computability theory in the 1930s, nowadays used as a theoretical foundation for functional programming. It allows non-terminating programs. The *simply typed λ-calculus* adds a type system on top of it; one can then rule out this possibility of non-termination by only allowing well-typed programs, thus ensuring the consistency of the corresponding logical system (here, intuitionistic propositional logic) at the price of losing Turing-completeness.

Such a termination guarantee might even come with quantitative time complexity bounds. For instance, Hillebrand et al. [HKM96] show that programs in the simply typed λ-calculus operating over certain data encodings and returning booleans can compute all functions[1] in the complexity class ELEMENTARY (i.e. those with a time complexity bounded by a tower of exponentials), and only those. This result illustrates the type-theoretic approach to *implicit computational complexity*, a well-established field concerned with machine-free characterizations of complexity classes via high-level programming languages[2]. Many works in this area have taken inspiration from *linear logic* [Gir87] to design more sophisticated type systems, starting with two characterizations of polynomial time [GSS92, Gir98]. As another example, Linear Logic by Levels [BM10] also characterizes ELEMENTARY and admits a translation from the simply typed λ-calculus [GRV09].

1.1.2. *Implicit automata.* Let us consider *strings* over some finite alphabet as input. Functions mapping these strings to booleans are equivalent to sets of strings, and the latter are called *languages*. Complexity classes (of decision problems) are often defined as *sets of languages*, but such sets also arise in *automata theory*. A typical example is the class of *regular languages*, that can be defined by regular expressions or equivalently by finite automata (we assume here that the reader knows about those): usually, it is not considered a complexity class, although this statement is sociological rather than formal[3].

Our research programme aims to provide for automata what (type-theoretic) implicit complexity has done for complexity classes. A characterization of regular languages had

---

[1]This does *not* mean that a given algorithm with elementary complexity must admit a direct implementation in the simply typed λ-calculus; instead, what must exist is a λ-calculus program computing the same function from inputs to outputs, with potentially different inner workings.

[2]We refer to the introduction of our previous paper [NP20] for a discussion of the difference between implicit computational complexity and descriptive complexity.

[3]One possible technical argument is that the class of regular languages is not closed under ALOGTIME (i.e. uniform AC^0) reductions.

already been obtained by Hillebrand and Kanellakis in the simply typed λ-calculus [HK96, Theorem 3.4]. Starting from this, we characterized [NP20] the smaller class of *star-free languages* by relying on a richer type system that supports so-called *non-commutative* types. As mentioned in [NP20, §7], some other results of this kind already exist, but not many; and as far as we know, the idea of "implicit automata" as a topic worthy of systematic investigation had not been put forth before in writing.

1.1.3. *Transducers.* Here, our goal is to go beyond languages and to consider *string-to-string functions* instead. There is a wide variety of classes of such functions that appear in automata theory, and several of them collapse to regular languages when we restrict them to a single output bit (this is the case for the so-called sequential functions, rational functions, regular functions... see the surveys [FR16, MP19]). In other words, many automata models that recognize regular languages are no longer equivalent when extended with the ability to produce an output string. Such automata with output are called *transducers*. We could therefore expect fine distinctions between the feature sets of various λ-calculi to be reflected in differences between the string functions that they can express.

Yet we only know of two precedents for string-to-string transduction classes captured using typed functional programming: sequential functions in a cyclic proof system [DP16] and polyregular functions in a λ-calculus with primitive data types [Boj18, §4]. Both are discussed further in the prequel paper [NP20, §7].

This brings us to our first main theorem:

Theorem 1.1. *A function $\Sigma^* \to \Gamma^*$ is $\lambda\ell^{\oplus\&}$-definable if and only if it is regular.*

By "$\lambda\ell^{\oplus\&}$-definable", we mean expressible in the $\lambda\ell^{\oplus\&}$-calculus (a system based on Intuitionistic Linear Logic) in the specific but mostly standard way described in Definition 2.12. As for *regular functions*, they are a well-studied class with many equivalent definitions, for instance two-way transducers or monadic second-order logic [EH01]. We may also point to several recent formalisms [AFR14, DGK18, BDK18] for regular functions using combinators as belonging to "implicit complexity for automata", albeit not of the type-theoretic kind (implicit complexity is an umbrella term which traditionally also includes tools such as recursive function algebras or term rewriting).

1.2. **Internal motivations from typed λ-calculi.** We mentioned earlier a characterization of ELEMENTARY in the simply typed λ-calculus by Hillebrand et al. [HKM96]. It uses a somewhat unusual (though perfectly justified) representation for the inputs. The conventional choice would have been the *Church encoding* of strings. They are indeed the usual tool to represent all computable functions in the untyped λ-calculus, and in some terminating type systems, any "reasonable" function can still be programmed over these encodings (for example, this is the case for System F [Wad07]). But in the simply typed case, some earlier results by Statman had suggested a hopeless lack of expressiveness (see e.g. the discussion in [FLO83] after its Theorem 4.4.3.). Then Hillebrand and Kanellakis's aforementioned result [HK96] showed that, surprisingly, one gets the class of regular languages by using Church-encoded strings in the simply typed λ-calculus!

Recently, the present paper's first author reused their ideas in [Ngu19] to solve an open problem from "standard" implicit complexity, concerning a characterization of polynomial time by Baillot et al. [BDBRDR18] that makes use of Church encodings. The question was whether a certain feature (namely type fixpoints) was necessary for this result. It turns

out [Ngu19] that when this feature is removed, the class of languages obtained is regular languages instead of P.[4]

The moral of the story, for us, is that the use of Church-encoded strings can lead naturally to connections with automata theory. Admittedly, this naturality judgment is inherently subjective. But concretely, it translates into a methodological commitment: we explore the expressiveness of typed λ-calculi that already exist (perhaps up to a few minor details), whereas it is usual in implicit complexity to engineer some (potentially ad-hoc) new type system to achieve desired complexity effects. Most of the time, the features of these preexisting λ-calculi have original motivations that are entirely unrelated to complexity or automata (for instance, the non-commutative types that we used in [NP20] originate from the study of natural language [Lam58] and the topology of proofs (see e.g. [Gir89, §II.9.])).

In the case of the simply typed λ-calculus, characterizing the definable string-to-string functions in the style of [HK96, Theorem 3.4] (again!) is in fact an old open problem (while a more restrictive notion of λ-definability is well understood [Zai87]). As we are not yet able to solve it, we instead tackle a version where *linearity* constraints have been added, resulting in Theorem 1.1. Recall that a function is said to be linear (in the sense of linear logic) when it uses its argument only once. The system that we use to express these constraints is Dual Intuitionistic Linear Logic [Bar96] with additive connectives (called here the $\lambda\ell^{\oplus\&}$-calculus).

1.3. **Conceptual interest for (categorical) automata theory.** This notion of linearity in programming language theory has a counterpart in the old theme of *restricting the copying power* of automata models (see e.g. [ERS80]). The latter is manifested in one of the possible definitions of the regular functions mentioned in Theorem 1.1: *copyless*[5] *streaming string transducers* (SSTs) [AČ10]. An SST is roughly speaking an automaton whose internal memory consists of a state (in a finite set) and some string-valued registers, and its transitions are copyless when they compute new register values without duplicating the old ones.

Put this way, Theorem 1.1 seems unsurprising. But there is more going on behind the scenes. In particular, while it is trivial that the composition of two $\lambda\ell^{\oplus\&}$-definable functions is itself $\lambda\ell^{\oplus\&}$-definable, composing copyless SSTs requires intricate combinatorics as can be seen in [BC18, Chapter 13] for example. As it turns out, the tools developed in order to prove Theorem 1.1 also yield a clean proof of the closure of copyless streaming string transducers under composition, which it even generalizes using the language of *category theory*, see below.

Another subtlety comes from our extension of Theorem 1.1 to *ranked trees*:

Theorem 1.2. *Let $\boldsymbol{\Sigma}$ and $\boldsymbol{\Gamma}$ be ranked alphabets. A function $\mathbf{Tree}(\boldsymbol{\Sigma}) \to \mathbf{Tree}(\boldsymbol{\Gamma})$ is $\lambda\ell^{\oplus\&}$-definable if and only if it is regular.*

The class of *regular tree functions* is obtained by generalizing the definition for strings based on monadic second-order logic (MSO, see e.g. [BD20]). There is also an automata model adapting SSTs to trees, namely the *bottom-up ranked tree transducers* (BRTT) [AD17]. However, it is conjectured that some regular functions *cannot* be computed by *copyless*

---

[4]Digression: in [NP19], we explored the input representation of [HKM96] transposed into a language similar to that of [BDBRDR18], without these type fixpoints. We gathered some evidence suggesting that one gets a characterization of deterministic logarithmic space (though the upper bound that we manage to prove is a bit weaker).

[5]The adjective "copyless" does not appear in the original paper [AČ10] but is nowadays commonly used to distinguish them from the later *copyful SSTs* [FR21].

BRTTs. Instead, a more sophisticated linearity condition, called the *single use restriction*[6], is imposed on BRTTs in [AD17] in order to characterize the regular tree functions. The additional flexibility[7] thus afforded, compared to copyless BRTTs, turns out to correspond directly to an important feature of linear type systems, namely the *additive conjunction*.

As a concrete manifestation of this correspondence, we conjecture that all tree functions definable in the λa-calculus of our previous paper [NP20] can be computed by copyless BRTTs. This λa-calculus does not contain the additive connectives $\&/\oplus$ of linear logic; to compensate, it has an affine type system, instead of a linear one (whence the *a*). We leave the proof of this fact for future work.

In the case of strings, single-use-restricted streaming string transducers are very close to copyless *non-determinstic* SSTs. (That additive connectives in linear logic have something to do with non-determinism has previously been observed in other settings, for instance in [MT03].) Their equivalence with copyless SSTs thus corresponds to a *determinization* theorem, that already has an indirect proof via MSO [AD11]. We provide here a direct construction, whose main technical ingredient is the "transformation forest" data structure applied to copyless SSTs in [BC18, Chapter 13] and reminiscent of the Muller–Schupp determinization [MS95] for automata over infinite words. Most importantly, this determinization result is again formulated in a general *category-theoretic* setting.

Together with the aforementioned analysis of the composition of SSTs, those are our two contributions to "categorical automata theory". This kind of use of categories to understand the essence of various constructions on automata – such as determinization or minimization – and to generalize them to other settings has a long history, see for instance [vHKR$^+$19] and the many references therein[8].

1.4. **Transducers over monoidal closed categories.** Another example of categorical automata theory is the work of Colcombet and Petrişan [CP17a, CP20] on minimization, whose direct relevance to us lies in the *categorical framework for automata models* that it introduces: objects serve as state spaces and morphisms as transitions. Our technical development takes place in a very similar framework.

- We first define a category $\mathcal{SR}$ that corresponds to *single-state* copyless SSTs.
- Since copylessness and the so-called single use restriction morally differ by the presence of the additive conjunction '&' of linear logic, we "add '&' freely" to achieve a similar effect: automata in the resulting category $\mathcal{SR}_\&$, although not identical to single-state single use restricted SSTs, are easily seen to be equally expressive.
- Finally, we perform another "completion" denoted $(-)_\oplus$ to incorporate a finite set of states, so that $\mathcal{SR}_\oplus$ (resp. $(\mathcal{SR}_\&)_\oplus$) corresponds to usual – i.e. stateful – copyless (resp. single use restricted) SSTs. (Similar completions by certain colimits have been previously exploited [CP17b] within Colcombet and Petrişan's framework.)

The linchpin on which the various results previously mentioned rely is:

---

[6]The expression "single use restriction" already appears in much earlier automata models for regular tree functions: attributed tree transducers [BE00] and macro tree transducers [EM99]. This suggests that some kind of linearity condition is at work in those models, though we have not investigated this point further.

[7]Alternative options to restore enough expressive powers are copyless BRTTs with regular look-ahead (considered in [AD17, §3.4] for streaming transducers over unranked trees) or preprocessing by MSO relabelings [BD20, §4]. Beware: in the latter reference, the term "single use" refers to copyless assignments.

[8]There are also connections between categories and algebraic language theory [Til87], which however seemed less relevant to our work here.

Theorem 1.3. $(\mathcal{SR}_\&)_\oplus$ *is a* symmetric monoidal closed *category (SMCC).*

On the one hand, from the point of view of categorical automata theory, an SMCC provides a setting in which constructions relying on *function spaces* (i.e. internal homsets) can be carried out (this is typically the case when one exploits the finiteness of Q^Q for any finite set of states Q). This is the case for our composition result, whose general version is stated over arbitrary SMCCs. While $(\mathcal{SR}_\&)_\oplus$ is an SMCC, $\mathcal{SR}_\oplus$ is *not*, and this explains why composing copyless SSTs (that correspond to $\mathcal{SR}_\oplus$) directly is difficult: instead, a detour through $(\mathcal{SR}_\&)_\oplus$ allows us to apply Theorem 1.3. This move from $\mathcal{SR}_\oplus$ to $(\mathcal{SR}_\&)_\oplus$ is made possible by our abstract determinization argument, which itself relies on the existence of *some* (but not all) function spaces in $\mathcal{SR}_\oplus$.

On the other hand, for the programming language theorist, the notion of SMCC is an axiomatization of the *denotational semantics* for the "purely linear" fragment of our $\lambda\ell^{\oplus\&}$-calculus. We can therefore apply a *semantic evaluation* argument, following a long tradition in implicit complexity (cf. [HK96, Ter12]), to deduce Theorem 1.1 from Theorem 1.3; similarly, Theorem 1.2 follows from the monoidal closure of a category $(\mathcal{TR}_\&)_\oplus$ for trees. (Semantics of linear logic have also been applied to *higher-order recursion schemes*, a topic at the interface with automata, in [Gre16, Mel17, CM19], as well as to the purely automata-theoretic *Church synthesis problem* in some publications [PR18, PR19] coauthored by the present paper's third author.) That said, to create suitable conditions for semantic evaluation, a quite lengthy syntactic analysis is required, with the presence of positive connectives in the $\lambda\ell^{\oplus\&}$-calculus causing some complications.

At this point, we must mention the kinship of this $(\mathcal{SR}_\&)_\oplus$ with one of the earliest denotational models of linear logic, the *Dialectica categories* [dP89] (originating as a categorical account of Gödel's "Dialectica" functional interpretation [Göd58]). Composing the *free finite coproduct completion* $(-)_\oplus$ with its dual product completion $(-)_\&$ is indeed reminiscent of a factorization into free sums and free products of a generalized Dialectica construction [Hof11]. Thus, Theorem 1.3 holds for reasons similar to those for the monoidal closure of Dialectica categories (with a function space formula that resembles the interpretation of implication in [Göd58]). Such Dialectica-like structures have appeared in quite varied contexts in the past few years, such as *lenses* from functional programming and *compositional game theory* (see e.g. [Hed18, §4] for both), and, more in line with the topic of this paper, the aforementioned works on Church's synthesis [PR18, PR19] (and a closely related work on automata over infinite trees [Rib20]).

To wrap up this introduction, let us mention that as a bridge between $\lambda\ell^{\oplus\&}$-definability and those automata models, we also define within our categorical framework a notion of transducer whose memory is made up of (purely linear) $\lambda\ell^{\oplus\&}$-terms. This idea of using linear λ-terms inside a transducer model also appears in a recent characterization of regular tree functions [GLS20].

Contents

2. Preliminaries

2.1. **Notations & elementary definitions.**

2.1.1. *Sets and categories.* The cardinality of a set X is written $|X|$. We sometimes consider a family $(x_i)_{i \in I}$ as a map $i \mapsto x_i$, which amounts to treating $\prod_{i \in I} X_i$ as a dependent product. Consistently with this, we make use of the dependent sum operation

$$\sum_{i \in I} X_i = \{(i, x) \mid i \in I, \, x \in X_i\}$$

We (seldom) write numerals n for the underlying sets $\{0, \ldots, n-1\}$ for conciseness.

Given a category $\mathcal{C}$, we write $\mathsf{Obj}(\mathcal{C})$ for its class of objects and $\mathsf{Hom}_{\mathcal{C}}(A, B)$ for the set of arrows (or morphisms) from A to B (for $A, B \in \mathsf{Obj}(C)$). The composition of two morphisms $f \in \mathsf{Hom}_{\mathcal{C}}(A, B)$ and $g \in \mathsf{Hom}_{\mathcal{C}}(B, C)$ is denoted by $g \circ f$. Following the traditional notations of linear logic, products and coproducts will be customarily written using '&' and '$\oplus$' respectively – except in the category of sets where we use the notations '$\times$' and '$+$' as usual – and we reserve $\top$ for the terminal object. We sometimes use basic combinators such as $\langle - \rangle / [-]$ for pairing/copairing and π_i / in_i for projections/coprojections. With these notations, recall that the binary coproduct of sets is the tagged union

$$X + Y = \{\mathsf{in}_1(x) \mid x \in X\} \cup \{\mathsf{in}_2(y) \mid y \in Y\}$$

The injection $\mathsf{in}_1 / \mathsf{in}_2$ may be omitted by abuse of notation when it is clear from the context, that is, for $x \in X$, we allow ourselves to write x for $\mathsf{in}_1(x)$ when it is understood that this refers to an element of $X + Y$.

Finally, if we are given a binary operation $\square$ over the objects of a category, we freely use the corresponding "I-ary" operation, with a notation of the form $\square_{i \in I} A_i$, over families indexed by a finite set I. Concretely speaking, this depends on a fixed total order over $I = \{i_1 < \ldots < i_{|I|}\}$ to unfold as $A_{i_1} \square (A_{i_2} \square (\ldots \square A_{i_{|I|}}) \ldots)$ – for convenience, the reader may consider that a choice of such an order for every finite set is fixed once and for all for the rest of the paper. In practice, the particular order does not matter since we will deal with operations $\square \in \{\oplus, \&, \otimes, \ldots\}$ that are symmetric in a suitable sense. Those operations also have units (i.e. identity elements), giving a canonical meaning to $\bigoplus_{i \in \varnothing} A_i$, $\&_{i \in \varnothing} A_i$, etc.

Finally, as is usual when dealing with categories, we sometimes allow ourselves to implicitly use the axiom of choice for classes to pick objects determined by their universal properties to build functors (for instance, given an object A in a category $\mathcal{C}$ with cartesian products, we shall speak of the functor $- \& A$ without first mentioning that a choice of cartesian products $X \& A$ exists for for every X in $\mathcal{C}$). This is merely for convenience; the reader may check that in all of our concrete examples of interest, canonical choices can be made without appealing to choice.

2.1.2. *Strings and ranked trees.* Alphabets designate finite sets and are written using the variable names Σ, Γ. The set of strings (or words) over an alphabet Σ is denoted by Σ^*. The concatenation of two strings $u, v \in \Sigma^*$ is written uv (or sometimes $u \cdot v$ for clarity); recall that Σ^* endowed with this operation is the free monoid over the set of generators Σ, and its identity element is the empty string ε. We write $|w|$ for the length of a word $w \in \Sigma^*$, and given a letter $c \in \Sigma$, the notation $|w|_c$ refers to the number of occurrences of c in w.
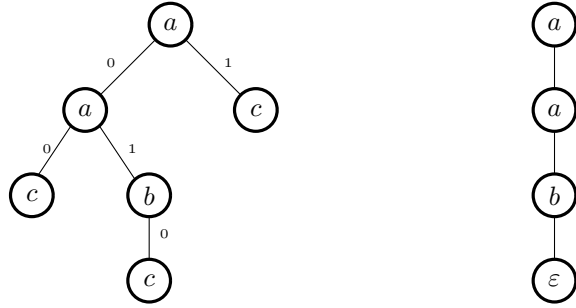
Figure 1: Graphical representations of the tree $a(a(c, b(c)), c)$ over the ranked alphabet $\{a : \{0, 1\}, b : \{0\}, c : \varnothing\}$ (left) and of the word $aab \in \{a, b\}^*$ seen as an element of $\mathbf{Tree}(\overline{\{a, b\}}) = \mathbf{Tree}(\{a : \{*\}, b : \{*\}, \varepsilon : \varnothing\})$ (right).

Ranked alphabets are pairs (Σ, ar) such that Σ is an alphabet and the *arity* ar is a family of finite sets[9] indexed by Σ; they are written using $\boldsymbol{\Sigma}, \boldsymbol{\Gamma}$. We may write $\{a_1 : A_1, \ldots, a_n : A_n\}$ for the ranked alphabet $(\{a_1, \ldots, a_n\}, \mathrm{ar})$ with $\mathrm{ar}(a_i) = A_i$.

Given a ranked alphabet $\boldsymbol{\Sigma}$, the set $\mathbf{Tree}(\boldsymbol{\Sigma})$ of trees/terms over a ranked alphabet $\boldsymbol{\Sigma}$ is defined as usual: if a is a letter of arity X in $\boldsymbol{\Sigma}$ and t a family of $\boldsymbol{\Sigma}$-trees, we write $a(t)$ for the corresponding tree. Examples of such trees are pictured in Figure 1.

Remark 2.1. Given an alphabet Σ, define $\overline{\Sigma}$ to be the ranked alphabet $(\Sigma + \{\varepsilon\}, \mathrm{ar})$ such that $\mathrm{ar}(\mathsf{in}_1(a)) = \{*\}$ and $\mathrm{ar}(\mathsf{in}_2(\varepsilon)) = \varnothing$. This gives a isomorphism $\mathbf{Tree}(\overline{\Sigma}) \simeq \Sigma^*$, illustrated on the right of Figure 1.

2.2. **Transducer models for regular functions over strings and trees.**

2.2.1. *Strings.* Let us first recall the machine model that provides our reference definition for regular functions: copyless streaming string transducers [AČ10] (SSTs). A SST is an automaton whose internal memory contains, additionally to its control state, a finite number of string-valued registers. It processes its input in a single left-to-right pass. Each time a letter is read, the contents of the registers may be recombined by concatenation to determine the new register values. Formally:

Definition 2.2. Fix a finite alphabet Γ. Let R and S be finite sets; we shall consider their elements to be "register variables".

A Γ-*register transition*[10] from R to S is a function $t : S \to (\Gamma + R)^*$. Such a transition is said to be *copyless* when for every $r \in R$, there is at most one occurrence of $\mathsf{in}_2(r)$ among all the words $t(s)$ for $s \in S$ (i.e. when $\sum_{s \in S} |t(s)|_{\mathsf{in}_2(r)} \leq 1$). We write $[R \to_{\mathcal{SR}(\Gamma)} S]$ for the set of copyless Γ-register transitions from R to S, or $[R \to_{\mathcal{SR}} S]$ when Γ is clear from context.

Let $t \in [R \to_{\mathcal{SR}(\Gamma)} S]$. For $x = (x_r)_{r \in R} \in (\Gamma^*)^R$ and $s \in S$, we denote by $(t^\dagger(x))_s \in \Gamma^*$ the word obtained from $t(s)$ by substituting every occurrence of a register variable $r \in R$ by

---

[9]This is slightly non-standard; the more usual notion would be that ar be only a family of numbers $\Sigma \to \mathbb{N}$. To talk more freely about function spaces, we work with finite sets rather than numbers in several instances, which motivates departing from the usual notion.

[10]Sometimes called a *substitution* in the literature, e.g. in [AFT12, DJR18].
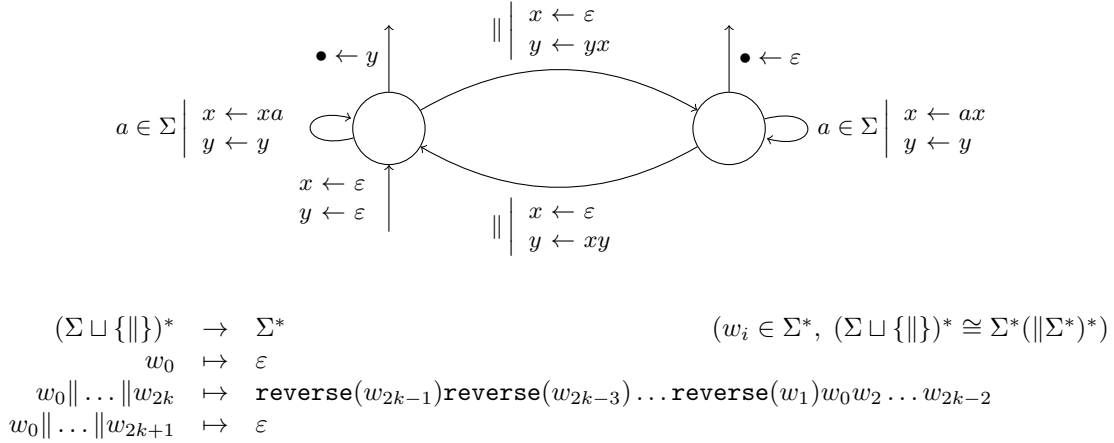
$$
\begin{aligned}
(\Sigma \sqcup \{\|\})^* &\to \Sigma^* & (w_i \in \Sigma^*, \ (\Sigma \sqcup \{\|\})^* \cong \Sigma^*(\|\Sigma^*)^*) \\
w_0 &\mapsto \varepsilon \\
w_0\|\ldots\|w_{2k} &\mapsto \mathtt{reverse}(w_{2k-1})\mathtt{reverse}(w_{2k-3})\ldots\mathtt{reverse}(w_1)w_0w_2\ldots w_{2k-2} \\
w_0\|\ldots\|w_{2k+1} &\mapsto \varepsilon
\end{aligned}
$$

Figure 2: An informal depiction of a SST and the induced map $(\Sigma \sqcup \{\|\})^* \to \Sigma^*$.

the string x_r – formally, by applying the morphism of free monoids $(\Gamma + R)^* \to \Gamma^*$ that maps $\mathsf{in}_1(c)$ to c and $\mathsf{in}_2(r)$ to x_r. This defines a set-theoretic map $t^\dagger : (\Gamma^*)^R \to (\Gamma^*)^S$, describing *how t acts on tuples of strings*.

For instance, $t : z \mapsto axby$ (where we omitted $\mathsf{in}_1/\mathsf{in}_2$) is in $[\{x, y\} \to_{\mathcal{SR}(\{a,b\})} \{z\}]$ (it is copyless since x and y appear only once), and $t^\dagger(x \mapsto b, \ y \mapsto aa) = (z \mapsto abbaa)$.

Definition 2.3 ([AČ10])**.** A (deterministic) *copyless streaming string transducer* (SST) with input alphabet Σ and output alphabet Γ is a tuple $\mathcal{T} = (Q, q_0, R, \delta, i, o)$ where

- Q is a finite set of states and $q_0 \in Q$ is the initial state;
- R is a finite set of registers;
- $\delta : \Sigma \times Q \to Q \times [R \to_{\mathcal{SR}(\Gamma)} R]$ is the transition function;
- $i \in [\varnothing \to_{\mathcal{SR}(\Gamma)} R] \cong (\Gamma^*)^R$ describes the initial register values;
- $o : Q \to [R \to_{\mathcal{SR}(\Gamma)} \{\bullet\}]$ (where $\bullet$ is an arbitrary element) describes how to recombine the final values of the registers, depending on the final state, to produce the output.

(The SSTs that we consider in this paper are always copyless.)

The function $\Sigma^* \to \Gamma^*$ computed by $\mathcal{T}$ maps an input string $w = w_1 \ldots w_n \in \Sigma^*$ to the output string $o(q_n)^\dagger \circ t_n^\dagger \circ \ldots \circ t_1^\dagger \circ i^\dagger(\varnothing) \in \Gamma^*$ where

- the empty family $\varnothing$ is indeed the unique element of $(\Gamma^*)^\varnothing$;
- the codomain $(\Gamma^*)^{\{\bullet\}}$ of $o^\dagger(q_n)$ is identified with Γ^*;
- the register transitions $(t_i)_{1 \le i \le n}$ and the final state $q_n \in Q$ are inductively defined, starting from the fixed initial state q_0, by $(q_i, t_i) = \delta(w_i, q_{i-1})$.

The functions that can be computed by copyless SSTs are called *regular string functions*.

Example 2.4. Let us describe a simple copyless SST with $\Sigma = \Gamma$ and a single state, so that $\delta : \Sigma \to [R \to_{\mathcal{SR}} R]$. We take $R = \{x, y\}$; both x and y are initialized with the empty string ε, and the register transition $t_c = \delta(c) \in [R \to_{\mathcal{SR}} R]$ associated to $c \in \Sigma$ is $(x \mapsto xc, \ y \mapsto cy)$ (to be pedantic, one should write $(x \mapsto \mathsf{in}_2(x)\mathsf{in}_1(c), \ y \mapsto \mathsf{in}_1(c)\mathsf{in}_2(y))$). Then for $w = w[1]\ldots w[n] \in \Sigma^*$, we have:

$$
t_{w[n]}^\dagger \circ \cdots \circ t_{w[1]}^\dagger(x \mapsto \varepsilon, \ y \mapsto \varepsilon) = (x \mapsto w, \ y \mapsto \mathtt{reverse}(w))
$$

If we take $o = xy$ (via the canonical isomorphism $[R \to_{\mathcal{SR}(\Gamma)} 1]^{\{q\}} \cong (\Gamma + R)^*$), the function computed by the SST is $w \in \Sigma^* \mapsto w \cdot \texttt{reverse}(w)$.

Figure 2 shows a more sophisticated SST with two states and the associated regular function.

2.2.2. *Trees as output.* Let us briefly discuss the challenges that arise when extending this model to handle *ranked trees* instead of strings. We will revisit this material in more detail in Section 5.

The notion of *regular tree-to-tree function* is defined by generalizing the characterization of regular string functions by Monadic Second-Order Logic [EM99, BE00, EH01], in a way that is compatible with the above isomorphism. There are two orthogonal difficulties that have to be overcome to define a SST-like model for regular tree functions: one comes from producing trees as output, while the other comes from taking trees as input. *Bottom-up ranked tree transducers*[11] (BRTTs) [AD17] (and the similar model of register tree transducers in [BD20, §4]) provide solutions for both.

String-to-tree regular functions require a modification of the kind of data stored in the registers of an SST. Tree-valued registers are *not enough*, for the following reasons: to recover the flexibility of string concatenation, one should be able to perform operations such as grafting the root of some tree to a leaf of another tree; but then the latter should be a tree with a distinguished leaf, serving as a "hole" waiting to be substituted by a tree. (This is fundamental in the theory of forest algebras, which proposes various counterparts for trees to the monoid of strings with concatenation, see [Boj].) By allowing both trees and "one-hole trees" as register values, with the appropriate notion of copyless register transition (cf. Section 5.3), one gets the *copyless streaming string-to-tree transducers*, whose expressive power corresponds exactly to the regular functions [AD17, Theorem 3.16].

2.2.3. *Trees as inputs.* To compute tree-to-tree regular functions, the first idea would be to blend the notion of copyless SST with the classical bottom-up *tree automata*. One would then get *copyless bottom-up ranked tree transducers*. However, this model is believed to be too weak to express all regular tree functions (even in the case of tree-to-string functions). An explicit counterexample is conjectured in [AD17, §2.3], in the case of regular functions on unranked trees; we adapt it here into a function from ranked trees to strings.

In the example below, for a ranked letter a of arity $2 = \{\triangleleft, \triangleright\}$, we use the abbreviation $a(t, u)$ for $a(\triangleleft \mapsto t, \triangleright \mapsto u)$.

Example 2.5 ("Conditional swap"). Define $f : \textbf{Tree}(\{a : 2, b : 2, c : \varnothing\}) \to \{a, b, c\}^*$ by

$$f(a(t, u)) = f(u) \cdot a \cdot f(t) \qquad f(t) = \texttt{inorder}(t) \text{ if } t \text{ doesn't match the previous pattern}$$

where $\texttt{inorder}$ prints the nodes of t following a depth-first in-order traversal. In other words, $f = \texttt{inorder} \circ g$ where $g(a(t, u)) = a(g(u), g(t))$ and $g(t) = t$ otherwise (i.e. when the root of t is either b or c).

Conjecture 2.6 (adapted from [AD17, §2.3]). The above f cannot be computed by a copyless BRTT.

---

[11]The name "streaming tree transducer" is used in [AD17] for a transducer model operating over unranked trees. BRTTs are proposed in the same paper as a simpler, equally expressive variant for the special case of ranked trees.

One must then allow more register transitions than the copyless ones. This cannot be done haphazardly, for arbitrary register transitions would lead to a much larger class of functions than regular tree functions. Alur and D'Antoni call their relaxed condition [AD17] the *single use restriction* (it will only be formally defined in Section 5.4); the following single-state BRTT for f provides a typical example of the new possibilities allowed.

Example 2.7 (Non-copyless BRTT for conditional swap). Take $R = \{x, y\}$, initialized at the c-labeled leaves with $(x \mapsto c, y \mapsto c)$. At a subtree $a(u, v)$, we need to combine the registers $x_\triangleleft, y_\triangleleft$ (resp. $x_\triangleright, y_\triangleright$) coming from the left (resp. right) child u (resp. v) to produce the values of the registers x, y at this node: this is performed by a register transition

$$t_a \in [\{x_\triangleleft, y_\triangleleft, x_\triangleright, y_\triangleright\} \to_{\mathcal{SR}} \{x, y\}] \qquad t_a(x) = x_\triangleright a x_\triangleleft \qquad t_a(y) = y_\triangleleft a y_\triangleright$$

The idea is that the register values produced by processing a subtree u are $f(u)$ for x and $\mathtt{inorder}(u)$ for y. The register transition for a b-labeled node is then $t_b(x) = t_b(y) = y_\triangleleft b y_\triangleright$, reflecting the fact that $f(b(u, v)) = \mathtt{inorder}(b(u, v))$.

This t_b is not copyless since $y_\triangleleft$ occurs twice: once in $t_b(x)$ and once in $t_b(y)$. The observation at the heart of the single use restriction is that the values of x and y for a given subtree can never be combined in the same expression in the remainder of the BRTT's run, so that allowing this duplication of $y_\triangleleft$ will never lead to having two copies of $y_\triangleleft$ inside the value of a single register. We will see much later in Example 5.20 that this BRTT is indeed single-use-restricted.

2.3. The $\lambda\ell^{\oplus\&}$-calculus, encodings of strings/trees, and definability of functions.

2.3.1. *Types & terms.* We consider a linear λ-calculus which we dub the $\lambda\ell^{\oplus\&}$-*calculus*, based (via the Curry–Howard correspondence) on propositional intuitionistic linear logic with both multiplicative and additive connectives (IMALL) together with a base linear type $\mathtt{o}$. The grammar of types is as follows:

$$\tau, \sigma ::= \mathtt{o} \mid \tau \multimap \sigma \mid \tau \otimes \sigma \mid \mathbf{I} \mid \tau \to \sigma \mid \tau \,\&\, \sigma \mid \tau \oplus \sigma \mid \top \mid 0$$

A typing context Ψ is a finite set of declarations $x_1 : \tau_i, \ldots, x_k : \tau_k$ where the x_i are pairwise distinct variables (which constitute the set of free variables of Ψ) and the τ_i are types. Typed $\lambda\ell^{\oplus\&}$-terms are given in Figure 3 along with the inductive definition of the typing judgment $\Psi;\ \Delta \vdash t : \tau$, where Ψ and Δ are contexts (with disjoint sets of free variables), τ is a type and t is a term. In such a judgment, Ψ is called the *non-linear* context and Δ the *linear* context; the basic idea is that variables in Ψ may be used arbitrarily many times, while those in Δ must be used *exactly once*. This is formally more restrictive than an *affineness condition*, where we would rather restrict variables in Δ to occur *at most once* in t.

In practice, $\lambda\ell^{\oplus\&}$ is not less expressive than its affine variant[12] since it features additives: the basic idea is that the affineness can be encoded at the level of types by using the linear type $\tau \,\&\, \mathbf{I}$ instead of the affine type τ (as argued for instance in [Gir95, §1.2.1]).

The simply typed λ-calculus admits an embedding into $\lambda\ell^{\oplus\&}$. Conversely, there is a mapping from $\lambda\ell^{\oplus\&}$ to the simply typed λ-calculus with products and sums by "forgetting

---

[12]Which would be obtained by adjoining the following *weakening rule* to the system presented in Figure 3:

$$\frac{\Psi;\ \Delta \vdash t : \tau}{\Psi;\ \Delta,\ \Delta' \vdash t : \tau}$$

$$\overline{\Psi;\ x:\tau \vdash x:\tau} \qquad\qquad \overline{\Psi, x:\tau;\ \cdot \vdash x:\tau}$$

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

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

$$\frac{\Psi;\ \Delta \vdash t:\tau \qquad \Psi;\ \Delta' \vdash u:\sigma}{\Psi;\ \Delta,\ \Delta' \vdash t\otimes u:\tau\otimes\sigma} \qquad \frac{\Psi;\ \Delta' \vdash u:\tau\otimes\sigma \qquad \Psi;\ \Delta, x:\tau, y:\sigma \vdash t:\kappa}{\Psi;\ \Delta,\ \Delta' \vdash \mathsf{let}\ x\otimes y = u\ \mathsf{in}\ t:\kappa}$$

$$\overline{\Psi;\ \cdot \vdash ():\mathbf{I}} \qquad \frac{\Psi;\ \Delta \vdash t:\mathbf{I} \qquad \Psi;\ \Delta' \vdash u:\tau}{\Psi;\ \Delta,\ \Delta' \vdash \mathsf{let}\ () = t\ \mathsf{in}\ u:\tau}$$

$$\frac{\Psi;\ \Delta \vdash t:\tau \qquad \Psi;\ \Delta \vdash u:\sigma}{\Psi;\ \Delta \vdash \langle t,u\rangle : \tau \,\&\, \sigma} \qquad \frac{\Psi;\ \Delta \vdash t:\tau \,\&\, \sigma}{\Psi;\ \Delta \vdash \pi_1(t):\tau} \qquad \frac{\Psi;\ \Delta \vdash t:\tau \,\&\, \sigma}{\Psi;\ \Delta \vdash \pi_2(t):\sigma}$$

$$\frac{\Psi;\ \Delta \vdash t:\tau}{\Psi;\ \Delta \vdash \mathsf{in}_1(t):\tau\oplus\sigma} \qquad \frac{\Psi;\ \Delta \vdash t:\sigma}{\Psi;\ \Delta \vdash \mathsf{in}_2(t):\tau\oplus\sigma}$$

$$\frac{\Psi;\ \Delta, x:\tau \vdash u:\kappa \qquad \Psi;\ \Delta, x:\tau \vdash v:\kappa \qquad \Psi;\ \Delta' \vdash t:\tau\oplus\sigma}{\Psi;\ \Delta,\Delta' \vdash \mathsf{case}(t, x.u, x.v):\kappa}$$

$$\overline{\Psi;\ \Delta \vdash \langle\rangle : \top} \qquad \frac{\Psi;\ \Delta \vdash t:0}{\Psi;\ \Delta,\ \Delta' \vdash \mathsf{abort}(t):\tau}$$

Figure 3: Typing rules of $\lambda\ell^{\oplus\&}$.

β-equivalence		
	$(\lambda x.t)\,u \ =_\beta\ t[u/x]$	$(\lambda^! x.t)\,u \ =_\beta\ t[u/x]$
	$\pi_1(\langle t,u\rangle) \ =_\beta\ t$	$\pi_2(\langle t,u\rangle) \ =_\beta\ u$
	$\mathsf{case}(\mathsf{in}_1(t), x.u, x.v) \ =_\beta\ u[t/x]$	$\mathsf{case}(\mathsf{in}_2(t), x.u, x.v) \ =_\beta\ v[t/x]$
	$\mathsf{let}\ x\otimes y = t\otimes u\ \mathsf{in}\ v \ =_\beta\ v[t/x][u/y]$	$\mathsf{let}\ () = ()\ \mathsf{in}\ t \ =_\beta\ t$

η-equivalence		
	$\lambda x.t\,x \ =_\eta\ t$	$\lambda^! x.t\,x \ =_\eta\ t$
	$\mathsf{let}\ x\otimes y = t\ \mathsf{in}\ u[x\otimes y/z] \ =_\eta\ u[t/z]$	$\langle \pi_1(t), \pi_2(t)\rangle \ =_\eta\ t$
	$\mathsf{let}\ x\otimes y = t\ \mathsf{in}\ v[u/z] \ =_\eta\ v[\mathsf{let}\ x\otimes y = t\ \mathsf{in}\ u/z]$	$x \ =_\eta\ \langle\rangle$
	$\mathsf{let}\ () = t\ \mathsf{in}\ u[()/z] \ =_\eta\ u[t/z]$	
	$\mathsf{let}\ () = t\ \mathsf{in}\ v[u/z] \ =_\eta\ v[\mathsf{let}\ () = t\ \mathsf{in}\ u/z]$	
	$\mathsf{case}(t, x.u[\mathsf{in}_1(x)/z], y.u[\mathsf{in}_2(y)/z]) \ =_\eta\ u[t/z]$	$\mathsf{abort}(t) \ =_\eta\ u$

Figure 4: Equations for $\lambda\ell^{\oplus\&}$-terms (relating terms that have matching types).

linearity" (and replacing the tensorial product eliminator $\mathsf{let}\ x\otimes y = t\ \mathsf{in}\ u$ by the variant based on projections $u[\pi_1(t)/x, \pi_2(t)/y]$).

As usual, we identify $\lambda\ell^{\oplus\&}$-terms up to renaming of bound variables (α-equivalence) and admit the standard definition of the capture-avoiding substitution. For the purpose of this paper, since we are not interested in the fine details of their operational semantics, we usually consider $\lambda\ell^{\oplus\&}$-terms up to $\beta\eta$-equivalence $=_{\beta\eta}$ as generated by the equations in Figure 4 and congruence. Note that those equations are implicitly typed and that typing is invariant under $\beta\eta$-equivalence.

Much like any λ-calculus, $\lambda\ell^{\oplus\&}$ can be seen as a programming language by considering a reduction relation $\to_{\beta\varepsilon}$, which happens to be included in $=_{\beta\eta}$. One property that we shall use is that $\lambda\ell^{\oplus\&}$ is *normalizing*, i.e., that the relation $\to_{\beta\varepsilon}$ is terminating. This allows to consider terms of very specific shape when working up to $\beta\eta$. While the argument is routine, we need this result, as well as a fine-grained understanding of the normal forms to discuss further preliminary syntactic lemmas, so we give an outline in Appendix B.

We now isolate an important class of types and terms for the sequel.

Definition 2.8. We call a type *purely linear* if it does not have any occurrence of the '$\to$' connective. A $\lambda\ell^{\oplus\&}$-term t is also called *purely linear* if there is a typing derivation $\Psi; \Delta \vdash t : \tau$ where any type occurring must be purely linear.

Intuitively, purely linear terms are those which are not allowed to duplicate any arguments involving $\mathbb{o}$. For any type derivation $\Psi; \Delta \vdash t : \tau$, if the types occurring in Ψ and Δ, as well as τ, are purely linear, then so is t; this is a consequence of normalization.

2.3.2. *Church encodings.* In order to discuss string (and tree) functions in $\lambda\ell^{\oplus\&}$, we need to discuss how they are encoded. Recall that in the pure (i.e. untyped) λ-calculus, the canonical way to encode inductive types[13] is via *Church encodings.* Such encodings are typable in the simply-typed λ-calculus. For instance, for natural numbers and strings over $\{a, b\}$, writing $\underline{w}$ for the Church encoding of w, we have

$$\mathrm{Nat}^! \;=\; (\mathbb{o} \to \mathbb{o}) \to \mathbb{o} \to \mathbb{o} \qquad\qquad \mathrm{Str}^!_{\{a,b\}} \;=\; (\mathbb{o} \to \mathbb{o}) \to (\mathbb{o} \to \mathbb{o}) \to \mathbb{o} \to \mathbb{o}$$

$$\underline{3} = \lambda^! s.\lambda^! z.s\,(s\,(s\,z)) \qquad\qquad \underline{aab} = \lambda^! a.\lambda^! b.\lambda^! \varepsilon.a\,(a\,(b\,\varepsilon))$$

Conversely, one may show that any closed simply typed λ-term of type $\mathrm{Nat}^!$ (resp. $\mathrm{Str}^!_{\{a,b\}}$) is $\beta\eta$-equivalent to the Church encoding of some number (resp. string). In the rest of this paper, we will use a less common, but more precise $\lambda\ell^{\oplus\&}$-type for Church encodings of strings of trees, first introduced in [Gir87, §5.3.3].

Definition 2.9. Let Σ be an alphabet. We define Str_Σ as $(\mathbb{o} \multimap \mathbb{o}) \to \ldots \to (\mathbb{o} \multimap \mathbb{o}) \to \mathbb{o} \to \mathbb{o}$ where there are $|\Sigma|$ occurrences of $\mathbb{o} \multimap \mathbb{o}$. Note in particular that thanks to the isomorphism[14]

---

[13]Including the natural numbers, if one wants for instance to show that the untyped λ-calculus captures all computable functions. We should also mention that the generalization of Church encodings to trees is actually due to Böhm and Berarducci [BB85].

[14]We keep this notion informal, but suffices to say that this is intended to be definable internally to $\lambda\ell^{\oplus\&}$.

$(A \,\&\, B) \to C \;\cong\; A \to B \to C$ (non-linear currying), we have[15]

$$\mathrm{Str}_\Sigma \;\cong\; \left(\underset{a\in\Sigma}{\&}(\mathbb{o} \multimap \mathbb{o})\right) \to \mathbb{o} \to \mathbb{o}$$

It can be checked that Str_Σ has the same (up to $\beta\eta$ equality) closed inhabitants as the usual $\mathrm{Str}_\Sigma^!$ presented above, but one should keep in mind that this choice is not entirely innocuous. It is in large part motivated by our main result (Theorem 1.1), which might no longer hold when taking $\mathrm{Str}_\Sigma^!$ instead of Str_Σ.

This situation generalizes to trees. For instance, the Church encoding of the tree depicted in Figure 1 is $\lambda^! a.\lambda^! b.\lambda^! c.\ a\ (a\ c\ (b\ c))\ c\ :\ (\mathbb{o} \to \mathbb{o} \to \mathbb{o}) \to (\mathbb{o} \to \mathbb{o}) \to \mathbb{o} \to \mathbb{o}$.

Definition 2.10. Given a ranked alphabet $\mathbf{\Sigma} = (\Sigma, \mathrm{ar})$, the $\lambda\ell^{\oplus\&}$ type $\mathrm{Tree}_{\mathbf{\Sigma}}$ is defined as

$$\mathrm{Tree}_{\mathbf{\Sigma}} \;=\; (\mathbb{o} \multimap \dots \multimap \mathbb{o}) \to \dots \to (\mathbb{o} \multimap \dots \multimap \mathbb{o}) \to \mathbb{o}$$

where there are $|\Sigma|$ top-level arguments, and, within the component corresponding to the letter $a \in \Sigma$, there are $|\mathrm{ar}(a)|$. In other words, we have the isomorphism

$$\mathrm{Tree}_{\mathbf{\Sigma}} \;\cong\; \left(\underset{a\in\Sigma}{\&}(\mathbb{o}^{\otimes\mathrm{ar}(a)} \multimap \mathbb{o})\right) \to \mathbb{o}$$

Remark 2.11. The isomorphism of Remark 2.1 translates to an equality $\mathrm{Str}_\Sigma = \mathrm{Tree}_{\overline{\Sigma}}$.

Church encodings give a map from trees in $\mathbf{Tree}(\mathbf{\Sigma})$ to $\lambda\ell^{\oplus\&}$-terms of type $\mathrm{Tree}_{\mathbf{\Sigma}}$ in the empty context. This map is in fact a bijection if terms are considered up to $\beta\eta$-equality: normalization of the $\lambda\ell^{\oplus\&}$-calculus enforces surjectivity, and one may use a set-theoretic semantics of $\lambda\ell^{\oplus\&}$ to build a left inverse (see the proof of Proposition C.5 in the appendix for further details).

2.3.3. *Computing with Church encodings.* We are now ready to give our notion of computation for our string (and tree) functions. First, we need an operation of type substitution in $\lambda\ell^{\oplus\&}$, which allow to substitute an arbitrary type κ for $\mathbb{o}$.

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

Type substitution extends in the obvious way to typing contexts as well, and even to *typing derivations*, so that

$$\Psi;\ \Delta \vdash t : \tau \qquad \Rightarrow \qquad \Psi[\kappa];\ \Delta[\kappa] \vdash t : \tau[\kappa]$$

In particular, it means that a Church encoding $t : \mathrm{Tree}_{\mathbf{\Sigma}}$ is also of type $\mathrm{Tree}_{\mathbf{\Sigma}}[\kappa]$ for any type κ. This ensures that the following notion of definable tree functions (strings being a special case) in the $\lambda\ell^{\oplus\&}$-calculus makes sense.

---

[15]In this encoding, the unique constructor of arity 0 is treated non-linearly, while in the prequel [NP20], it was treated linearly. We chose non-linearity here in order to be consistent with the definition for ranked trees (cf. Remark 2.11): indeed, while strings have a single end-marker, trees may have multiple leaves, so non-linearity is necessary in their case. This apparent inconsistency with our previous work is actually unproblematic as both string encodings are interconvertible, see e.g. [NP20, Remark 5.7].

Definition 2.12. A function $f : \mathbf{Tree}(\Sigma) \to \mathbf{Tree}(\Gamma)$ is called $\lambda\ell^{\oplus\&}$-*definable* when there exists a *purely linear* type κ together with a $\lambda\ell^{\oplus\&}$-term

$$\mathtt{f} \quad : \quad \mathrm{Tree}_\Sigma[\kappa] \multimap \mathrm{Tree}_\Gamma$$

such that f and $\mathtt{f}$ coincide up to Church encoding; i.e., for every tree $t \in \mathbf{Tree}(\Sigma)$

$$\underline{f(t)} \quad =_{\beta\eta} \quad \mathtt{f}\ \underline{t}$$

In particular, a string function $\Sigma^* \to \Gamma^*$ is $\lambda\ell^{\oplus\&}$-definable when the corresponding unary tree function $\mathbf{Tree}\left(\overline{\Sigma}\right) \to \mathbf{Tree}\left(\overline{\Gamma}\right)$ (cf. Remark 2.11) is $\lambda\ell^{\oplus\&}$-definable. Note that

$$\mathrm{Tree}_{\overline{\Sigma}}[\kappa] \multimap \mathrm{Tree}_{\overline{\Gamma}} \quad = \quad \mathrm{Str}_\Sigma[\kappa] \multimap \mathrm{Str}_\Gamma$$

Remark 2.13. Once again, our set-up, summarized in Definition 2.12, is biased toward making our main theorem true; there are many non-equivalent alternatives which also make perfect sense. For instance, changing the following would be reasonable:

- allow κ to be be arbitrary (i.e. to contain !) or with some restrictions.
- consider the non-linear arrow $\to$ instead of $\multimap$ at the toplevel.
- change the type of Church encodings (recall the distinction $\mathrm{Str}_\Sigma^!/\mathrm{Str}_\Sigma$).

Most of these alternatives share the good structural properties outlined below. Giving more automata-theoretic characterizations for those and comparing them lies beyond the scope of this paper, but would be interesting.

The two first choices above will turn out to have a clear operational meaning: the pure linearity of κ corresponds to single-use-restricted assignment (as mentioned in the introduction), whereas the use of the linear function arrow '$\multimap$' corresponds to the fact that a streaming tree transducer *traverses its input in a single pass*.

As our main theorems claim, $\lambda\ell^{\oplus\&}$-definable functions and regular functions coincide, so all our examples of regular functions can be coded in $\lambda\ell^{\oplus\&}$, as we show concretely below.

Example 2.14. The `reverse` function $\Sigma^* \to \Sigma^*$ from Example 2.4 is $\lambda\ell^{\oplus\&}$-definable. Supposing that we have $\Sigma = \{a_1, \ldots, a_k\}$, one $\lambda\ell^{\oplus\&}$-term that implements it is

$$\lambda s.\lambda^! a_1. \ldots .\lambda^! a_k.\lambda^! \varepsilon.\ s\ (\lambda x.\lambda z.x\ (a_1\ z)) \ldots (\lambda x.\ (a_k\ z))\ (\lambda x.x)\ \varepsilon\ :\ \mathrm{Str}_\Sigma[\mathbb{0} \multimap \mathbb{0}] \multimap \mathrm{Str}_\Sigma$$

Example 2.15. The SST of Figure 2 is computed by a $\lambda\ell^{\oplus\&}$-term of type $\mathrm{Str}_{\Sigma\sqcup\{\|\}}[\tau] \multimap \mathrm{Str}_\Sigma$ with $\tau = \mathrm{Bool} \otimes ((\mathbb{0} \multimap \mathbb{0})\ \&\ \mathbf{I}) \otimes ((\mathbb{0} \multimap \mathbb{0})\ \&\ \mathbf{I})$. Intuitively, Bool corresponds to the current state of the SST while each component $(\mathbb{0} \multimap \mathbb{0})\ \&\ \mathbf{I}$ corresponds to a register. Define the auxiliary terms $\delta : (\mathbb{0} \multimap \mathbb{0}) \multimap \tau \multimap \tau$, $\delta_\| : \mathbb{0} \multimap \tau \multimap \tau$ and $o : \mathbb{0} \multimap (\tau \multimap \tau) \multimap \mathbb{0}$ as Supposing that we have $\Sigma = \{a_1, \ldots, a_k\}$, and that the letter $\|$ corresponds to the first constructor in the input string, the $\lambda\ell^{\oplus\&}$-definition is given by

$$\lambda s.\lambda^! a_1. \ldots .\lambda^! a_k.\lambda^! \varepsilon.\ o\ (s\ \delta_\|\ (\delta\ a_1)\ \ldots\ (\delta\ a_k))$$

where the terms δ, $\delta_\|$ and o are defined in Figure 5.

Example 2.16. Consider the ranked alphabet $\mathbf{\Sigma} = \{a : 2, b : 2, c : \varnothing\}$ (where $2 = \{\triangleleft, \triangleright\}$) and the alphabet $\Gamma = \{a, b, c\}$. The conditional swap of Example 2.5 is $\lambda\ell^{\oplus\&}$-definable as a term of type

$$\mathrm{Tree}_\Sigma[(\mathbb{0} \multimap \mathbb{0})\ \&\ (\mathbb{0} \multimap \mathbb{0})] \to \mathrm{Str}_\Gamma$$

reminiscent of the BRTT given in Example 2.7. Observe the use of an *additive conjunction* '&' (that is not of the form $(-\ \&\ \mathbf{I})$ meant to make data discardable), reflecting the fact that this

$$\delta = \lambda a\, z.\ \mathsf{let}\ (b, z') = z\ \mathsf{in}$$
$$\mathsf{let}\ (x, y) = z'\ \mathsf{in}$$
$$\mathsf{if}\ b\ \mathsf{then}$$
$$(\mathtt{tt}, \langle \lambda u.\ \pi_1(x)\ (a\ u), \mathsf{let}\ () = \pi_2(y)\ \mathsf{in}\ \pi_2(x)\rangle, y)$$
$$\mathsf{else}$$
$$(\mathtt{ff}, \langle \lambda u.\ a\ (\pi_1(x)\ u), \mathsf{let}\ () = \pi_2(y)\ \mathsf{in}\ \pi_2(x)\rangle, y)$$

$$\delta_{\parallel} = \lambda z.\ \mathsf{let}\ (b, z') = z\ \mathsf{in}$$
$$\mathsf{let}\ (x, y) = z'\ \mathsf{in}$$
$$\mathsf{if}\ b\ \mathsf{then}$$
$$(\mathtt{ff}, \langle \lambda u.u, \mathsf{let}\ () = \pi_2(y)\ \mathsf{in}\ \pi_2(x)\rangle, \langle \lambda v.\pi_1(y)\ (\pi_1(x)\ u), \mathsf{let}\ () = \pi_1(x)\ \mathsf{in}\ \pi_2(y)\rangle)$$
$$\mathsf{else}$$
$$(\mathtt{tt}, \langle \lambda u.u, \mathsf{let}\ () = \pi_2(y)\ \mathsf{in}\ \pi_2(x)\rangle, \langle \lambda v.\pi_1(x)\ (\pi_1(y)\ u), \mathsf{let}\ () = \pi_1(x)\ \mathsf{in}\ \pi_2(y)\rangle)$$

$$o = \lambda \varepsilon\, z.\ \mathsf{let}\ (b, z') = z\ (\mathtt{tt}, \langle \lambda u.u, ()\rangle, \langle \lambda u.u\rangle)\ \mathsf{in}$$
$$\mathsf{let}\ (x, y) = z'\ \mathsf{in}$$
$$\mathsf{let}\ () = \pi_2(x)\ \mathsf{in}$$
$$\mathsf{if}\ b\ \mathsf{then}$$
$$\pi_1(y)\ \varepsilon$$
$$\mathsf{else}$$
$$\mathsf{let}\ () = \pi_2(y)\ \mathsf{in}\ \varepsilon$$

Figure 5: Auxiliary terms for Example 2.15 ($\mathtt{tt} = \mathsf{in}_1(())$, $\mathtt{ff} = \mathsf{in}_2()$ and $\mathsf{if}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v$ is a notation for $\mathsf{case}(t, x.\mathsf{let}\ () = x\ \mathsf{in}\ u, y.\mathsf{let}\ () = y\ \mathsf{in}\ v)$).

BRTT is single-use-restricted but not copyless. To wit, setting $\tau = (\mathbb{o} \multimap \mathbb{o})\ \&\ (\mathbb{o} \multimap \mathbb{o})$ and assuming free variables $a, b : \mathbb{o} \multimap \mathbb{o}$, define the auxiliary terms

$$\delta_a = \lambda l.\lambda r.\ \langle \pi_1(l) \circ a \circ \pi_1(r), \pi_1(r) \circ a \circ \pi_1(l)\rangle \quad : \tau \multimap \tau \multimap \mathbb{o} \multimap \mathbb{o}$$
$$\delta_b = \lambda l.\lambda r.\ (\lambda x.\ \langle x, x\rangle)\ (\pi_1(l) \circ b \circ \pi_1(r)) \quad\quad : \tau \multimap \tau \multimap \mathbb{o} \multimap \mathbb{o}$$

where $f \circ g$ stands for the composition $\lambda z.\ f\ (g\ z)$. The conditional swap is then coded as

$$\lambda t.\lambda^{!}a.\lambda^{!}b.\lambda^{!}c.\lambda^{!}\varepsilon.\ \pi_2\ (t\ \delta_a\ \delta_b\ (\lambda x.\ c\ x))\ \varepsilon$$

2.4. **Monoidal categories and related concepts.** Our use of category theory, while absolutely essential, stays at a fairly elementary level. We assume familiarity with the notions of category, functor, natural transformation, (cartesian) product and coproduct (and their nullary cases, terminal and initial objects), but not much more than that; the remaining categorical prerequisites are summed up here for convenience. The reader familiar with monoidal closed categories can safely skip directly to §2.4.3.

2.4.1. *Monoidal categories, symmetry and functors.* The idea of categorical semantics is to interpret the types of a programming language – in our case, the purely linear fragment of the $\lambda\ell^{\oplus\&}$-calculus – as objects, and the programs (terms) as morphisms. (A formal statement tailored to our purposes will be given later in Lemma 3.26.) In this perspective, the additive conjunction '&' of the $\lambda\ell^{\oplus\&}$-calculus is interpreted as a categorical *cartesian product*, while the additive disjunction '⊕' corresponds to a *coproduct*; this justifies our use of the notations &/⊕ for products/coproducts. We now define monoidal products, which are meant to interpret the multiplicative conjunction '⊗'.

Definition 2.17 ([Mel09, Sections 4.1 to 4.4])**.** Let $\mathcal{C}$ be a category. A *monoidal product* $\otimes$ over $\mathcal{C}$ is given by the combination of

- a bifunctor $- \otimes - : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$
- a distinguished object $\mathbf{I}$
- natural isomorphisms $\lambda_A : \mathbf{I} \otimes A \to A$ (*left unitor*), $\rho_A : A \otimes \mathbf{I} \to A$ (*right unitor*), and $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ (*associator*) subject to the following coherence conditions:

$$
\begin{array}{c}
(A \otimes B) \otimes (C \otimes D) \\
\end{array}
$$

with $\alpha_{A \otimes B, C, D}$, $\alpha_{A, B, C \otimes D}$, $((A \otimes B) \otimes C) \otimes D$, $A \otimes (B \otimes (C \otimes D))$, $\alpha_{A,B,C} \otimes \mathrm{id}_D$, $\mathrm{id}_A \otimes \alpha_{B,C,D}$, $(A \otimes (B \otimes C)) \otimes D \xrightarrow{\alpha_{A, B \otimes C, D}} A \otimes ((B \otimes C) \otimes D)$

$$
(A \otimes \mathbf{I}) \otimes B \xrightarrow{\alpha_{A, \mathbf{I}, B}} A \otimes (\mathbf{I} \otimes B)
$$

$\rho_A \otimes \mathrm{id}_B$, $\mathrm{id}_A \otimes \lambda_B$, $A \otimes B$

Such a monoidal product is called *symmetric* if it comes with natural isomorphisms $\gamma_{A,B} : A \otimes B \to B \otimes A$ subject to the following coherences

$$
(A \otimes B) \otimes C \xrightarrow{\alpha_{A,B,C}} A \otimes (B \otimes C) \xrightarrow{\gamma_{A, B \otimes C}} (B \otimes C) \otimes A \xrightarrow{\alpha_{B,C,A}} B \otimes (C \otimes A)
$$

$\gamma_{A,B} \otimes \mathrm{id}_C$, $(B \otimes A) \otimes C \xrightarrow{\alpha_{B,A,C}} B \otimes (A \otimes C)$, $\mathrm{id}_B \otimes \gamma_{A,C}$

$$
A \otimes B \xrightarrow{\gamma_{A,B}} B \otimes A
$$

$\gamma_{B,A}$, $A \otimes B$

In the sequel, we use the name *(symmetric) monoidal category* for a category $\mathcal{C}$ that comes equipped with a (symmetric) monoidal structure $\otimes, \mathbf{I}, \ldots$. We write such structures $(\mathcal{C}, \otimes, \mathbf{I})$ for short[16]. Of course, if a category $\mathcal{C}$ has products & and a terminal object $\top$, then $(\mathcal{C}, \&, \top)$ is a symmetric monoidal category, and similarly for coproducts and intial objects.

We shall sometimes need to refer to morphisms between monoidal categories, which are essentially functors together with natural transformations witnessing that the monoidal structure is preserved.

---

[16]Which is slightly abusive, as λ, ρ, α and γ are also part of the structure (and not uniquely determined from the triple $(\mathcal{C}, \otimes, \mathbf{I})$).

Definition 2.18 ([Mel09, Section 5.1]). Let $(\mathcal{C}, \otimes, \mathbf{I})$ and $(\mathcal{D}, \widehat{\otimes}, \widehat{\mathbf{I}})$ be two monoidal categories. A *lax monoidal functor* is given by a functor $F : \mathcal{C} \to \mathcal{D}$ together with natural transformations

$$m_0 : \widehat{\mathbf{I}} \to F(\mathbf{I}) \qquad\qquad m_{A,B} : F(A) \,\widehat{\otimes}\, F(B) \to F(A \otimes B)$$

making the following diagrams commute.

$$
\begin{array}{ccc}
(F(A) \,\widehat{\otimes}\, F(B)) \,\widehat{\otimes}\, F(C) & \xrightarrow{\ \alpha_{F(A),F(B),F(C)}\ } & F(A) \,\widehat{\otimes}\, (F(B) \,\widehat{\otimes}\, F(C)) \\
{\scriptstyle m_{A,B}\widehat{\otimes}\mathrm{id}_{F(C)}}\downarrow & & \downarrow{\scriptstyle \mathrm{id}_{F(A)}\widehat{\otimes}m_{B,C}} \\
F(A \otimes B) \,\widehat{\otimes}\, F(C) & & F(A) \,\widehat{\otimes}\, F(B \otimes C) \\
{\scriptstyle m_{A\otimes B,C}}\downarrow & & \downarrow{\scriptstyle m_{A,B\otimes C}} \\
F((A \otimes B) \otimes C) & \xrightarrow{\ F(\alpha_{A,B,C})\ } & F(A \otimes (B \otimes C))
\end{array}
$$

$$
\begin{array}{ccc}
F(A) \,\widehat{\otimes}\, \widehat{\mathbf{I}} & \xrightarrow{\ \rho_{F(A)}\ } & F(A) \\
{\scriptstyle \mathrm{id}_{F(A)}\widehat{\otimes}m_0}\downarrow & & \uparrow{\scriptstyle F(\rho_A)} \\
F(A) \,\widehat{\otimes}\, F(\mathbf{I}) & \xrightarrow{\ m_{A,\mathbf{I}}\ } & F(A \otimes \mathbf{I})
\end{array}
\qquad
\begin{array}{ccc}
\widehat{\mathbf{I}} \,\widehat{\otimes}\, F(A) & \xrightarrow{\ \lambda_{F(A)}\ } & F(A) \\
{\scriptstyle m_0\widehat{\otimes}\mathrm{id}_{F(A)}}\downarrow & & \uparrow{\scriptstyle F(\lambda_A)} \\
F(\mathbf{I}) \,\widehat{\otimes}\, F(A) & \xrightarrow{\ m_{\mathbf{I},A}\ } & F(\mathbf{I} \otimes A)
\end{array}
$$

A lax monoidal functor is called *strong monoidal* if the natural transformations m_0 and $m_{A,B}$ are isomorphisms.

Let us note that while every concrete instance of monoidal functor in the paper, save for the ultimate example in Appendix F, is also going to be a *symmetric* monoidal functor (i.e., satisfy additional coherence diagrams involving γ), we do not make use of that fact.

2.4.2. *Function spaces.* Our next definition concerns the categorical semantics of the linear function arrow '$\multimap$'. (Since we will only need a semantics for the *purely linear* fragment of the $\lambda\ell^{\oplus\&}$-calculus, we will not discuss the non-linear arrow '$\to$' here.)

Definition 2.19 ([Mel09, Sections 4.5 to 4.7]). Let $(\mathcal{C}, \otimes, \mathbf{I})$ be a (symmetric) monoidal category and $A, B \in \mathsf{Obj}(\mathcal{C})$. An *internal homset* from A to B is an object $A \multimap B \in \mathsf{Obj}(\mathcal{C})$ with a prescribed arrow $\mathrm{ev}_{A,B} : (A \multimap B) \otimes A \to B$ (the *evaluation map*) such that, for every other arrow $f : C \otimes A \to B$, there is a unique map $\Lambda(f)$ (called the *curryfication of* f) making the following diagram commute:

$$
\begin{array}{ccc}
(A \multimap B) \otimes A & \xrightarrow{\ \mathrm{ev}_{A,B}\ } & B \\
{\scriptstyle \Lambda(f)\otimes\mathrm{id}}\uparrow & \nearrow{\scriptstyle f} & \\
C \otimes A & &
\end{array}
$$

When there exists an internal homset for every pair objects in $\mathcal{C}$, we say that $(\mathcal{C}, \otimes, \mathbf{I})$ is a *(symmetric) monoidal closed* category.

As for (co)products, internal homsets are determined up to unique isomorphism, so we may talk somewhat loosely about *the* internal homset later on. While we work with the universal property given in Definition 2.19 when the construction of an internal homset involves a bit of combinatorics, we will also sometimes use the following characterization.

Proposition 2.20. *The object $A \multimap B$ is an internal homset for $A, B \in \mathsf{Obj}(\mathcal{C})$ if and only if there is a family of isomorphisms*

$$\mathsf{Hom}_\mathcal{C}\,(C \otimes A,\,B) \quad \cong \quad \mathsf{Hom}_\mathcal{C}\,(C,\,A \multimap B)$$

which is natural *in the parameter C varying contravariantly over $\mathcal{C}$ (in other words, if $\mathsf{Hom}_\mathcal{C}\,(- \otimes A,\,B)$ and $\mathsf{Hom}_\mathcal{C}\,(-,\,A \multimap B)$ are naturally isomorphic as functors $\mathcal{C}^{\mathrm{op}} \to \mathsf{Set}$).*

Proof. This is an instance of [ML98, Chapter III, Section 2, Proposition 1]. □

2.4.3. *Affineness and quasi-affineness.* Given a monoidal product $\otimes$, morphisms from A to $A \otimes A$ need not exist in general; this accounts for the linearity constraints in $\lambda\ell^{\oplus\&}$. But monoidal categories do not incorporate the ability of register transitions in SSTs to discard the content of a register, a behavior more aligned with the *affine* λ-calculi. This notion thus plays a role in our development, so we discuss its incarnation in categorical semantics.

Definition 2.21. A (symmetric) monoidal category $(\mathcal{C}, \otimes, \mathbf{I})$ is called *affine*[17] if $\mathbf{I}$ is a terminal object of $\mathcal{C}$.

Most symmetric monoidal categories are not affine. However, there is a generic way of building an affine monoidal category from a monoidal category. Recall that if $\mathcal{C}$ is a category and X is an object of $\mathcal{C}$, one may consider the *slice category* $\mathcal{C}/X$

- whose objects are morphisms $A \to X$ ($A \in \mathsf{Obj}(\mathcal{C})$),
- and such that $\mathsf{Hom}_{\mathcal{C}/X}\,(f : A \to X,\, g : B \to X) = \{h \in \mathsf{Hom}_\mathcal{C}\,(A, B) \mid g \circ h = f\}$.

If $\mathcal{C}$ has a monoidal structure $(\otimes, \mathbf{I})$, this structure can be lifted to $\mathcal{C}/\mathbf{I}$ by taking the identity $\mathbf{I} \to \mathbf{I}$ as the unit and

$$\left(A \xrightarrow{\;f\;} \mathbf{I} \right) \otimes \left(B \xrightarrow{\;g\;} \mathbf{I} \right) \quad = \quad \left(A \otimes B \xrightarrow{\;f \otimes g\;} \mathbf{I} \otimes \mathbf{I} \xrightarrow{\;\lambda_\mathbf{I} = \rho_\mathbf{I}\;} \mathbf{I} \right)$$

as the monoidal product. This gives rise to an affine monoidal structure over $\mathcal{C}/\mathbf{I}$, and a strong monoidal structure for the forgetful functor $\mathsf{dom} : \mathcal{C}/\mathbf{I} \to \mathcal{C}$.

In the converse direction, one can sometimes turn an object A from $\mathcal{C}$ into one of $\mathcal{C}/\mathbf{I}$. This is the case when A admits a cartesian product with $\mathbf{I}$, which may be written $A \mathbin{\&} \mathbf{I}$ (note that if $\mathcal{C}$ is affine, A itself is such a cartesian product). We are then led to consider the projection $\pi_2 : A \mathbin{\&} \mathbf{I} \to \mathbf{I}$ as an object of the slice category.

Definition 2.22. A (symmetric) monoidal category $(\mathcal{C}, \otimes, \mathbf{I})$ is called *quasi-affine* if every $A \in \mathsf{Obj}(\mathcal{C})$ has a cartesian product $A \mathbin{\&} \mathbf{I}$ with the monoidal unit.

Remark 2.23. We have a map $A \in \mathsf{Obj}(\mathcal{C}) \mapsto \left(A \mathbin{\&} \mathbf{I} \xrightarrow{\;\pi_2\;} \mathbf{I} \right) \in \mathsf{Obj}\left(\mathcal{C}/\mathbf{I} \right)$ in any quasi-affine category, according to the above discussion. It turns out that it extends to a functor J which embeds $\mathcal{C}$ into this affine slice category; moreover, J is *right adjoint* to the forgetful functor dom. The interested reader may even check (although we will not make use of this) that the existence of a right adjoint to dom is *equivalent* to quasi-affineness.

---

[17]Such categories are also sometimes called *semi-cartesian* [nLa20]. We rather chose affine here for conciseness and because we will have to handle categories which have both cartesian products & and an additional affine monoidal product $\otimes$.

2.4.4. *Monoids.* Since we are interested in string transductions, the free monoids Σ^* are going to make an appearance. Let us thus conclude this section by recalling the notion of monoid *internal to a monoidal category*.

Definition 2.24 ([Mel09, Section 6.1])**.** Given a monoidal category $(\mathcal{C}, \otimes, \mathbf{I})$, an *internal monoid* (or a *monoid object*) is a triple (M, μ, η) where $M \in \mathsf{Obj}(\mathcal{C})$ and $\mu : M \otimes M \to M$, $\eta : \mathbf{I} \to M$ are morphisms making the following unitality and associativity diagrams commute

$$\mathbf{I} \otimes M \xrightarrow{\lambda_\mathbf{I}} M \xleftarrow{\rho_\mathbf{I}} M \otimes \mathbf{I} \quad (M \otimes M) \otimes M \xrightarrow{\alpha_{M,M,M}} M \otimes (M \otimes M) \xrightarrow{\mathrm{id} \otimes \mu} M \otimes M$$

with $\eta \otimes \mathrm{id}$, μ, μ, $\mathrm{id} \otimes \eta$ down to $M \otimes M$, and $\mu \otimes \mathrm{id}$ down to $M \otimes M \xrightarrow{\mu} M$, and μ on the right.

A useful example of this notion is the "internalization" of the monoid of endomorphisms of A when A is part of a monoidal *closed* category.

Proposition 2.25. *Let* $(\mathcal{C}, \otimes, \mathbf{I})$ *be a monoidal category. Any internal homset* $A \multimap A$ *(with* $A \in \mathsf{Obj}(C)$*) that exists in* $\mathcal{C}$ *has an internal monoid structure* $(A \multimap A,\ \eta,\ \mu)$ *such that*

$$\eta = \Lambda'(\mathrm{id}_A) \qquad \mu \circ (\Lambda'(f) \otimes \Lambda'(g)) \circ \lambda_\mathbf{I} = \Lambda'(f \circ g) \quad for\ f, g \in \mathsf{Hom}_\mathcal{C}(A, A)$$

where $\Lambda' : \mathsf{Hom}_\mathcal{C}(A, A) \xrightarrow{\sim} \mathsf{Hom}_\mathcal{C}(\mathbf{I}, A \multimap A)$ *is defined as* $\Lambda' : h \mapsto \Lambda(h \circ \lambda_A)$ *from the curryfication* Λ *and the left unitor* λ_A.

Proof sketch. One can define the monoid multiplication $\mu : (A \multimap A) \otimes (A \multimap A) \to (A \multimap A)$ as the curryfication $\mu = \Lambda(\mathsf{app2})$ of the morphism $\mathsf{app2}$ built by composing the sequence

$$((A \multimap A) \otimes (A \multimap A)) \otimes A \xrightarrow{\alpha} (A \multimap A) \otimes ((A \multimap A) \otimes A) \xrightarrow{\mathrm{id} \otimes \mathrm{ev}} (A \multimap A) \otimes A \xrightarrow{\mathrm{ev}} A$$

and check that it satisfies the coherence diagrams for internal monoids (that also involve the unit η defined in the proposition statement) and the equation relating μ to Λ'. $\square$

Let us conclude our categorical preliminaries on the following.

Proposition 2.26. *Let* $(\mathcal{C}, \otimes, \mathbf{I})$ *be a monoidal category and let* $M \mathbin{\&} \mathbf{I} \in \mathsf{Obj}(C)$ *be a cartesian product of some* $M \in \mathsf{Obj}(C)$ *with the monoidal unit* $\mathbf{I}$. *Suppose that* (M, μ, η) *is a monoid object. Then* $M \mathbin{\&} \mathbf{I}$ *has an internal monoid structure defined by*

$$\langle \mu \circ (\pi_1 \otimes \pi_1),\ \lambda_\mathbf{I} \circ (\pi_2 \otimes \pi_2) \rangle\ :\ (M \mathbin{\&} \mathbf{I}) \otimes (M \mathbin{\&} \mathbf{I}) \to M \mathbin{\&} \mathbf{I} \qquad \langle \eta, \mathrm{id}_\mathbf{I} \rangle\ :\ \mathbf{I} \to M \mathbin{\&} \mathbf{I}$$

where $\pi_1 : M \mathbin{\&} \mathbf{I} \to M$ *and* $\pi_2 : M \mathbin{\&} \mathbf{I} \to \mathbf{I}$ *are the projections and* $\langle -, - \rangle$ *is the pairing given by the universal property of the cartesian product.*

Furthermore, this makes $\left(M \mathbin{\&} \mathbf{I} \xrightarrow{\pi_2} \mathbf{I} \right) \in \mathsf{Obj}\left(\mathcal{C}/_\mathbf{I} \right)$ *into a monoid object of* $\mathcal{C}/_\mathbf{I}$.

The routine verification of the required commutations of diagrams is left to the reader.

Remark 2.27. Our applications of this proposition will take place in *quasi-affine* monoidal categories. For those, it admits a more conceptual proof: the right adjoint $J : \mathcal{C}/_\mathbf{I} \to \mathcal{C}$ to the forgetful functor dom (cf. Remark 2.23) is *lax monoidal*, and therefore so is $\mathsf{dom} \circ J$ which maps A to $A \mathbin{\&} \mathbf{I}$ on objects; furthemore, the image of a monoid object by a lax monoidal functor is itself a monoid object in a canonical way [Mel09, Section 6.2].

3. Regular string functions in the $\lambda\ell^{\oplus\&}$-calculus

The goal of this section is to prove our main theorem pertaining to string functions.

Theorem 1.1. *A function* $\Sigma^* \to \Gamma^*$ *is* $\lambda\ell^{\oplus\&}$-definable *if and only if it is* regular.

To prove Theorem 1.1, we introduce a generalized notion of SST parameterized by a structure that we call a *(string) streaming setting* $\mathfrak{C}$, which is a structure whose main component is a category to be thought of as the collection of possible register transitions. From the point of view of expressiveness, $\mathfrak{C}$ can be thought of as a gadget delimiting a class of transition monoids which may be used for computations on top of finite structure of a $\mathfrak{C}$-SST. The reason why we use categories as parameters is to be able to bridge easily the usual notion of SST and the categorical semantics of $\lambda\ell^{\oplus\&}$ in a single framework.

In Section 3.1, we define our streaming settings and $\mathfrak{C}$-SSTs. We make the connections with usual SSTs and $\lambda\ell^{\oplus\&}$, in §3.2 and §3.3 respectively, through two distinguished streaming settings $\mathfrak{SR}$ and $\mathfrak{L}$. This allows to reframe Theorem 1.1 as the equivalence between $\mathfrak{SR}$-SSTs and single-state $\mathfrak{L}$-SSTs. Then, in Section 3.4, we study the free coproduct completion of categories $(-)_{\oplus}$, which readily extends to streaming settings. In particular, properties of $\mathfrak{SR}_{\oplus}$ are explored. Section 3.5 deals with the dual construction $(-)_{\&}$, the free product completion. A tight link between the expressiveness of $\mathfrak{C}_{\&}$-SSTs and *non-deterministic* $\mathfrak{C}$-SSTs is established. Section 3.6 then combines those results to study the composition $((-)_{\&})_{\oplus}$ of those two completions (which we describe as a direct construction $(-)_{\oplus\&}$), relying on the previous sections. In particular, it is shown that the category at the center of $\mathfrak{SR}_{\oplus\&}$ is a model of the purely linear fragment of $\lambda\ell^{\oplus\&}$. Finally, Section 3.7 briefly summarizes how to combine the results of the previous sections into a proof of Theorem 1.1.

3.1. **A categorical framework for automata: streaming settings.** We now introduce string streaming settings, which should be seen as a sort of memory framework for transducers iterating performing a single left-to-right pass over a word. This is the abstract notion that will allow us to generalize SSTs:

Definition 3.1. Let X be a set. A *string streaming setting* with output X is a tuple $\mathfrak{C} = (\mathcal{C}, \mathbb{T}, \bot\!\!\bot, (\!|-|\!))$ where

- $\mathcal{C}$ is a category
- $\mathbb{T}$ and $\bot\!\!\bot$ are arbitrary objects of $\mathcal{C}$
- $(\!|-|\!)$ is a set-theoretic map $\mathsf{Hom}_{\mathcal{C}}(\mathbb{T}, \bot\!\!\bot) \to X$

Since the properties of the underlying category of a streaming setting will turn out to be the most crucial thing in the sequel, we shall abusively apply adjectives befitting categories to streaming settings, such as "affine symmetric monoidal" to streaming settings in the sequel.

The notion of streaming setting is a convenient tool motivated by our subsequent development rather than our primary object of study. A closely related framework in which some of our abstract results can be formulated is defined in [CP20] (see Remark 3.3).

For the rest of this section, we will refer to string streaming setting simply as streaming settings; we also fix two alphabets Σ and Γ for the rest of this section.

Definition 3.2. Let $\mathfrak{C} = (\mathcal{C}, \mathbb{T}, \bot\!\!\bot, (\!|-|\!))$ be a streaming setting with output X. A $\mathfrak{C}$-SST with input alphabet Σ and output X is a tuple $(Q, q_0, R, \delta, i, o)$ where

- Q is a finite set of states and $q_0 \in Q$
- R is an object of $\mathcal{C}$

- δ is a function $\Sigma \times Q \to Q \times \mathsf{Hom}_{\mathcal{C}}(R, R)$
- $i \in \mathsf{Hom}_{\mathcal{C}}(\top, R)$ is an initialization morphism
- $(o_q)_{q \in Q} \in \mathsf{Hom}_{\mathcal{C}}(R, \bot\!\!\!\bot)^Q$ is a family of output morphisms – alternatively, we will sometimes consider it as a map $o : Q \to \mathsf{Hom}_{\mathcal{C}}(R, \bot\!\!\!\bot)$.

We write $\mathcal{T} : \Sigma^* \to_{\mathfrak{C}\text{-SST}} X$ to mean that $\mathcal{T}$ is a $\mathfrak{C}$-SST with input alphabet Σ and output X (the latter depends only on $\mathfrak{C}$).

The corresponding function $[\![\mathcal{T}]\!] : \Sigma^* \to X$ is then computed as for standard SSTs (cf. Definition 2.3): an input word w generates a sequence of states $q_0, \ldots, q_{|w|} \in Q$ and a sequence of morphisms $f_i : R \to R$ in $\mathcal{C}$, and the output is then $(\!| o_{q_{|w|}} \circ f_{|w|} \circ \cdots \circ f_1 \circ i |\!) \in X$.

An important class of $\mathfrak{C}$-SSTs are those for which the set of states Q is a singleton, significantly simplifying the above data. They are called *single-state $\mathfrak{C}$-SSTs*.

Remark 3.3. Single-state $\mathfrak{C}$-SSTs are very close to the $\mathcal{C}$-automata over words defined by Colcombet and Petrişan [CP20, Section 3], or more precisely $(\mathcal{C}, \top, \bot\!\!\!\bot)$-automata with our notations. The main difference is that the latter's output would juste be an element of $\mathsf{Hom}_{\mathcal{C}}(\top, \bot\!\!\!\bot)$: there is no post-processing $(\!|-|\!)$ to produce an output.

As for the addition of finite states, ultimately, it does not increase the framework's expressive power: we shall see in Remark 3.37 that $\mathfrak{C}$-SSTs are equivalent to single-state SSTs over a modified category. We chose to incorporate states into our definition for convenience.

Example 3.4. Let $\mathfrak{Set}_X = (\mathsf{Set}, \{\bullet\}, X, (\!|-|\!))$ where $(\!|-|\!)$ is the canonical isomorphism between $\mathsf{Hom}_{\mathsf{Set}}(\{\bullet\}, X) = X^{\{\bullet\}}$ and X. Then *any* function $\Sigma^* \to X$ can be "computed" by a single-state $\mathfrak{Set}_X$-SST by taking $R = \Sigma^*$.

Example 3.5. Let $\mathfrak{Finset}_2 = (\mathsf{FinSet}, \{\bullet\}, \{0, 1\}, (\!|-|\!))$ with $(\!|-|\!)$ the canonical isomorphism $\mathsf{Hom}_{\mathsf{FinSet}}(\{\bullet\}, \{0, 1\}) \cong \{0, 1\}$. Single-state $\mathfrak{Finset}_2$-SST are essentially the usual notion of deterministic finite automata[18]. Therefore, the functions they compute are none other than the indicator functions of regular languages.

Example 3.6. Consider the category $\mathcal{POL}_{\mathbb{Q}}$ whose objects are natural numbers, whose morphisms are tuples of multivariate polynomials over $\mathbb{Q}$ with matching arities (so that $\mathsf{Hom}_{\mathcal{POL}_{\mathbb{Q}}}(n, k) = (\mathbb{Q}[X_1, \ldots, X_n])^k$) and where composition is lifted from the composition of polynomials in the usual way, making $\mathcal{POL}_{\mathbb{Q}}$ into a category with (strict) cartesian products. Then, taking $\mathfrak{Pol}_{\mathbb{Q}} = (\mathcal{POL}_{\mathbb{Q}}, 0, 1, (\!|-|\!))$ where $(\!|-|\!)$ is the isomorphism identifying $\mathbb{Q}$ and polynomials without variables ($n = 0$), we can recover the definition of *polynomial automata* from [BDSW17] as single-state $\mathfrak{Pol}_{\mathbb{Q}}$-SSTs.

Example 3.7. The core of a paper by Hines [Hin03] can be recast in our framework as saying that *two-way automata* are the same thing as single-state SSTs over a well-chosen streaming setting, whose underlying category is called $\mathsf{Int}(\mathsf{pSet})$ in [Hin03] (we will not give further details here). Recall that the transducer counterparts of these devices, namely *two-way transducers*, are among the characterizations of regular string functions [EH01].

Interestingly, this provides yet another connection with linear logic: this category $\mathsf{Int}(\mathsf{pSet})$ belongs to a family of techniques called "geometry of interaction" related to the game semantics of linear logic. As an example of recent application of such techniques to the interface between λ-calculus and automata, we refer to the work of Clairambault and Murawski [CM19] on higher-order recursion schemes. $\mathsf{Int}(\mathsf{pSet})$ is also close in spirit to the construction of free symmetric compact closed categories [KL80].

---

[18]Actually, *complete* DFA, i.e. DFA with total transition functions.

Given two streaming settings $\mathfrak{C}$ and $\mathfrak{D}$ with a common output set X, $\mathfrak{C}$-SSTs are said to *subsume* $\mathfrak{D}$-SSTs if for every $\mathfrak{D}$-SST $\mathcal{T}$ there is a $\mathfrak{C}$-SST $\mathcal{T}'$ with $[\![\mathcal{T}]\!] = [\![\mathcal{T}']\!]$. We say that $\mathfrak{C}$-SSTs and $\mathfrak{D}$-SSTs are *equivalent* if both classes subsume one another.

There is a straightforward notion of morphism of streaming settings with common output.

Definition 3.8. Let $\mathfrak{C} = (\mathcal{C}, \mathbb{T}_{\mathfrak{C}}, \bot\!\!\!\bot_{\mathfrak{C}}, (\![-]\!)_{\mathfrak{C}})$ and $\mathfrak{D} = (\mathcal{D}, \mathbb{T}_{\mathfrak{D}}, \bot\!\!\!\bot_{\mathfrak{D}}, (\![-]\!)_{\mathfrak{D}})$ be streaming settings with the same output set X. A morphism of streaming settings is given by a functor $F : \mathcal{C} \to \mathcal{D}$ and $\mathcal{D}$-arrows $i : \mathbb{T}_{\mathfrak{D}} \to F(\mathbb{T}_{\mathfrak{C}})$ and $o : F(\bot\!\!\!\bot_{\mathfrak{C}}) \to \bot\!\!\!\bot_{\mathfrak{D}}$ such that

$$\forall f \in \mathsf{Hom}_{\mathcal{C}}\left(\mathbb{T}_{\mathfrak{C}}, \bot\!\!\!\bot_{\mathfrak{C}}\right), \ (\![o \circ F(f) \circ i]\!)_{\mathfrak{D}} = (\![f]\!)_{\mathfrak{C}}$$

This notion is useful to compare the expressiveness of classes of generalized SSTs because of the following lemma.

Lemma 3.9. *If there is a morphism of streaming settings $\mathfrak{C} \to \mathfrak{D}$, then $\mathfrak{D}$-SSTs subsume $\mathfrak{C}$-SSTs and single-state $\mathfrak{D}$-SSTs subsume single-state $\mathfrak{C}$-SSTs.*

Proof sketch. Given a $\mathfrak{C}$-SST $(Q, q_0, R, \delta, i, (o_q)_{q\in Q})$ (with the notations of Definition 3.2) and a morphism of streaming settings $(F : \mathcal{C} \to \mathcal{D}, \ i' : \mathbb{T}_{\mathfrak{D}} \to F(\mathbb{T}_{\mathfrak{C}}), \ o' : F(\bot\!\!\!\bot_{\mathfrak{C}}) \to \bot\!\!\!\bot_{\mathfrak{D}})$, one builds a $\mathfrak{D}$-SST that computes the same function as follows. The set of states and initial state are unchanged (so our proof applies both to the stateful and the single-state case). The memory object becomes $F(R)$, and the $\mathsf{Hom}_{\mathcal{C}}\left((,R),R\right)$ component of the transition δ function is passed through the functor F to yield a $\mathfrak{D}$-morphism $F(R) \to F(R)$. The new initialization morphism is $F(i) \circ i'$ and the new output morphisms are $(o' \circ F(o_q))_{q\in Q}$. $\square$

Remark 3.10. For any streaming setting $\mathfrak{C}$, the functor $\mathsf{Hom}_{\mathcal{C}}\left(\mathbb{T}, -\right)$ is a morphism of streaming settings $\mathfrak{C} \to \mathfrak{Set}$ with $i = \mathrm{id}$ and $o = (\![-]\!)_{\mathfrak{C}}$.

In the sequel, we will omit giving the morphisms $i : \top_{\mathcal{D}} \to F(\top_{\mathcal{C}})$ and $o : F(\bot\!\!\!\bot_{\mathcal{C}}) \to \bot\!\!\!\bot_{\mathcal{D}}$ most of the time, as they will be isomorphisms deducible from the context. The one exception to this situation will be in Lemma F.2.

3.2. **The category $\mathcal{SR}(\Gamma)$ of Γ-register transitions.** We now show that usual copyless SSTs are indeed an instance of our general notion of categorical SSTs. To do so we must arrange copyless register transitions (Definition 2.2) into a category: given $t \in [R \to_{\mathcal{SR}(\Gamma)} S]$ and $t' \in [S \to_{\mathcal{SR}(\Gamma)} T]$, we must be able to compose them into $t' \circ t \in [R \to_{\mathcal{SR}(\Gamma)} T]$. Moreover, this composition should be compatible with the action of register transitions on tuples of strings, i.e. the latter should be *functorial*: $(t' \circ t)^\dagger = t'^\dagger \circ t^\dagger$.

Definition 3.11 (see e.g. [AFT12, Section C]). Let $t \in [R \to_{\mathcal{SR}(\Gamma)} S]$ and $t' \in [S \to_{\mathcal{SR}(\Gamma)} T]$; recall that t and t' are defined as maps between sets $t : S \to (\Gamma + R)^*$ and $t' : T \to (\Gamma + S)^*$.

We define the *composition of register transitions* $t' \circ_{\mathcal{SR}(\Gamma)} t : T \to (\Gamma + R)^*$ to be the set-theoretic composition $t^\ddagger \circ t'$ where $t^\ddagger : (\Gamma + S)^* \to (\Gamma + R)^*$ is the unique monoid morphism extending the copairing of in_1 and t (i.e. $(\mathsf{in}_1(c) \mapsto \mathsf{in}_1(c), \mathsf{in}_2(s) \mapsto t(s)) : \Gamma + S \to (\Gamma + R)^*$).

Proposition 3.12. *There is a category $\mathcal{SR}(\Gamma)$ (given a finite alphabet Γ which we will often omit in the notation) whose objects are finite sets of registers, whose morphisms are copyless register transitions – $\mathsf{Hom}_{\mathcal{SR}(\Gamma)}(R, S) = [R \to_{\mathcal{SR}(\Gamma)} S]$ – and whose composition is given by the above definition. This means in particular that, with the above notations, $t' \circ t \in [R \to_{\mathcal{SR}(\Gamma)} T]$, i.e. copylessness is preserved by composition. Furthermore:*
- *This category admits the empty set of registers as the terminal object: $\top = \varnothing$.*

- *The action of register transitions on tuples of strings gives rise to a functor $(-)^\dagger : \mathcal{SR} \to \mathsf{Set}$, with $X^\dagger = (\Gamma^*)^X$ on objects.*

The above proposition, which the interested reader may verify from the definitions, is merely a restatement using categorical vocabulary of properties that are already used in the literature on usual SSTs.

Definition 3.13. We write $\mathfrak{SR}(\Gamma)$ for the streaming setting $(\mathcal{SR}(\Gamma), \top = \varnothing, \bot = \{\bullet\}, (\!\!|-|\!\!))$ where $(\!\!|-|\!\!) : [\varnothing \to_{\mathcal{SR}(\Gamma)} \{\bullet\}] \to \Gamma^*$ is the canonical isomorphism $((\Gamma + \varnothing)^*)^{\{\bullet\}} \cong \Gamma^*$.

Fact 3.14. Standard copyless SSTs $\Sigma^* \to_{\mathrm{SST}} \Gamma^*$ are the same thing as $\mathfrak{SR}$-SSTs $\Sigma^* \to \Gamma^*$.

Remark 3.15. The functor $\mathsf{Hom}_{\mathcal{R}}(\top, -)$ mentioned in Remark 3.10 is, in the case of $\mathcal{R}$, naturally isomorphic to $(-)^\dagger$. Therefore, the latter can be extended to a morphism $\mathfrak{SR} \to \mathfrak{Set}$ of string streaming settings.

Proposition 3.16. *The category $\mathcal{SR}$ can be endowed with a symmetric monoidal structure, where the monoidal product $R \otimes S$ is the disjoint union of register sets $R + S$ and the unit is the empty set of registers. Since the latter is also the terminal object of $\mathcal{SR}$, this defines an affine symmetric monoidal category.*

Note that given $t \in [R \to_{\mathcal{SR}(\Gamma)} S]$ and $t' \in [T \to_{\mathcal{SR}(\Gamma)} U]$, there is only one sensible way to define a set-theoretic map $t \otimes t' : U + S \to (\Gamma + (R + T))^*$. The above proposition states, among other things, that $t \otimes t' \in [R + T \to_{\mathcal{SR}(\Gamma)} S + U]$. Checking this, as well as the requisite coherence diagrams for monoidal categories, is left to the reader.

Next, let us observe that $\{\bullet\} \in \mathsf{Obj}(\mathcal{SR})$, representing a single register, can be equipped with the structure of an internal monoid $(\{\bullet\}, \mu_\bullet, \eta_\bullet)$ by setting

$$\eta_\bullet(\bullet) = \varepsilon \qquad \text{and} \qquad \mu_\bullet(\bullet) = \mathsf{in}_2(\mathtt{l})\mathsf{in}_2(\mathtt{r}) \quad \text{where } \mathtt{l} = \mathsf{in}_1(\bullet) \text{ and } \mathtt{r} = \mathsf{in}_2(\bullet)$$

so that $\mu_\bullet \in [\{\bullet\} \to_{\mathcal{SR}(\Gamma)} \{\bullet\} \otimes \{\bullet\}]$ has the codomain $\Gamma + (\{\bullet\} + \{\bullet\}) = \Gamma + \{\mathtt{l}, \mathtt{r}\}$ when considered as a map between sets. This internal monoid is the key to giving an inductive characterization of $\mathcal{SR}$. Given a string $w = w_1 \dots w_n \in \Gamma^*$, let us write $\widehat{w} \in [\varnothing \to_{\mathcal{SR}(\Gamma)} \{\bullet\}]$ for the register transition defined by the map $\widehat{w} : \bullet \mapsto \mathsf{in}_1(w_1) \dots \mathsf{in}_1(w_n) \in (\Gamma + \varnothing)^*$.

Theorem 3.17. *Let $(\mathcal{C}, \otimes, \mathbf{I})$ be an* affine *symmetric monoidal category.*

For any internal monoid (M, μ, η) of $\mathcal{C}$ and any family $(m_c)_{c \in \Gamma} \in \mathsf{Hom}_{\mathcal{C}}(\mathbf{I}, M)$ of morphisms, there exists a strong monoidal functor $F : \mathcal{SR}(\Gamma) \to \mathcal{C}$ *such that:*

- $F(\varnothing) = \mathbf{I}$, $F(\{\bullet\}) = M$ *and* $F(\widehat{c}) = m_c$ *for every* $c \in \Gamma$;
- $F(\mu_\bullet) = \mu$ *and* $F(\eta_\bullet) = \eta$ *with the above definitions (implying that* $F(\{\mathtt{l}, \mathtt{r}\}) = M \otimes M$);
- *the isomorphisms* $\mathbf{I} \to F(\varnothing)$ *and* $F(\{\bullet\}) \otimes F(\{\bullet\}) \to F(\{\bullet\} + \{\bullet\})$ *that are part of the strong monoidal structure for F are equal to* $\mathrm{id}_\mathbf{I}$ *and* $\mathrm{id}_{M \otimes M}$ *respectively.*

Note that since F is a monoidal functor, it transports the monoid object $(\{\bullet\}, \mu_\bullet, \eta_\bullet)$ to a structure of internal monoid over $F(\{\bullet\}) = M$ in a canonical way [Mel09, Section 6.2]. A fact that encapsulates the idea of the second item above – but which, strictly speaking, also depends on the third one – is that the result of this transport is precisely (M, μ, η).

Proof. Although the intuition of the proof is simple, its execution involves a significant amount of bureaucracy; in particular, it manipulates canonical isomorphisms given by Mac Lane's coherence theorem for symmetric monoidal categories [ML98, Section XI.1]. For this reason, we will only illustrate the idea here in the concrete case of the cartesian category of sets; *the full proof can be found in Appendix E.*

For $(\mathcal{C}, \otimes, \mathbf{I}) = (\mathsf{Set}, \times, \{*\})$, we can reformulate the data given in the statement as a monoid M with a family of elements $(m_c)_{c \in \Gamma} \in M^\Gamma$, that can be identified with functions $m_c : \{*\} \to M$ for $c \in \Gamma$. The functor that we build then maps an object R – a finite set of registers – to the set M^R. The action on morphisms is best illustrated through an example: $(t : z \mapsto axby) \in [\{x, y\} \to_{\mathcal{SR}(\{a,b\})} \{z\}]$ (where $\mathsf{in}_1/\mathsf{in}_2$ are omitted) becomes the map $(u_x, u_y) \in M^{\{x,y\}} \mapsto m_a u_x m_b u_y \in M \cong M^{\{z\}}$. Note that when we apply this construction to $M = \Gamma^*$ with $m_c = c$, we recover the functor $(-)^\dagger : \mathcal{SR}(\Gamma) \to \mathsf{Set}$ from Proposition 3.12. $\square$

Remark 3.18. Informally speaking, we think of $\mathcal{SR}(\Gamma)$ as the affine symmetric monoidal category freely generated by an internalization of the free monoid Γ^*. To truly express this, one would need to add to Theorem 3.17 the uniqueness of F up to natural isomorphism. This would imply, among other things, that the morphisms of $\mathcal{SR}$ are inductively given by

- the identities id
- the compositions
- the structural morphisms associated to the tensor product
- the unique morphism $\{\bullet\} \to \varnothing$
- canonical morphisms $\widehat{c} : \top \to \{\bullet\}$ for every individual letter $c \in \Gamma$
- a canonical morphism $\eta : \top \to \{\bullet\}$ corresponding to the empty word
- a multiplication morphism $\mu : \{\bullet\} \otimes \{\bullet\} \to \{\bullet\}$ corresponding to string concatenation.

However, we do not prove this inductive presentation, nor the uniqueness property, since they are not necessary for our purposes.

Since the monoid object structure of internal homsets (Proposition 2.25) has a somewhat explicit description, Theorem 3.17 admits a specialized and simplified formulation for affine symmetric monoidal closed categories. For our purposes, it will be useful to give a version that also applies in the quasi-affine case.

Corollary 3.19. *Let* $(\mathcal{C}, \otimes, \mathbf{I})$ *be a* quasi-affine *symmetric monoidal* closed *category and A be an object in $\mathcal{C}$. For any family $(f_c)_{c \in \Gamma} \in \mathsf{Hom}_{\mathcal{C}}(A, A)^\Gamma$ of endomorphisms, there exists a strong monoidal functor $F : \mathcal{SR}(\Gamma) \to \mathcal{C}$ such that*

$$F(\varnothing) = \mathbf{I} \qquad F(\{\bullet\}) = (A \multimap A) \,\&\, \mathbf{I} \qquad \forall w \in \Gamma^*, \, F(\widehat{w}) = \left\langle \Lambda'\left(f_{w[1]} \circ \cdots \circ f_{w[n]}\right), \mathrm{id}_{\mathbf{I}} \right\rangle$$

where we use the notations $\widehat{(-)}$ from Theorem 3.17 and $\Lambda'(-)$ from Proposition 2.25.

Our first application of this result will be to show in the next subsection that all regular functions are $\lambda\ell^{\oplus\&}$-definable. For this purpose, the level of abstraction that we are working with is unnecessary: it would suffice to encode copyless SSTs as $\lambda\ell^{\oplus\&}$-terms, a programming exercise that is not particularly difficult. But later, the generalized preservation theorem of Section 4.1 will apply Corollary 3.19 in its full generality.

Proof. Proposition 2.25 gives a canonical internal monoid structure to $A \multimap A$, which by Proposition 2.26 can be lifted to a monoid object $((A \multimap A) \,\&\, \mathbf{I}, \mu, \eta)$. For $f \in \mathsf{Hom}_{\mathcal{C}}(A, A)$, let $\Lambda''(f) = \langle \Lambda'(f), \mathrm{id}_{\mathbf{I}} \rangle : \mathbf{I} \to (A \multimap A) \,\&\, \mathbf{I}$.

We apply Theorem 3.17 to the slice category $\mathcal{C}/\mathbf{I}$ – which satisfies the affineness assumption – with the internal monoid $\pi_2 : (A \multimap A) \,\&\, \mathbf{I} \to \mathbf{I}$ (cf. Proposition 2.26 again) and the family $(\Lambda''(f_c))_{c \in \Gamma}$ (each $\Lambda''(f)$ is a morphism in the slice category from its unit $\mathrm{id}_{\mathbf{I}}$ to this π_2 since $\pi_2 \circ \Lambda''(f) = \mathrm{id}_{\mathbf{I}}$). We compose the resulting functor $\mathcal{SR} \to \mathcal{C}/\mathbf{I}$ with the forgetful functor $\mathsf{dom} : \mathcal{C}/\mathbf{I} \to \mathcal{C}$ to get $F : \mathcal{SR} \to \mathcal{C}$ such that $F(\varnothing) = \mathbf{I}$ and $F(\{\bullet\}) = (A \multimap A) \,\&\, \mathbf{I}$.

As a composition of strong monoidal functors, F is also strong monoidal. This takes care of all but one of the corollary's conclusions.

For the remaining one, we need to do some preliminary work. First, let us recall two commuting diagrams below. The left one comes from the naturality of the family of (iso)morphisms $m_{R,S} : F(R) \otimes F(S) \to F(R + S)$ that make F a (strong) monoidal functor, while the right one is among the coherence conditions in Definition 2.18.

$$
\begin{array}{ccc}
\mathbf{I} \otimes \mathbf{I} & \xrightarrow{\;m_{\varnothing,\varnothing}\;} & F(\varnothing + \varnothing) \\
{\scriptstyle F(\widehat{u}) \otimes F(\widehat{v})}\Big\downarrow & & \Big\downarrow{\scriptstyle F(\widehat{u} \otimes \widehat{v})} \\
F(\{\bullet\}) \otimes F(\{\bullet\}) & \xrightarrow{\;m_{\{\bullet\},\{\bullet\}}\;} & F(\{\bullet\} + \{\bullet\})
\end{array}
\qquad
\begin{array}{ccc}
\mathbf{I} \otimes F(\varnothing) & \xrightarrow{\;\lambda_{F(\varnothing)}\;} & F(\varnothing) \\
{\scriptstyle m_0 \otimes \mathrm{id}_{F(\varnothing)}}\Big\downarrow & & \Big\uparrow{\scriptstyle F(\lambda_\varnothing)} \\
F(\varnothing) \otimes F(\varnothing) & \xrightarrow{\;m_{\varnothing,\varnothing}\;} & F(\varnothing + \varnothing)
\end{array}
$$

Here, $m_0 : \mathbf{I} \to F(\varnothing)$ is also part of the strong monoidal structure of F and $u, v \in \Gamma^*$. The construction of Theorem 3.17 gives us $m_0 = \mathrm{id}_{\mathbf{I}}$ and $m_{\{\bullet\},\{\bullet\}} = \mathrm{id}_{F(\{\bullet\}+\{\bullet\})}$; furthermore, we have $\varnothing + \varnothing = \varnothing$ and $\lambda_\varnothing = \mathrm{id}_\varnothing$ in $\mathcal{SR}$. Thus, in the end, we can combine the two equalities expressed by the above diagrams and simplify them to get

$$
\forall u, v \in \Gamma^*, \quad F(\widehat{u}) \otimes F(\widehat{v}) = F(\widehat{u} \otimes \widehat{v}) \circ \lambda_{\mathbf{I}}
$$

Theorem 3.17 also guarantees that $F(\mu_\bullet) = \mu$, $F(\eta_\bullet) = \eta$ and $F(\widehat{c}) = \Lambda''(f_c)$ for all $c \in \Gamma$. At the same time, by combining Propositions 2.25 and 2.26, one can derive

$$
\eta = \Lambda''(\mathrm{id}_A) \qquad \mu \circ (\Lambda''(f) \otimes \Lambda''(g)) \circ \lambda_{\mathbf{I}} = \Lambda''(f \circ g) \quad \text{for } f, g \in \mathsf{Hom}_{\mathcal{C}}(A, A)
$$

We use all the above in a proof by induction of the desired conclusion.

- The base case is $F(\widehat{\varepsilon}) = F(\eta_\bullet) = \eta = \Lambda''(\mathrm{id}_A)$ (indeed, $\widehat{\varepsilon} = \eta_\bullet = (\bullet \mapsto \varepsilon)$ by definition).
- For the inductive case, write any word of length $n+1$ in Γ^* as wc for $w \in \Gamma^*$ with $|w| = n$ and $c \in \Gamma$, and suppose that by induction hypothesis $F(\widehat{w}) = \Lambda''(w)$. A direct computation of register transitions suffices to check that $\widehat{wc} = \mu_\bullet \circ (\widehat{w} \otimes \widehat{c})$. Then

$$
\begin{aligned}
F(\widehat{wc}) &= F(\mu_\bullet) \circ F(\widehat{w} \otimes \widehat{c}) = \mu \circ (F(\widehat{w}) \otimes F(\widehat{c})) \circ \lambda_{\mathbf{I}} \\
&= \mu \circ \left(\Lambda''\left(f_{w[n]} \circ \cdots \circ f_{w[1]}\right) \otimes \Lambda''(f_c)\right) \circ \lambda_{\mathbf{I}} = \Lambda''\left(f_{w[1]} \circ \cdots \circ f_{w[n]} \circ f_c\right) \qquad \square
\end{aligned}
$$

3.3. The syntactic category $\mathcal{L}$ of purely linear $\lambda\ell^{\oplus\&}$-terms.

Now we relate our notion of generalized SSTs to the $\lambda\ell^{\oplus\&}$-calculus. If $\Gamma = \{b_1, \ldots, b_n\}$ is an alphabet, call $\widetilde{\Gamma}$ the non-linear typing context

$$
\widetilde{\Gamma} = (b_1 : \mathbb{o} \multimap \mathbb{o}, \, \ldots, \, b_n : \mathbb{o} \multimap \mathbb{o}, \, \varepsilon : \mathbb{o})
$$

Definition 3.20. We call $\mathcal{L}(\widetilde{\Gamma})$ (or just $\mathcal{L}$ when Γ is clear from the context) the category

- whose objects are purely linear $\lambda\ell^{\oplus\&}$ types;
- whose morphisms from τ to σ are terms t such that $\widetilde{\Gamma}; \cdot \vdash t : \tau \multimap \sigma$, considered *up to $\beta\eta$-equivalence*;
- whose identity is given by $\lambda x.x$ and composition of f and g by $\lambda x.\, f\,(g\,x)$.

Remark 3.21. In the definition of morphisms, represented by $\lambda\ell^{\oplus\&}$ terms, we only make a restriction on the types of the $\lambda\ell^{\oplus\&}$-terms. Because $\lambda\ell^{\oplus\&}$ is normalizing (Theorem B.1), we could have further assumed the terms to be normal and thus, to only contain subterms whose types are also purely linear. Therefore, it makes sense to say that $\mathcal{L}(\widetilde{\Gamma})$ is about the *purely linear fragment of $\lambda\ell^{\oplus\&}$* augmented with (inert) constants from $\widetilde{\Gamma}$.

Similarly, for the equivalence relation we use to actually define the homsets, we use the full $\beta\eta$-equivalence, which could, on the face of it, require to go ouside of the purely linear fragment of $\lambda\ell^{\oplus\&}$ to establish certain equalities (for instance, consider the rather artifical derivation $\lambda x.x =_{\beta\eta} \lambda x.(\lambda^! y.y)\ x =_{\beta\eta} \lambda x.x$). This is also unnecessary as it can be checked that the normalization argument relies on a reduction relation $\to_{\beta\varepsilon}$ which is confluent up to commutative conversions $\approx_c$ (Theorem B.4); since both $\to_{\beta\varepsilon}$ and $\approx_c$ preserve the purely linear fragment, this is enough to conclude that we could ignore non-purely linear terms when defining $=_{\beta\eta}$ for the purely linear fragment.

$\mathcal{L}$ is a monoidal closed category with products and coproducts, which captures the expressiveness of purely linear $\lambda\ell^{\oplus\&}$-terms enriched with constants for the "empty word" and prepending letters of Γ to the left of a "word" when regarding the type $\mathbb{o}$ being regarded as the type of such words. This leads to the expected notion of streaming setting.

Definition 3.22. $\mathfrak{L}$ is the streaming setting $(\mathcal{L}, \mathbf{I}, \mathbb{o}, (\!|-|\!)_{\mathfrak{L}})$ with output Γ^* such that $(\!|t|\!)_{\mathfrak{L}} = w$ if and only if[19] $\lambda^! b_1. \ldots . \lambda^! b_n.\ \lambda^! \varepsilon.\ t$ is $\beta\eta$-equivalent to the Church encoding of w (this defines a total function because of Proposition C.5).

Our interest in $\mathfrak{L}$ lies in the following lemma. (Recall that a $\mathfrak{C}$-SST is said to be *single-state* if its set of states is a singleton.)

Lemma 3.23. *A function $\Sigma^* \to \Gamma^*$ is computable by a single-state $\mathfrak{L}$-SSTs if and only if it is $\lambda\ell^{\oplus\&}$-definable in the sense of Definition 2.12.*

To prove one direction of this equivalence, we need a technical lemma on $\lambda\ell^{\oplus\&}$-terms defining string functions. In order to state the lemma in the more general case of tree functions, so that it may be reused in Section 5, we extend the notation $\widetilde{\Gamma}$ above as follows: given a ranked alphabet $\boldsymbol{\Sigma} = \{a_1 : A_1,\ \ldots,\ a_n : A_n\}$ (recall the notation from §2.1), let

$$\widetilde{\boldsymbol{\Sigma}} = (a_1 : \mathbb{o} \multimap \ldots \multimap \mathbb{o},\ \ldots,\ a_n : \mathbb{o} \multimap \ldots \multimap \mathbb{o})$$

where the type of a_i has $|A_i|$ arguments (thus, it contains $|A_i| + 1$ occurrences of $\mathbb{o}$).

Lemma 3.24. *Let $\boldsymbol{\Sigma} = \{a_1 : A_1, \ldots, a_n : A_n\}$ and $\boldsymbol{\Gamma} = \{b_1 : B_1, \ldots, b_k : B_k\}$ be ranked alphabets such that there is some $A_i = \varnothing$ (i.e., $\mathbf{Tree}(\boldsymbol{\Sigma}) \neq \varnothing$). Up to $\beta\eta$-equivalence, every term of type $\mathrm{Tree}_{\boldsymbol{\Sigma}}[\kappa] \multimap \mathrm{Tree}_{\boldsymbol{\Gamma}}$ is of the shape*

$$\lambda s.\lambda^! b_1. \ldots . \lambda^! b_k.\ o\ (s\ d_1\ \ldots\ d_n)$$

such that o and the d_i are purely linear $\lambda\ell^{\oplus\&}$-terms with no occurrence of s. In particular:

$$\widetilde{\boldsymbol{\Gamma}}; \cdot \vdash o : \kappa \multimap \mathbb{o} \qquad\qquad \widetilde{\boldsymbol{\Gamma}}; \cdot \vdash d_i : \kappa \multimap \ldots \multimap \kappa$$

(with the type of d_i having $|B_i|$ arguments).

We expend considerable effort in proving this lemma; some of it is spent on routine yet cumbersome bureaucracy, and some on actual technical subtleties. But since the obstacles are unrelated to the various conceptual points concerning automata and semantics that we wished to stress, we relegate the proof to Appendix C.

The idea is to analyze the shape of $\lambda\ell^{\oplus\&}$-terms in *normal form*. For this purpose, the naive notion of β-normal form, that is, the non-existence of β-reductions from a term, is inadequate because of the *positive* connectives $\otimes/\oplus$. We thus start by defining a better

---

[19]Recall that this defines a total function because of the bijection between Church encodings and normal forms; see Proposition C.5.

notion of normal form, that can be reached by combining β-reductions with applications of η-conversions to unlock new β-redexes. This is done in Appendix B, where we prove a normalization theorem. Appendix C then provides a lengthy case analysis of normal forms inhabiting the type $\mathrm{Tree}_{\boldsymbol{\Sigma}}[\kappa] \multimap \mathrm{Tree}_{\boldsymbol{\Gamma}}$ to establish Lemma 3.24.

Similar issues arise in the literature concerned with deciding $\beta\eta$-convertibility in λ-calculi with positive connectives (see [Sch16] for a comprehensive and pedagogical overview of this subject). In our case, we are interested in normal forms only because they lend themselves to syntactic analysis.

We can now use this to establish the equivalence between $\lambda\ell^{\oplus\&}$-definability and $\mathfrak{L}$-SSTs.

Proof of Lemma 3.23. Before beginning the proof, it should be noted that SSTs process strings from left to right while Church encodings work rather from right to left. This is not a big issue in the presence of higher-order functions.

$\left(\mathfrak{L}\text{-SST} \subseteq \lambda\ell^{\oplus\&}\right)$ Given a $\mathfrak{L}$-SST $\mathcal{T} = (\{*\}, *, \tau, \delta, i, o)$, δ may be regarded as family of $\lambda\ell^{\oplus\&}$ terms $(t_a)_{a\in\Sigma}$ (with free variables in $\widetilde{\Gamma}$). Suppose that $\Sigma = \{a_1, \ldots, a_k\}$ and recall that Example 2.14 provides a $\lambda\ell^{\oplus\&}$-term $\mathtt{rev} : \mathrm{Str}_{\Sigma}[\mathbb{o} \multimap \mathbb{o}] \multimap \mathrm{Str}_{\Sigma}$ implementing the reversal of its input string. $[\![\mathcal{T}]\!]$ is implemented by the following $\lambda\ell^{\oplus\&}$-term of type $\mathrm{Str}_{\Sigma}[\tau \multimap \tau] \multimap \mathrm{Str}_{\Gamma}$:

$$\lambda s.\ \lambda^! b_1.\ \ldots\ \lambda^! b_n.\ \lambda^! \varepsilon.\ o\ (\mathtt{rev}\ s\ t_{a_1} \ldots t_{a_k}\ (i\ ()))$$

$\left(\lambda\ell^{\oplus\&} \subseteq \mathfrak{L}\text{-SST}\right)$ Given a term of type $\mathrm{Str}_{\Sigma}[\tau] \multimap \mathrm{Str}_{\Gamma}$, by Lemma 3.24, it is $\beta\eta$-equivalent to

$$\lambda s.\ \lambda^! b_1.\ \ldots\ \lambda^! b_n.\ \lambda^! \varepsilon.\ t\ (s\ u_1\ \ldots\ u_k\ v)$$

where t, v and the u_i are some terms typable in $\widetilde{\Gamma}$. The underlying string function is computed by the $\mathfrak{L}$-SST

$$\mathcal{T} = (\{*\}, *, \tau \multimap \tau, \delta, \lambda x.x, \lambda f.o\ (f\ i))$$

where $\delta(a_i, *) = (*, \lambda g.\lambda x.a_i\ (g\ x))$. $\square$

Lemma 3.23 therefore enables us to reframe Theorem 1.1 as statement comparing the expressiveness of single-state $\mathfrak{L}$-SSTs and $\mathfrak{S}\mathfrak{R}$-SSTs. This motivates our abstract development focused on comparing the expressiveness of various $\mathfrak{C}$-SSTs.

Toward this goal, we shall construct morphisms of streaming settings from and to $\mathfrak{L}$. One of them is straightforward using our previous technical development.

Lemma 3.25. *There is a morphism of streaming settings* $\mathfrak{S}\mathfrak{R} \to \mathfrak{L}$.

We build this morphism using the generic construction of Corollary 3.19. Again, this is not strictly necessary; see Lemma 5.18 for a sketch of a "manual" definition of such a morphism in the case of trees. But Corollary 3.19 is needed for other purposes anyway (Section 4.1) and gives us the correctness proof almost "for free".

Proof. We first give the construction of the underlying functor. First, note that $\mathcal{L}$ has all cartesian products and internal homsets, given by the syntactic connectives on types with the same notations. Thus, it is symmetric monoidal closed and quasi-affine; we can therefore apply Corollary 3.19 to the base type $\mathbb{o}$, regarded as an object of $\mathcal{L}$, and to the family of endomorphisms $(\widetilde{\Gamma};\ \cdot \vdash c : \mathbb{o} \multimap \mathbb{o})_{c\in\Gamma} \in \mathsf{Hom}_{\mathcal{L}}\left(\mathbb{o}, \mathbb{o}\right)^{\Gamma}$. (Indeed, by definition, every letter

$c \in \Gamma$ serves as a variable c given the type $\mathbb{o} \multimap \mathbb{o}$ in the typing context $\widetilde{\Gamma}$.) This gives us $F : \mathcal{SR} \to \mathcal{L}$ such that $F(\varnothing) = \mathbf{I}$, $F(\{\bullet\}) = (\mathbb{o} \multimap \mathbb{o}) \mathbin{\&} \mathbf{I}$ and

$$\forall w \in \Gamma^*, \quad F(\widehat{w}) =_{\beta\eta} \lambda z.\, \mathsf{let}\ () = z\ \mathsf{in}\ \langle (\lambda x.\, w_1\, (w_2 \ldots (w_n\, x) \ldots)), () \rangle$$

Since $\mathbb{T}_{\mathfrak{SR}} = \varnothing$ and $\mathbb{T}_{\mathfrak{L}} = \mathbf{I}$, we can simply take $i = \mathrm{id}_{\mathbf{I}} : \mathbb{T}_{\mathfrak{L}} \to F(\mathbb{T}_{\mathfrak{SR}})$ as part of our morphism of streaming settings. The map $o : F(\bot\!\!\!\bot_{\mathfrak{SR}}) \to \bot\!\!\!\bot_{\mathfrak{L}}$ is more interesting. Since $\bot\!\!\!\bot_{\mathfrak{SR}} = \{\bullet\}$ and $\bot\!\!\!\bot_{\mathfrak{L}} = \mathbb{o}$, we can see that o must be a $\lambda\ell^{\oplus\&}$-term of type $((\mathbb{o} \multimap \mathbb{o}) \mathbin{\&} \mathbf{I}) \multimap \mathbb{o}$ in the context $\widetilde{\Gamma}$. The choice that works is $o = \lambda p.\, \pi_1(p)\, \varepsilon$ (recalling that $(\varepsilon : \mathbb{o}) \in \widetilde{\Gamma}$ stands for the empty string). Proving that $(\!| o \circ F(\widehat{w}) \circ i |\!)_{\mathfrak{L}} = (\!|\widehat{w}|\!)_{\mathfrak{SR}}$ for any $w \in \Gamma^*$ is merely a matter of performing $\beta\eta$-conversions starting from the above equation on $F(\widehat{w})$; we leave this to the reader. Since every $f \in \mathsf{Hom}_{\mathcal{SR}}(\mathbb{T}, \bot\!\!\!\bot)$ is of the form $f = \widehat{w}$ for some $w \in \Gamma^*$, this suffices to show that (F, i, o) fits the definition of a morphism of streaming settings. $\quad\square$

However, note that this does not alone tells us that $\mathfrak{SR}$-SSTs are subsumed by $\lambda\ell^{\oplus\&}$, as Lemma 3.23 only allows us to use single-state $\mathcal{L}$-SSTs. To circumvent this, we will later prove that a streaming setting with coproducts allows to simulate state with single-state SSTs, thus completing a proof of the easier direction of Theorem 1.1.

In order to build morphisms from $\mathfrak{L}$ to other streaming settings, we shall make use of the following:

Lemma 3.26. *Let $\mathfrak{C}$ be a streaming setting $(\mathcal{C}, \mathbf{I}, \bot\!\!\!\bot, (\!|-|\!))$ whose underlying category $\mathcal{C}$ is symmetric monoidal closed with monoidal unit $\mathbf{I}$. Suppose that $\mathcal{C}$ additionally has chosen finite products and coproducts. Let $(f_b)_{b \in \Gamma}$ be a family of morphisms $\mathsf{Hom}_{\mathcal{C}}(\bot\!\!\!\bot, \bot\!\!\!\bot)^{\Gamma}$ and $e \in \mathsf{Hom}_{\mathcal{C}}(\mathbb{T}, \bot\!\!\!\bot)$ a distinguished morphism such that $(\!|e|\!)$ is the empty word and, for every $g \in \mathsf{Hom}_{\mathcal{C}}(\mathbb{T}, \bot\!\!\!\bot)$, we have $(\!|f_b \circ g|\!) = b(\!|g|\!)$ (that is, f_b acts by concatenating the single-letter word b on the left).*

Then there is a canonical morphism $\mathfrak{L} \to \mathfrak{C}$ of streaming settings. Moreover the underlying functor is strong monoidal for $\oplus$, $\&$, $\otimes$ and preserves $\multimap$.

Without spelling out the details, this Lemma essentially states that $\mathcal{L}$ is *initial* among symmetric monoidal closed categories with products and coproducts. We do not offer a proof of this statement, which we consider folklore. The interested reader may find a similar development in [Bie94, Chapter 4] for the case of $\mathcal{L}(\varnothing)$ (i.e., where the $\lambda\ell^{\oplus\&}$-terms have no free variables). Let us note that, because of the specific way we defined $\mathcal{L}$, Remark 3.21, showing the conservativity of the congruence $=_{\beta\eta}$ of $\lambda\ell^{\oplus\&}$ over the purely linear fragment should be the first step in this proof. This is because the notion of symmetric monoidal closed category does not require the existence of an exponential modality !, and thus, all the equations in the initial symmetric monoidal closed category with products and coproducts should only satisfy those equations mentioning those constructs.

3.4. **The free coproduct completion (or finite states).**

3.4.1. *Definition and basic properties.* We give here an elementary definition of "the" free finite coproduct completion of categories $\mathcal{C}$ and some of its basic properties. The construction consists essentially in considering finite families of objects of $\mathcal{C}$ as "formal coproducts" (equivalently, one could use finite lists as in [Gal20, Definition 3]).

Definition 3.27. Let $\mathcal{C}$ be a category. The *free finite coproduct completion* $\mathcal{C}_{\oplus}$ is defined as follows:

- An object of $\mathcal{C}_\oplus$ is a pair $(U, (C_u)_{u \in U})$ consisting of a finite set U and a family of objects of $\mathcal{C}$ over U. We write those as formal sums $\bigoplus_{u \in U} C_u$ in the sequel.
- A morphism from $\bigoplus_{u \in U} C_u \to \bigoplus_{v \in V} C_v$ is a U-indexed family of pairs $(v_u, g_u)_{u \in U}$ with $v_u \in V$ and $g_u : C_u \to C_{v_u}$ in $\mathcal{C}$. In short,

$$\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\bigoplus_{u \in U} C_u, \bigoplus_{v \in V} C_v\right) = \prod_{u \in U} \sum_{v \in V} \mathsf{Hom}_\mathcal{C}(C_u, C_v)$$

- The identity at object $\bigoplus_{u \in U} C_u$ is the family $(u, \mathrm{id}_{C_u})_{u \in U}$. Given two composable maps

$$(w_v, h_v)_{v \in V} : \bigoplus_{v \in V} C_v \to \bigoplus_{w \in W} C_w \qquad \text{and} \qquad (v_u, g_u)_{u \in U} : \bigoplus_{u \in U} C_u \to \bigoplus_{v \in V} C_v$$

  the composite is defined to be the family

$$(w_{v_u}, h_{v_u} \circ g_u) : \bigoplus_{u \in U} C_u \to \bigoplus_{w \in W} C_w$$

There is a full and faithful functor $\iota_\oplus : \mathcal{C} \to \mathcal{C}_\oplus$ taking an object $C \in \mathsf{Obj}(\mathcal{C})$ to the one-element family $\bigoplus_1 C \in \mathsf{Obj}(\mathcal{C}_\oplus)$. Objects lying in the image of this functor will be called *basic* objects of $\mathcal{C}_\oplus$. The formal sum notation reflects that families $\bigoplus_{u \in U} C_u$ should really be understood as coproducts of those basic objects C_u. More generally, it is straightforward to check that, for any finite set I and family $\bigoplus_{u \in U_i} C_u$ over $i \in I$, canonical coproducts in $\mathcal{C}_\oplus$ can be computed as follows

$$\bigoplus_{i \in I} \bigoplus_{u \in U_i} C_{i,u} \quad = \quad \bigoplus_{(i,u) \in \sum_{i \in I} U_i} C_{i,u}$$

As advertised, this is a free finite coproduct completion in the following sense: for any functor $F : \mathcal{C} \to \mathcal{D}$ to a category $\mathcal{D}$ with finite coproducts, there is an extension $\widetilde{F} : \mathcal{C}_\oplus \to \mathcal{D}$ preserving finite coproducts making the following diagram commute:

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{F} & \mathcal{D} \\ {\scriptstyle \iota_\oplus}\downarrow & {\scriptstyle \widetilde{F}}\nearrow & \\ \mathcal{C}_\oplus & & \end{array}$$

and it is unique up to unique natural isomorphism under those conditions.

Finally, suppose that we are a monoidal structure on $\mathcal{C}$. Then, it is possible to extend it to a monoidal structure over $\mathcal{C}_\oplus$ in a rather canonical way: we require that $\otimes$ distributes over $\oplus$, i.e., that $A \otimes (B \oplus C) \cong (A \otimes B) \oplus (A \otimes C)$. Formally speaking, we set[20]

---

[20] For readers more familiar with the free cocompletion $\mathsf{Set}^{\mathcal{C}^{\mathrm{op}}}$ of $\mathcal{C}$, note that the coproduct-preserving functor E determined by

$$\begin{array}{ccc} & \mathcal{C} & \\ {\scriptstyle \iota_\oplus}\swarrow & & \searrow{\scriptstyle \mathbf{y}} \\ \mathcal{C}_\oplus & \dashrightarrow{\scriptstyle E} & \mathsf{Set}^{\mathcal{C}^{\mathrm{op}}} \end{array}$$

is full and faithful, as well as strong monoidal when $\mathsf{Set}^{\mathcal{C}^{\mathrm{op}}}$ is equipped with the *Day convolution* as monoidal product. The latter is computed as the following coend using a monoidal product $\otimes$ in $\mathcal{C}$

$$(P \otimes Q)(U) = \int^{V,W} \mathsf{Hom}_\mathcal{C}(U, V \otimes W) \times P(V) \times Q(W)$$

$$\left(\bigoplus_{u \in U} C_u\right) \otimes \left(\bigoplus_{v \in V} C_v\right) \quad = \quad \bigoplus_{(u,v) \in U \times V} C_u \otimes C_v$$

If $\mathbf{I}$ is the unit of the tensor product in $\mathcal{C}$, then the basic object $\bigoplus_1 \mathbf{I}$ is taken to be the unit of the tensor product in $\mathcal{C}_\oplus$. An affine symmetric monoidal structure on $\mathcal{C}$ can be lifted in a satisfactory manner to this new tensor product (in particular $\iota_\oplus(\top)$ is still a terminal object).

The following property, which is arguably obvious from the definition of $\mathcal{C}_\oplus$, will turn out to be quite important later.

Proposition 3.28. *For any $A, B \in \mathsf{Obj}(\mathcal{C}_\oplus)$, there is a natural isomorphism*

$$\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(-), A \oplus B\right) \quad \cong \quad \mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(-), A\right) + \mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(-), B\right)$$

Remark 3.29. Compare with the following natural isomorphism, which holds for any category $\mathcal{D}$ and $A, B \in \mathsf{Obj}(\mathcal{D})$:

$$\mathsf{Hom}_{\mathcal{D}}\left(A \oplus B, -\right) \quad \cong \quad \mathsf{Hom}_{\mathcal{D}}\left(A, -\right) \times \mathsf{Hom}_{\mathcal{D}}\left(B, -\right)$$

In fact, this characterizes the coproduct of A and B. Just like Proposition 2.20, this is an instance of a general equivalence between universal properties and natural bijections involving homsets.

3.4.2. *Conservativity over affine monoidal settings.* First, note that the coproduct completion can be lifted at the level of streaming settings.

Definition 3.30. Given a streaming setting $\mathfrak{C} = (\mathcal{C}, \top, \bot\!\!\bot, (\!|-|\!)_{\mathcal{C}})$, define $\mathfrak{C}_\oplus$ as the tuple

$$(\mathcal{C}_\oplus, \iota_\oplus(\top), \iota_\oplus(\bot\!\!\bot), (\!|-|\!)_{\mathcal{C}_\oplus})$$

where $(\!|-|\!)_{\mathcal{C}_\oplus}$ is obtained by precomposing the canonical isomorphism (recalling that $\iota_\oplus$ is full and faithful)

$$\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(\top), \iota_\oplus(\bot\!\!\bot)\right) \cong \mathsf{Hom}_{\mathcal{C}}\left(\top, \bot\!\!\bot\right)$$

Before moving on, let us make the following definition: an object A in a monoidal category $(\mathcal{C}, \otimes, \mathbf{I})$ is said to have *unitary support* if there exists a map $\mathbf{I} \to A$. This is quite useful in affine categories for transductions, as it ensures the following.

Lemma 3.31. *Let $\mathcal{C}$ be a symmetric affine monoidal category. Then, for any pair of finite families $(C_u)_{u \in U}$ and $(C_v)_{v \in V}$ of objects of $\mathcal{C}$ such that all C_u and C_v have unitary support, we have a $U \times V$-indexed family of embeddings*

$$\mathtt{padwithjunk}_{u,v} : \mathsf{Hom}_{\mathcal{C}}\left(C_u, C_v\right) \to \mathsf{Hom}_{\mathcal{C}}\left(\bigotimes_{u \in U} C_u, \bigotimes_{v \in V} C_v\right)$$

The basic idea behind Lemma 3.31 can be pictured using string diagrams as in Figure 6: a morphism $C_u \to C_v$ can be pictured as a single string, which is to be embedded in a diagram with U-many inputs and V-many outputs. The fact that $\mathcal{C}$ is affine allows us to cut all input strings for $u' \neq u$ using a weakening node, and unitary support allow us to create some "junk" strings with no input to connect to those $v' \neq v$. This might fail for arbitrary symmetric affine monoidal categories: take for instance the category of finite sets and surjections between them, with the coproduct as a monoidal product.
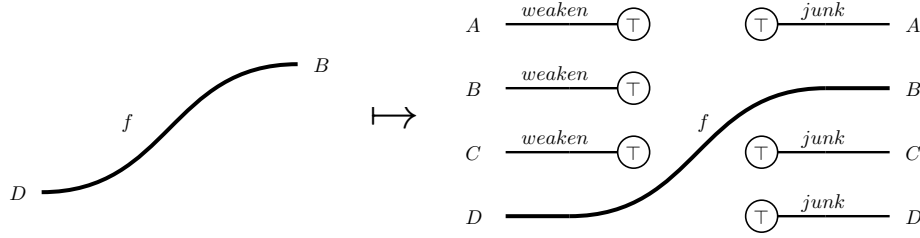
Figure 6: $\mathtt{padwithjunk}_{D,B} : \mathsf{Hom}_{\mathcal{C}}(D,C) \to \mathsf{Hom}_{\mathcal{C}}(A \otimes B \otimes C \otimes D, A \otimes B \otimes C \otimes D)$

We are now ready to state our first theorem asserting that, in those favorable circumstances, coproduct completions do not give rise to more expressive SSTs.

Theorem 3.32. *Let $\mathfrak{C}$ be an affine symmetric monoidal streaming setting where all objects C such that $\mathsf{Hom}_{\mathcal{C}}(\mathbb{T}, C) \neq \varnothing \neq \mathsf{Hom}_{\mathcal{C}}(C, \bot\!\!\!\bot)$ have unitary support.*
$\mathfrak{C}$-SSTs are equivalent to $\mathfrak{C}_{\oplus}$-SSTs.

Proof. Since $\iota_{\oplus}$ extends to a morphism of streaming settings, $\mathfrak{C}_{\oplus}$-SSTs subsume $\mathfrak{C}$-SSTs.

Conversely, let $\mathcal{T} = \big(Q, q_0, \bigoplus_{u \in U} C_u, \delta, i, o\big)$ be a $\mathfrak{C}_{\oplus}$-SST with input Σ^* and C_u basic objects. Then, we construct a $\mathfrak{C}$-SST

$$\mathcal{T}' = \left(Q \times U, \ (q_0, u_0), \ \bigotimes_{u \in U} C_u, \ \delta', \ i_{u_0}, \ o' \right)$$

such that $[\![\mathcal{T}]\!] = [\![\mathcal{T}']\!]$. We define successively (u_0, i_{u_0}), δ' and o'.

- We have $i \in \mathsf{Hom}_{\mathcal{C}_{\oplus}}\big(\iota_{\oplus}(\mathbb{T}), \bigoplus_{i \in I} C_i\big)$ which can be rewritten as a factorization

$$\iota_{\oplus}(\mathbb{T}) = \bigoplus_{1} \mathbb{T} \xrightarrow{(\mathsf{in}_{u_0}, * \mapsto \mathsf{id})} \bigoplus_{U} \mathbb{T} \xrightarrow{(\mathsf{id}, u \mapsto i_u)} \bigoplus_{u \in U} C_u$$

for some $u_0 \in U$ (the i_u for $u \neq u_0$ are taken arbitrarily thanks to the assumption that the C_u have unitary support). This u_0 is the second component of the initial state of $\mathcal{T}'$.

- We set $\delta'(a, (q, u)) = (q', \alpha_u(f'))$ if $\delta(a, q) = (q', f')$, where

$$(\alpha_u)_{u \in U} \ : \ \prod_{u \in U} \left[\mathsf{Hom}_{\mathcal{C}_{\oplus}}\left(\bigoplus_{u \in U} C_u, \ \bigoplus_{u \in U} C_u \right) \to \mathsf{Hom}_{\mathcal{C}_{\oplus}}\left(\bigotimes_{u \in U} C_u, \ \bigotimes_{u \in U} C_u \right) \right]$$

is defined by taking the pointwise composite of

$$\widetilde{\alpha}_u : \quad \mathsf{Hom}_{\mathcal{C}_{\oplus}}\left(\bigoplus_{u \in U} C_u, \ \bigoplus_{u' \in U} C_{u'} \right) \quad \to \quad \sum_{u' \in U} \mathsf{Hom}_{\mathcal{C}_{\oplus}}(C_u, C_{u'})$$

$$\beta_u : \quad \sum_{u' \in U} \mathsf{Hom}_{\mathcal{C}_{\oplus}}(C_u, C_{u'}) \quad \to \quad U \times \mathsf{Hom}_{\mathcal{C}_{\oplus}}\left(\bigotimes_{u \in U} C_u, \ \bigotimes_{u \in U} C_u \right)$$

$$\pi : \quad U \times \mathsf{Hom}_{\mathcal{C}_{\oplus}}\left(\bigotimes_{u \in U} C_u, \ \bigotimes_{u \in U} C_u \right) \quad \to \quad \mathsf{Hom}_{\mathcal{C}_{\oplus}}\left(\bigotimes_{u \in U} C_u, \ \bigotimes_{u \in U} C_u \right)$$

where $\widetilde{\alpha}_u$ is obtained by evaluating its input $f \in \prod_{u \in U} \sum_{u' \in U} \mathsf{Hom}_{\mathcal{C}_{\oplus}}(C_u, C_{u'})$ at u, $\beta_u = \sum_{u'} \mathtt{padwithjunk}_{u,u'}$ (with $\mathtt{padwithjunk}$ given as per Lemma 3.31) and π taken to be the second projection.

- Finally, we set $o'(q, u) \in \mathsf{Hom}_{\mathcal{C}} \left(\bigotimes_{v \in U} C_v, \bot\!\!\!\bot \right)$ to $\widetilde{o}_u \in \mathsf{Hom}_{\mathcal{C}} \left(C_u, \bot\!\!\!\bot \right)$ precomposed with the projection $\pi_u \in \mathsf{Hom}_{\mathcal{C}} \left(\bigotimes_{v \in U} C_v, C_u \right)$.

To conclude, one can check by induction that $[\![\mathcal{T}]\!](w) = [\![\mathcal{T}']\!](w)$ for every $w \in \Sigma^*$. $\qquad \square$

Corollary 3.33. $\mathfrak{SR}_{\oplus}$*-SSTs are equivalent to* $\mathfrak{SR}$*-SSTs.*

Proof. All objects of $\mathcal{SR}$ have unitary support via an induction: tensor with the map $\varepsilon : \mathbf{I} \to \Gamma^*$ corresponding to the empty word at the recursive step. $\qquad \square$

3.4.3. *State-dependent memory SSTs.* The free coproduct completion encourages us to define the notion of *state-dependent memory SST*, generalizing usual copyless SSTs as follows: instead of taking a single object $C \in \mathsf{Obj}(\mathcal{C})$ as an abstract infinitary memory, we allow to take a family $(C_q)_{q \in Q} \in \mathsf{Obj}(\mathcal{C})^Q$ indexed by the states of the SST.

Definition 3.34. A *state-dependent memory $\mathfrak{C}$-SST* (henceforth abbreviated sdm-$\mathfrak{C}$-SST or sdmSST when $\mathfrak{C}$ is clear form context) with input Σ^* is a tuple $(Q, q_0, \delta, (C_q)_{q \in Q}, i, o)$ where

- Q is a finite set of states
- $q_0 \in Q$ is some initial state
- $\delta : \Sigma \to \prod_{q \in Q} \sum_{r \in Q} \mathsf{Hom}_{\mathcal{C}} \left(C_q, C_r \right)$ is a transition function
- $i \in \mathsf{Hom}_{\mathcal{C}} \left(\top, C_{q_0} \right)$ is the initialization morphism
- $o \in \prod_{q \in Q} \mathsf{Hom}_{\mathcal{C}} \left(C_q, \bot\!\!\!\bot \right)$ is the output family of morphism

In the sequel, we shall often use sdmSSTs because we find them convenient to give more elegant constructions that produce little "junk", as is encoded in Lemma 3.31. They essentially give the full power of coproducts in any given situation as shown below.

Lemma 3.35. *Let $\mathfrak{C}$ be a streaming setting. State-dependent memory $\mathfrak{C}$-SSTs are exactly as expressive as $\mathfrak{C}_{\oplus}$-SSTs.*

Proof sketch. Given a $\mathfrak{C}_{\oplus}$-SST $\left(Q, q_0, \bigoplus_{u \in U} C_u, \delta, i, o \right)$ where $i(*) = \mathsf{in}_{u_0}(i')$ one may check that the following sdm-$\mathfrak{C}$-SST computes the same function:

$$\left(Q \times U, (q_0, u_0), (C_u)_{(q,u) \in Q \times U}, \delta', i', (o(q)_u)_{(q,u) \in Q \times U} \right)$$

where $\delta'(a)_{q,u} = ((r, u'), f)$ if and only if $\delta(a, q) = (r, v)$ and $v_u = (u', f)$.

Conversely, letting $(Q, q_0, (C_q)_{q \in Q}, \delta, i, o)$ be a sdm-$\mathfrak{C}$-SST, an equivalent $\mathfrak{C}_{\oplus}$-SST is given by $(Q, q_0, \bigoplus_{q \in Q} C_q, \delta', \mathsf{in}_{q_0}(i), o')$, where it is sufficient to define $o'(q)$ as $(o_q)_q$ and to ensure that if $\delta'(a, q) = (r, (r_{q'}, f_{q'})_{q' \in Q})$, then $\delta(a)_q = (r, f_q)$ and $r_q = r$. This can be done. $\qquad \square$

Finally, let us remark that the notions of single-state, "normal" and state-dependent memory $\mathfrak{C}$-SSTs coincide if $\mathfrak{C}$ has all coproducts.

Lemma 3.36. *If $\mathfrak{C}$ is a streaming setting with coproducts, single-state $\mathfrak{C}$-SSTs are as expressive as general $\mathfrak{C}$-SSTs and sdm-$\mathfrak{C}$-SSTs.*

Proof. Take a sdmSST $(Q, q_0, (C_q)_{q \in Q}, \delta, i, o)$ to the single-state SST

$$\left(\{\bullet\}, \bullet, \bigoplus_{q \in Q} C_q, \delta', \mathsf{in}_{q_0} \circ i, [o(q)]_{q \in Q} \right)$$

where δ' is defined from δ through the maps

$$
\left[\prod_{q\in Q}\sum_{r\in Q}\mathsf{Hom}_{\mathcal{C}}\left(C_q, C_r\right)\right]^{\Sigma} \rightarrow \left[\prod_{q\in Q}\mathsf{Hom}_{\mathcal{C}}\left(C_q, \bigoplus_{r\in Q}C_r\right)\right]^{\Sigma} \xrightarrow{\sim} \left[\mathsf{Hom}_{\mathcal{C}}\left(\bigoplus_{q\in Q}C_q, \bigoplus_{r\in Q}C_r\right)\right]^{\Sigma}
$$

$\square$

Remark 3.37. The comparison between single-state, standard and state-dependent memory $\mathfrak{C}$-SSTs can be summed up in terms of completion with the following "equalities":

$$\mathfrak{C}\text{-SSTs} \quad = \quad \text{single-state } \mathfrak{C}_{\oplus\mathrm{const}}\text{-SSTs} \qquad\qquad \text{sdm-}\mathfrak{C}\text{-SSTs} \quad = \quad \text{single-state } \mathfrak{C}_{\oplus}\text{-SSTs}$$

where $\mathfrak{C}_{\oplus\mathrm{const}}$ designates the following restriction of $\mathfrak{C}_{\oplus}$: the category $\mathcal{C}_{\oplus}$ is restricted to the full subcategory $\mathcal{C}_{\oplus\mathrm{const}}$ whose objects are constant formal sums $\bigoplus_{i\in I} C$ for some $C \in \mathsf{Obj}(\mathcal{C})$.

3.4.4. *Some function spaces in $\mathcal{SR}_{\oplus}$.* Now we study $\mathcal{SR}_{\oplus}$ in some more detail. This category is unfortunately not able to interpret even the $\otimes/\multimap$ fragment of $\lambda\ell^{\oplus\&}$, because, like $\mathcal{SR}$, it lacks internal homsets $A \multimap B$ for every pair $(A,B) \in \mathsf{Obj}(\mathcal{SR}_{\oplus})^2$. However, they exist when A lies in the image of $\iota_{\oplus}$. It will turn out to be very useful later on.

Now we reframe a useful technical argument, typically made when dealing with determinization and composition of standard copyless SSTs to obtain the internal homsets we desire. The core of this argument (sketched in [BC18, p.206-207]) is that register transitions may be effectively coded using a combination of state and a larger set of registers. Here, the intuition is that the free coproduct completion allows the category $\mathcal{SR}_{\oplus}$ to integrate all the features of the additional finite set of states provided by the SSTs.

Lemma 3.38. *Let $R, S \in \mathsf{Obj}(\mathcal{SR})$. There is an internal homset $\iota_{\oplus}(R) \multimap \iota_{\oplus}(S)$ in $\mathcal{SR}_{\oplus}$.*

In Example 3.39, we work through the proof below in a concrete case.

Proof. First, recall that $\iota_{\oplus}$ is full and faithful, and that it is thus pertinent to focus our preliminary analysis on morphisms in $\mathcal{SR}$. Recall that a register transition $f : R \to S$, which is a set-theoretical map $S \to (\Gamma + R)^*$ where for every $r \le R$, $\sum_{s\in S}|f(s)|_{\mathsf{in}_2(r)} \le 1$ (i.e., it is copyless). Consider the map $(\Gamma + R)^* \to R^*$ erasing the letters of Γ. Then, the image of the induced map $p : \mathsf{Hom}_{\mathcal{SR}}(R,S) \to [S \to R^*]$ is clearly finite because of copylessness. In fact, letting $\mathsf{LO}(X)$ be the set of all total orders over some set X, we have an isomorphism between the image of $\mathsf{Hom}_{\mathcal{SR}}(R,S)$ under p and the following dependent sum

$$
\mathcal{O}(R,S) \quad = \quad \sum_{\hat{f}:R\rightharpoonup S}\prod_{s\in S}\mathsf{LO}(\hat{f}^{-1}(s))
$$

The intuition is that $\hat{f}$ tracks where register variables in R get affected and the additional data encode in which order they appear in an affectation. Once this crucial finitary information $(\hat{f}, (<_s)_{s\in S})$ is encoded in the internal homset using coproducts, it only remains to recover the information we erased with p, i.e. what words in Γ^* located between occurrences of register variables. This information cannot be bounded by the size of R and S, but the number of intermediate words can; we may index them by $S + \mathsf{dom}(\hat{f})$.

Putting everything together, it means that we take

$$
\iota_{\oplus}(R) \multimap \iota_{\oplus}(S) \quad = \quad \bigoplus_{(\hat{f},<)\in\mathcal{O}(R,S)} \iota_{\oplus}(S + \mathsf{dom}(\hat{f}))
$$

Now, we need to define the evaluation map $\mathrm{ev}_{R,S} : [\iota_\oplus(R) \multimap \iota_\oplus(S)] \otimes \iota_\oplus(R) \to \iota_\oplus(S)$. Recall that the tensor distributes over $\oplus$, so we really need to exhibit $\mathrm{ev}_{R,S}$ in

$$\mathsf{Hom}_{\mathcal{SR}_\oplus} \left(\bigoplus_{(\hat{f},<)} \iota_\oplus(S + \mathsf{dom}(\hat{f}) + R) , \; \iota_\oplus(S) \right) \; \cong \; \prod_{(\hat{f},<)} \mathsf{Hom}_{\mathcal{SR}} \left(S + \mathsf{dom}(\hat{f}) + R , \; S \right)$$

where the indices $(\hat{f}, <)$ ranges over $\mathcal{O}(R,S)$ on both sides. Call $\mathrm{ev}_{R,S,\hat{f},<}$ the corresponding family of $\mathcal{SR}$-morphisms, whose members are set-theoretic maps $S \to (S + \mathsf{dom}(\hat{f}) + R)^*$. Calling $\{r_1, \ldots, r_k\}$ the subset of $\mathsf{dom}(\hat{f})$ ordered by $r_1 < \ldots < r_k$, we set

$$\mathrm{ev}_{R,S,\hat{f},<}(s) \;=\; \mathsf{in}_0(s)\mathsf{in}_2(r_1)\mathsf{in}_1(r_1) \ldots \mathsf{in}_2(r_k)\mathsf{in}_1(r_k)$$

This concludes the definition of ev. We now leave checking that this satisfies the required universal property to the reader. $\square$

Example 3.39. Let us illustrate this construction in a simple case. Consider the following register transition for the concrete base alphabet $\{a, b\}$ and register names x, y, z, u, r, s:

$$
\begin{aligned}
r &\leftarrow zaxabyaa \\
s &\leftarrow bab
\end{aligned}
$$

Up to the evident isomorphism $\{x, y, z, u\} \cong \{x\} \otimes \{y, z, u\}$, this determines a morphism $f \in \mathsf{Hom}_{\mathcal{SR}} (\{x\} \otimes \{y, z, u\}, \{r, s\})$. Let us describe the unique map $\Lambda(f)$ in

$$\mathsf{Hom}_{\mathcal{SR}_\oplus} (\iota_\oplus(\{x\}), \; \iota_\oplus(\{y, z, u\}) \multimap \iota_\oplus(\{r, s\})) \cong \sum_{\hat{f}, <} \mathsf{Hom}_{\mathcal{SR}} \left(\{x\}, \{r, s\} + \mathsf{dom}(\hat{f}) \right)$$

such that $\mathrm{ev} \circ (\Lambda(f) \otimes \mathsf{id}) = f$. On its first component, we set $\hat{f}(y) = \hat{f}(z) = r$ and leave it undefined on u; as for the order, we set $z <_r y$. The last component corresponds to the register transition h depicted below on the left, where, for legibility, we write $\mathbf{r}, \mathbf{s}, \mathbf{z}$ and $\mathbf{y}$ for $\mathsf{in}_1(r), \mathsf{in}_1(s), \mathsf{in}_2(z)$ and $\mathsf{in}_2(y)$. Using the same notations, we also display on the right the relevant component of $\mathrm{ev}_{\{y,z,u\},\{r,s\},\hat{f},<} \in \mathsf{Hom}_{\mathcal{SR}} ((\{r, s\} + \{z, y\}) + \{z, y\}, \{r, s\})$, so that the reader may convince themself that the composite $\mathrm{ev}_{\{y,z,u\},\{r,s\},\hat{f},<} \circ (h \otimes \mathsf{id}_{\{y,z,u\}})$ is indeed f.

$$
\begin{aligned}
\mathbf{r} &\leftarrow \varepsilon \\
\mathbf{z} &\leftarrow axab \\
\mathbf{y} &\leftarrow aa \\
\mathbf{s} &\leftarrow bab
\end{aligned}
\qquad\qquad
\begin{aligned}
r &\leftarrow \mathsf{in}_1(\mathbf{r}) \, \mathbf{y} \, \mathsf{in}_1(\mathbf{y}) \, \mathbf{z} \, \mathsf{in}_1(\mathbf{z}) \\
s &\leftarrow \mathsf{in}_1(\mathbf{s})
\end{aligned}
$$

Lemma 3.38 can be extended to define internal homsets $\iota_\oplus(R) \multimap C$ for arbitrary $C \in \mathsf{Obj}(\mathcal{SR}_\oplus)$ through the natural isomorphism of Proposition 3.28. However, extending this to all homsets (i.e. allowing any object of $\mathcal{SR}_\oplus$ in the left-hand side) seems impossible: the lack of *products* prevents us from doing so.

3.5. **The product completion (or non-determinism).** The above point (among others) leads us to study the free finite product completion of streaming settings. As for coproducts, we first discuss the categorical construction before turning to the expressiveness. Here, the situation is more intricate as it turns out that sdm-$\mathfrak{C}_\&$-SSTs of interest will roughly have the power of *non-deterministic* sdm-$\mathfrak{C}$-SSTs. We make that connection precise.

Thankfully, a determinization theorem for usual copyless SSTs, i.e. $\mathfrak{SR}$-SSTs, exists in the literature [AD11] (the proof in the given reference is indirect and goes through monadic second-order logic). It could be applied without difficulty to show that non-determinism does not increase the power of sdm-$\mathfrak{SR}$-SSTs.

To keep the exposition self-contained *and* illustrate our framework, we give in Section 4.2 a direct determinization argument generalized to our setting by using, crucially, the concept of internal homsets (and Lemma 3.38 for the desired application).

3.5.1. *Definition and basic properties.* The product completion can, of course, be defined as the dual of the coproduct completion.

Definition 3.40. Let $\mathcal{C}$ be a category. Its *free finite product completion* is $\mathcal{C}_\& = ((\mathcal{C}^{\mathrm{op}})_\oplus)^{\mathrm{op}}$.

While conceptually immaculate, this definition merits a bit of unfolding. The objects of $\mathcal{C}_\&$ are still finite families $(C_x)_{x \in X}$ – in this context, we write them as formal products $\&_{x \in X} C_x$. As for homsets, we have the dualized situation

$$\mathsf{Hom}_{\mathcal{C}_\&} \left(\underset{x \in X}{\&} C_x, \underset{y \in Y}{\&} C_y \right) \quad \cong \quad \prod_{y \in Y} \sum_{x \in X} \mathsf{Hom}_{\mathcal{C}_\&} (C_x, C_y)$$

We also a full and faithful functor $\iota_\& : \mathcal{C} \to \mathcal{C}_\&$ with a similar universal property as for the coproduct completion.

As with the coproduct completion, one may want to produce a tensor product in $\mathcal{C}_\oplus$ if the underlying category $\mathcal{C}$ has one. The very same recipe can be applied: we define the tensor so that the distributivity $A \otimes (B \& C) \cong (A \otimes B) \& (A \otimes C)$ holds.

$$\left(\underset{x \in X}{\&} C_x \right) \otimes \left(\underset{y \in Y}{\&} C_y \right) \quad = \quad \underset{(x,y) \in X \times Y}{\&} C_x \otimes C_y$$

Remark 3.41. One might be disturbed by this distributivity of $\otimes$ over $\&$, which goes against the non-linear intuition of thinking of $\&$ and $\otimes$ as "morally the same". This feeling may also be exacerbated by the familiar iso

$$\mathsf{Hom}_{\mathcal{SR}_\oplus} (\top, R \otimes S) \cong \mathsf{Hom}_{\mathcal{SR}_\oplus} (\top, R) \times \mathsf{Hom}_{\mathcal{SR}_\oplus} (\top, S)$$

This indeed becomes false when going from $\mathcal{SR}$ to $\mathcal{SR}_\&$. The useful mnemonic here (which is untrue for pure LL!) is that the *multiplicative* connective distributes over both *additive* connectives in the same way.

Remark 3.42. While $\mathcal{C}_\&$ inherits a monoidal product from $\mathcal{C}$ much like with the coproduct completion, it does *not* preserve the *affineness* of monoidal products. The product completion indeed adds a new terminal object, namely the empty family, which can *never* be isomorphic to the singleton family $\iota_\&(\top)$. More generally, $\iota_\&$ preserves colimits rather than limits.

It is also straightforward to extend the product completion at the level of streaming settings $\mathfrak{C} \mapsto \mathfrak{C}_\&$.

3.5.2. *Relationship with non-determinism.* At this juncture, our goal is to prove the equivalence between sdm-$\mathfrak{SR}_{\&}$-SSTs and sdm-$\mathfrak{SR}$-SSTs. One direction is trivial; for the other, we actually prove that sdm-$\mathfrak{SR}_{\&}$-SSTs are subsumed by sdm-$\mathfrak{SR}_{\oplus}$-SSTs.

This result involves some non-trivial combinatorics. We prove it via *uniformization* of non-deterministic sdm-$\mathfrak{SR}$-SSTs, a mild generalization of determinization[21].

Our non-deterministic devices make *finitely branching* choices, following the case of the usual non-deterministic SSTs [AD11, Section 2.2]. To express this, we use the notation $\mathcal{P}_{<\infty}(X)$ for the set of *finite* subsets of a set X. (Note that if Q is some finite set of states, we have the simplification $\mathcal{P}(X) = \mathcal{P}_{<\infty}(X)$.)

Definition 3.43. Let $\mathfrak{C}$ be streaming setting with output X. A *non-deterministic* sdm-$\mathfrak{C}$-SST with input Σ^* and output X is a tuple $(Q, I, (C_q)_{q\in Q}, \Delta, i, o)$ where
- Q is a finite set of states and $I \subseteq Q$
- $(C_q)_{q\in Q}$ is a family of objects of $\mathcal{C}$
- Δ is a finite transition relation: $\Delta \in \mathcal{P}_{<\infty}\left(\Sigma \times \sum_{(q,r)\in Q^2} \mathsf{Hom}_{\mathcal{C}}\left(C_q, C_r\right)\right)$
- $i \in \prod_{q_0 \in I} \mathsf{Hom}_{\mathcal{C}}\left(\top, C_{q_0}\right)$ is a family of input morphisms
- $o \in \prod_{q\in Q}\left(\mathsf{Hom}_{\mathcal{C}}\left(C_q, \bot\!\!\!\bot\right) + 1\right)$ is a family of partial output morphisms

A *partial* sdm-$\mathfrak{C}$-SST $(Q, q_0, (C_q)_{q\in Q}, \delta, i, o)$ has the same definition as a (deterministic) sdm-$\mathfrak{C}$-SST (Definition 3.34), except for the output o, which is allowed to be partial, just as in the last item above.

By transposing the usual notion of run of a non-deterministic finite automaton, one sees that a non-deterministic sdm-$\mathfrak{C}$-SST $\mathcal{T}$ gives rise to a function $[\![\mathcal{T}]\!] : \Sigma^* \to \mathcal{P}_{<\infty}(X)$ (for an input alphabet Σ and output set X). Similarly, a partial sdm-$\mathfrak{C}$-SST $\mathcal{T}'$ is interpreted as a partial function $[\![\mathcal{T}]\!] : \Sigma^* \rightharpoonup X$.

Remark 3.44. In line with Remark 3.37, we may describe non-deterministic sdm-$\mathfrak{C}$-SSTs as single-state (deterministic) SSTs over an enriched streaming setting.

Let $\mathfrak{C} = (\mathcal{C}, \top_{\mathfrak{C}}, \bot\!\!\!\bot_{\mathfrak{C}}, (\!|-\!|)_{\mathfrak{C}})$, with output X. We first define the category $\mathsf{NFA}(\mathcal{C})$:
- its objects consist of a finite set Q with a family $(C_q)_{q\in Q} \in \mathsf{Obj}(\mathcal{C})^Q$;
- its morphisms are $\mathsf{Hom}_{\mathsf{NFA}(\mathcal{C})}\left((C_q)_{q\in Q}, (C'_r)_{r\in R}\right) = \mathcal{P}_{<\infty}\left(\sum_{(q,r)\in Q\times R} \mathsf{Hom}_{\mathcal{C}}\left(C_q, C_r\right)\right)$
- the composition of morphisms extends that of binary relations:
$$\varphi \circ \psi = \{(q, s, f \circ g) \mid (q, r, f : C_q \to C'_r) \in \varphi, \ (r, s, g : C'_r \to C''_s) \in \psi\}$$

To lift this to a construction $\mathsf{NFA}(\mathfrak{C})$ on streaming settings, we take:
- $\top_{\mathsf{NFA}(\mathfrak{C})}$ and $\bot\!\!\!\bot_{\mathsf{NFA}(\mathfrak{C})}$ are the $\{\bullet\}$-indexed families containing respectively $\top_{\mathfrak{C}}$ and $\bot\!\!\!\bot_{\mathfrak{C}}$, so that $\mathsf{Hom}_{\mathsf{NFA}(\mathcal{C})}\left(\top_{\mathsf{NFA}(\mathfrak{C})}, \bot\!\!\!\bot_{\mathsf{NFA}(\mathfrak{C})}\right) = \mathcal{P}_{<\infty}\left(\{(\bullet, \bullet)\} \times \mathsf{Hom}_{\mathcal{C}}\left(\top_{\mathfrak{C}}, \bot\!\!\!\bot_{\mathfrak{C}}\right)\right)$
- $(\!|\varphi|)_{\mathsf{NFA}(\mathfrak{C})} = \{(\!|f|)_{\mathfrak{C}} \mid (\bullet, \bullet, f) \in \varphi\} \subseteq X$, so that the output set of a $\mathsf{NFA}(\mathfrak{C})$-SST is $\mathcal{P}_{<\infty}(X)$.

---

[21]We work with uniformization here we find it slightly more convenient to handle. This choice of uniformization over determinization is rather inessential.

There is a slight mismatch between the above definition of non-deterministic sdm-$\mathfrak{C}$-SSTs and single-state $\mathsf{NFA}(\mathfrak{C})$-SSTs: in the latter, two distinct input (resp. output) morphisms may correspond to the same initial (resp. final) state. However, the former can encode such situations by enlarging the set of states (to keep it finite, the use of $\mathcal{P}_{<\infty}(-)$ in the definitions is crucial).

One can also give an analogous account of partial sdm-$\mathfrak{C}$-SSTs; we leave the interested reader to work out the details.

Coming back to our main point, we now state the slight variation of the determinization theorem for copyless SSTs [AD11] that fits our purposes.

Definition 3.45. Given an arbitrary function $F : X \to \mathcal{P}(Y)$, we say that $f : X \rightharpoonup Y$ *uniformizes* F if and only if $\mathsf{dom}(f) = X \setminus F^{-1}(\varnothing)$ and $\forall x \in \mathsf{dom}(f).\ f(x) \in F(x)$.

Theorem 3.46. *For every non-derministic $\mathfrak{GR}$-SSTs $\mathcal{T}$ with input Σ^*, there exists a partial deterministic $\mathfrak{GR}$-SST $\mathcal{T}'$ such that $[\![\mathcal{T}']\!]$ uniformizes $[\![\mathcal{T}]\!]$.*

We now show that studying the uniformization property between classes of sdmSSTs parameterized by streaming setting $\mathfrak{C}$ and $\mathfrak{D}$ is morally the same as comparing the expressiveness of sdm-$\mathfrak{C}_\&$-SSTs and sdm-$\mathfrak{D}$-SSTs.

Lemma 3.47. *Non-deterministic sdm-$\mathfrak{C}$-SSTs are uniformizable by partial sdm-$\mathfrak{D}$-SSTs if and only if sdm-$\mathfrak{D}$-SSTs subsume sdm-$\mathfrak{C}_\&$-SSTs.*

Proof. First, let us assume that sdm-$\mathfrak{C}$-SSTs uniformize sdm-$\mathfrak{D}$-SSTs and let

$$\mathcal{T} \ = \ (Q, q_0, (A_q)_{q \in Q}, \delta, i, o) \qquad \text{with} \qquad A_q \ = \ \underset{x \in X_q}{\&} C_{q,x}$$

be a $\mathfrak{C}_\&$-SST with input Σ. We first define a non-deterministic sdm-$\mathfrak{C}$-SST $\mathcal{T}'$ by setting

$$\mathcal{T}' = \big(I + Q', I, (C_{p(m)})_{m \in I + Q'}, \Delta, i', o'\big)$$

$$
\begin{array}{ll}
Q' \ = \ \sum_{q \in Q} X_q & \qquad p: \quad I + Q' \ \to \ Q' \\
I \ = \ \sum_{x \in X_{q_0}} \{f \mid i_* = (x, f)\} & \qquad \quad\ \mathsf{in}_1(x, f) \ \mapsto \ (q_0, x) \\
& \qquad \quad\ \mathsf{in}_2(q, x) \ \mapsto \ (q, x)
\end{array}
$$

$$i'_{\mathsf{in}_1(x,f)} = f \qquad\qquad o'_m = \mathsf{in}_1\big(\pi_2\big((o_{\pi_1(p(m))})_{\pi_2(p(m))}\big)\big)$$

$$\Delta \ = \ \left\{ (a, ((m, \mathsf{in}_2(r, y)), f)_m) \ \middle|
\begin{array}{l}
\forall (x, q) \in Q'. \forall m \in p^{-1}(x, q). \\
\qquad \pi_1(\delta(a)_q) \ = \ r \\
\wedge \quad \pi_2(\delta(a)_q)_y \ = \ (x, f)
\end{array}
\right\}$$

Taking $\mathcal{T}''$ to be a sdm-$\mathfrak{D}$-SST uniformizing $\mathcal{T}'$, we have $\{-\} \circ [\![\mathcal{T}'']\!] = [\![\mathcal{T}']\!] = \{-\} \circ [\![\mathcal{T}]\!]$, so we are done.

For the converse, assume that sdm-$\mathfrak{D}$-SSTs subsume sdm-$\mathfrak{C}_\&$-SSTs and suppose we have some non-deterministic sdm-$\mathfrak{C}$-SST $\mathcal{T} = (Q, I, (A_q)_{q \in Q}, \Delta, i, o)$ to uniformize. Fix a total order $\preceq$ over the morphisms of $\mathcal{C}$ occurring in Δ (recall that there are finitely many of them). Consider a partial deterministic sdm-$\mathfrak{C}_\&$-SST $\mathcal{T}'$ obtained from $\mathcal{T}$ by a powerset construction

$$\mathcal{T}' \ = \ \left(\mathcal{P}(Q),\ I,\ \left(\underset{r \in R}{\&} A_r \right)_{R \subseteq Q},\ \delta,\ i,\ o' \right)$$

where $\delta(a)_R \;=\; (S, (r_s, f_s)_{s \in S})$ if and only if $\forall s \in S \left[\begin{array}{l} (a, ((r_s, s), f_s)) \in \Delta \\ \forall g.(a, ((r_s, s), g)) \in \Delta \Rightarrow f_s \preceq g \end{array} \right.$

and $o'_R = o \upharpoonright r$ for some arbitrary r such that the right hand-side is defined; if there is no such r, o'_R is undefined and we call R a *dead* set of states. By padding o' with some arbitrary values on such dead states, we may extend $\mathcal{T}'$ to a non-partial deterministic $\mathfrak{C}$-SST $\mathcal{T}''$ so that $\llbracket \mathcal{T}' \rrbracket \subseteq \llbracket \mathcal{T}'' \rrbracket$. We may then consider a sdm-$\mathfrak{D}$-SST $\mathcal{T}''' = (Q', q_0, (D_q)_{q \in Q'}, \delta, i'', o'')$ such that $\llbracket \mathcal{T}' \rrbracket(w) = \llbracket \mathcal{T}'' \rrbracket(w) = \llbracket \mathcal{T}''' \rrbracket$ for $w \in \mathsf{dom}(\llbracket \mathcal{T}' \rrbracket)$. This $\mathcal{T}'''$ is almost our uniformizer; we only need to restrict the domain of its output function. This can be achieved by adding a $\mathcal{P}(Q)$ component to the state space corresponding to the set of states reached by $\mathcal{T}$ and forcing the output function to be undefined if this component contains a dead set of states. $\square$

Putting Lemma 3.47 together with Theorem 3.46 yields the desired result.

Theorem 3.48. *sdm-$\mathfrak{SR}_\&$-SSTs are subsumed by sdm-$\mathfrak{SR}$-SSTs.*

We also provide a direct self-contained proof of the following statement generalizing Theorem 3.48.

Theorem 3.49. *Let $\mathfrak{C}$ and $\mathfrak{D}$ be streaming settings such that there is a morphism of streaming settings $\mathfrak{C} \to \mathfrak{D}$, whose underlying functor is $F : \mathcal{C} \to \mathcal{D}$. Assume further that $\mathcal{D}$ carries a symmetric monoidal affine structure and has internal homsets $F(C) \multimap F(C')$ for every pair of objects $C, C' \in \mathsf{Obj}(\mathcal{C})$.*

Then, sdm-$\mathfrak{C}_\&$-SSTs are subsumed by sdm-$\mathfrak{D}$-SSTs.

Let us check that our technical development until now allows deriving Theorem 3.48 from the above result. We instantiate $\mathfrak{C} = \mathfrak{SR}$ and $\mathfrak{D} = \mathfrak{SR}_\oplus$, with the functor $\iota_\oplus : \mathcal{SR} \to \mathcal{SR}_\oplus$. We have already seen that $\mathcal{SR}_\oplus$ is a symmetric monoidal affine category. The assumption on internal homsets is exactly Lemma 3.38. The theorem then tells us that sdm-$\mathfrak{SR}_\&$-SSTs are subsumed by sdm-$\mathfrak{SR}_\oplus$-SSTs, and the latter are no more expressive than sdm-$\mathfrak{SR}$-SSTs by Corollary 3.33.

Note that, conversely, Theorem 3.48 also implies Theorem 3.46 through Lemma 3.47. Therefore, while Theorem 3.46 is a variant of the previously known determinization of copyless SSTs, we generalized it in a more abstract setting. Our main contribution is identifying the notion of internal homsets as one of the key components which make the direct determinization proof (that does not appear in [AD11], but might be part of the folklore, see e.g. [BC18, Problem 139 (p. 226)]) work. The proof itself is thus rather unsuprising, but rather involved, so we postpone it to Subsection 4.2.

3.6. **The $\oplus\&$-completion (a Dialectica-like construction).** We now consider the composition of the coproduct completion with the product completion $(\mathcal{C}_\&)_\oplus$. Unraveling the formal definition and distributing sums and products at the right spots, we define an isomorphic category $\mathcal{C}_{\oplus\&}$ which is a bit less cumbersome to manipulate in practice.

Theorem 3.50. *Given an arbitrary category $\mathcal{C}$, there is an isomorphism of categories (not just an equivalence) between $(\mathcal{C}_\&)_\oplus$ and the category $\mathcal{C}_{\oplus\&}$ defined below.*

- *The objects of $\mathcal{C}_{\oplus\&}$ are triples $(U, (X_u)_u, (C_{u,x})_{(u,x)})$ where U is a finite set, $(X_u)_{u \in U}$ is a family of finite sets and $C_{u,x}$ is a family of objects of $\mathcal{C}$ indexed by $(u, x) \in \sum_{u \in U} X_u$. We drop the first index u when it is determined by $x \in X_u$ from context and write those objects $\bigoplus_{u \in U} \bigwiththis_{x \in X_u} C_x$ for short.*

- *Its homsets are defined as*

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}} \left(\bigoplus_{u \in U} \underset{x \in X_u}{\&} C_x, \; \bigoplus_{v \in V} \underset{y \in Y_v}{\&} C_y \right) \;\; = \;\; \prod_{u \in U} \sum_{v \in V} \prod_{y \in Y_v} \sum_{x \in X_u} \mathsf{Hom}_{\mathcal{C}} \left(C_x, C_y \right)$$

- *Its identities are maps*

$$\prod_{u \in U} \; \sum_{u' \in U} \; \prod_{x' \in X_u} \; \sum_{x \in X_{u'}} \; \mathsf{Hom}_{\mathcal{C}} \left(C_x, C_{,x'} \right)$$

$$u \mapsto \left[u \;, \; \left(x \; \mapsto \; (x \;, \; \mathrm{id}_{C_x}) \right) \right]$$

- *Composition is defined as in Figure 7. The interesting steps of this computation are those involving v and y; since they are similar, let us focus on v. A map of the form*

$$\prod_{v \in V} A_v \;\; \times \;\; \sum_{v \in V} B_v \;\; \longrightarrow \;\; \sum_{v \in V} A_v \times B_v$$

$$(a_v)_{v \in V} \quad, \quad (v', \; b) \quad \mapsto \quad (v', \; (a_{v'} \;, \; b))$$

is applied. This makes the two $\mathcal{C}_{\oplus\&}$-morphisms interact (an interaction represented in Remark 3.52 below as a move in a game): the v' provided by the right one selects $a_{v'}$ among all the possibilities $(a_v)_v$ proposed by the left one.

Proof. By mechanical unfolding of the definitions. □

Remark 3.51. The reader may notice that composition in $\mathcal{C}_{\oplus\&}$ is very similar to the interpretation of cuts in Gödel's Dialectica interpretation [Göd58] and/or composition in categories of *polynomial functors* [GK13, MvG18]. This intuition can be made formal. In particular, see [Hof11] for a decomposition of a general version of the categorical Dialectica construction into free completions with *simple* sums and products. In our context, the completion with simple coproducts would be the $(-)_{\oplus\mathrm{const}}$ of Remark 3.37; conversely, a "dependent Dialectica" could be defined in the fibrational setting of [Hof11] analogously to our $(-)_{\oplus\&}$-completion by removing the simplicity restriction.

Remark 3.52. For the uninitiated, it can be helpful to compute this completion on the trivial category $\mathbb{1}$ with one object and only its identity morphism. In this case, objects consists of a pair of a finite set U together with a family $(X_u)_{u \in U}$ of finite sets that can be regarded as a two-move sequential game (with no outcome) between player $\oplus$ and $\&$: first $\oplus$ plays some $u \in U$ and then $\&$ plays some $x \in X_u$. One can then consider *simulation games* between $(U, (X_u)_u)$ (the "left hand-side") and $(V, (Y_v)_v)$ (the "right hand-side") proceeding as follows:

- first, $\&$ plays some $u \in U$ on the left
- then, $\oplus$ plays some $v \in V$ on the right
- $\&$ answers with some $y \in Y_v$ on the right
- finally $\oplus$ answers with $x \in X_u$ on the left.

	U , $(X_u)_u$	$\to$	$V, (Y_v)_v$
$\&$	u		
$\oplus$			v
$\&$			y
$\oplus$	x		

Morphisms in $\mathbb{1}_{\oplus\&}$ are $\oplus$-strategies in such games. Identities correspond to copycat strategies and composition is (a simple version) of an usual scheme in game semantics. As for $\mathcal{C}_{\oplus\&}$, one may consider that once this simulation game is played, $\oplus$ needs to provide a datum in some $\mathsf{Hom}_{\mathcal{C}} (C_x, C_y)$ which depends on the outcome of the game.

$$\mathsf{Hom}\left(\bigoplus_v \underset{y}{\&}\, C_y,\ \bigoplus_w \underset{z}{\&}\, C_z\right) \times \mathsf{Hom}\left(\bigoplus_u \underset{x}{\&}\, C_x,\ \bigoplus_v \underset{y}{\&}\, C_y\right)$$

$$\|$$

$$\prod_v \sum_w \prod_z \sum_y \mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \quad \times \quad \prod_u \sum_v \prod_y \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right)$$

$$\downarrow$$

$$\prod_u \left(\prod_v \sum_w \prod_z \sum_y \mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \ \times \ \sum_v \prod_y \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right) \right)$$

$$\downarrow$$

$$\prod_u \sum_v \left(\sum_w \prod_z \sum_y \mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \quad \times \quad \prod_y \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right) \right)$$

$$\downarrow \sim$$

$$\prod_u \sum_{v,w} \left(\prod_z \sum_y \mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \quad \times \quad \prod_y \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right) \right)$$

$$\downarrow$$

$$\prod_u \sum_{v,w} \prod_z \left(\sum_y \mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \quad \times \quad \prod_y \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right) \right)$$

$$\downarrow$$

$$\prod_u \sum_{v,w} \prod_z \sum_y \left(\mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \quad \times \quad \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right) \right)$$

$$\downarrow \sim$$

$$\prod_u \sum_{v,w} \prod_z \sum_{y,x} \left(\mathsf{Hom}_{\mathcal{C}}\left(C_y, C_z\right) \times \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_y\right) \right)$$

$$\downarrow {\scriptstyle \text{composition in } \mathcal{C}}$$

$$\prod_u \sum_{v,w} \prod_z \sum_{y,x} \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_z\right)$$

$$\downarrow {\scriptstyle \text{project away } v, y}$$

$$\prod_u \sum_w \prod_z \sum_x \mathsf{Hom}_{\mathcal{C}}\left(C_x, C_z\right)$$

$$\|$$

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigoplus_u \underset{x}{\&}\, C_x,\ \bigoplus_w \underset{z}{\&}\, C_z\right)$$

Figure 7: Composition in $\mathcal{C}_{\oplus\&}$ ($- \in -$ are omitted from indices)

We write $\iota_{\oplus\&} : \mathcal{C} \to \mathcal{C}_{\oplus\&}$ for the (full and faithful) embedding sending an object C to the one-element family $\bigoplus_1 C$. As the coproduct completion preserves limits, it means that $\mathcal{C}_{\oplus\&}$ always boasts both binary cartesian products and coproducts. Concretely, products are computed using the distributivity of products over coproducts

$$\underset{i \in I}{\&} \bigoplus_{j \in J_i} A_j \quad \cong \quad \bigoplus_{f \in \prod_{i \in I} J_i} \underset{i \in I}{\&} A_{f(i)}$$

If $\mathcal{C}$ has a symmetric monoidal structure $(\otimes, \mathbf{I})$, the lifting is computed in $\mathcal{C}_{\oplus\&}$ as

$$\bigotimes_{i \in I} \bigoplus_{u \in U_i} \underset{x \in X_u}{\&} C_x \quad = \quad \bigoplus_{f \in \prod_{i \in I} U_i} \underset{g \in \prod_{i \in I} X_{i,f(i)}}{\&} \bigotimes_{i \in I} C_{g(i)}$$

3.6.1. *The monoidal closure theorem.* Recall from the introduction that a result of central importance in this paper is the fact that the category $(\mathcal{SR}_\&)_\oplus$ is symmetric monoidal closed (Theorem 1.3). This will be a consequence of the general property below.

Theorem 3.53. *Let $(\mathcal{C}, \otimes, \mathbf{I})$ be a symmetric monoidal category. Assume that its coproduct completion $\mathcal{C}_\oplus$ admits internal homsets $\iota_\oplus(A) \multimap \iota_\oplus(B)$ for every $A, B \in \mathsf{Obj}(\mathcal{C})$. Then its Dialectica-like completion $\mathcal{C}_{\oplus\&}$ is monoidal closed.*

Indeed, since Lemma 3.38 gives us precisely the assumption on internal homsets in the above statement for $\mathcal{C} = \mathcal{SR}$, we immediately get:

Corollary 3.54 (equivalent to Theorem 1.3). *$\mathcal{SR}_{\oplus\&}$ is monoidal closed.*

Let us now sketch how the proof of Theorem 3.53 goes. To this end, assume that we have $A = \bigoplus_{u \in U} \underset{x \in X_u}{\&} \iota_{\oplus\&}(A_x)$ and $B = \bigoplus_{v \in V} \underset{y \in Y_v}{\&} \iota_{\oplus\&}(B_y)$ be objects of $\mathcal{C}_{\oplus\&}$ and assume that we have internal homsets $\iota_{\oplus\&}(A_x) \multimap \iota_{\oplus\&}(B_y)$ for every $(u, x) \in \sum_{u \in U} X_u$ and $(v, y) \in \sum_{v \in V} Y_v$. Let

$$\iota_\&^\oplus : \mathcal{C}_\oplus \to \mathcal{C}_{\oplus\&} \quad \text{such that} \quad \iota_{\oplus\&} = \iota_\&^\oplus \circ \iota_\oplus$$

be the full and faithful embedding be obtained by applying the universal property of $\mathcal{C}_\oplus$ to the coproduct-preserving functor $\iota_{\oplus\&}$. The linear arrow $A \multimap B$ can then be defined as follows

$$A \multimap B \quad = \quad \underset{u \in U}{\&} \bigoplus_{v \in V} \underset{y \in Y_v}{\&} \bigoplus_{x \in X_u} \iota_\&^\oplus(\iota_\oplus(A_x) \multimap \iota_\oplus(B_y))$$

It is clear that this is meant to mimic the definition of $\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(-, -)$ by substituting external hom, products and coproducts with internal ones, as is customary with Dialectica. To show that $B \multimap -$ is right adjoint to $- \otimes B$, one may directly establish that we have an isomorphism of homsets

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(A \otimes B, C) \cong \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(A, B \multimap C)$$

natural in A using the recipe of Figure 8. Unmarked steps follow from definitions or standard abstract nonsense (characterization of internal (co)products in terms of external ones). The two ($\dagger$) steps involve instances of the isomorphism

$$\sum_{i \in I} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}} \left(\underset{x \in X}{\&} \iota_{\oplus\&}(Z_x), \, D_i \right) \quad \cong \quad \mathsf{Hom}_{\mathcal{C}_{\oplus\&}} \left(\underset{x \in X}{\&} \iota_{\oplus\&}(Z_x), \, \bigoplus_{i \in I} D_i \right)$$

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(A \otimes B, C)$$

$$= \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\left[\bigoplus_{w \in W} \bigwedge_{z \in Z_w} \iota_{\oplus\&}(A_z)\right] \otimes \left[\bigoplus_{u \in U} \bigwedge_{x \in X_u} \iota_{\oplus\&}(B_x)\right], \bigoplus_{v \in V} \bigwedge_{y \in Y_v} \iota_{\oplus\&}(C_y)\right)$$

$$= \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigoplus_{(w,u) \in W \times U} \bigwedge_{(z,x) \in Z_w \times X_u} \iota_{\oplus\&}(A_z \otimes B_x), \bigoplus_{v \in V} \bigwedge_{y \in Y_v} \iota_{\oplus\&}(C_y)\right)$$

$$\cong \prod_{w \in W} \prod_{u \in U} \sum_{v \in V} \prod_{y \in Y_v} \sum_{z \in Z_w} \sum_{x \in X_u} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota_{\oplus\&}(A_z) \otimes \iota_{\oplus\&}(B_x), \iota_{\oplus\&}(C_y)\right)$$

$$\cong \prod_{w \in W} \prod_{u \in U} \sum_{v \in V} \prod_{y \in Y_v} \sum_{z \in Z_w} \sum_{x \in X_u} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota_{\oplus\&}(A_z), \iota_{\&}^{\oplus}(\iota_{\oplus\&}(B_x) \multimap \iota_{\oplus\&}(C_y))\right)$$

$$\cong \prod_{w \in W} \prod_{u \in U} \sum_{v \in V} \prod_{y \in Y_v} \sum_{z \in Z_w} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota_{\oplus\&}(A_z), \bigoplus_{x \in X_u} \iota_{\&}^{\oplus}(\iota_{\oplus\&}(B_x) \multimap \iota_{\oplus\&}(C_y))\right) \quad (\dagger)$$

$$\cong \prod_{w \in W} \prod_{u \in U} \sum_{v \in V} \prod_{y \in Y_v} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigwedge_{z \in Z_w} \iota_{\oplus\&}(A_z), \bigoplus_{x \in X_u} \iota_{\&}^{\oplus}(\iota_{\oplus\&}(B_x) \multimap \iota_{\oplus\&}(C_y))\right) \quad (\heartsuit)$$

$$\cong \prod_{w \in W} \prod_{u \in U} \sum_{v \in V} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigwedge_{z \in Z_w} \iota_{\oplus\&}(A_z), \bigwedge_{y \in Y_v} \bigoplus_{x \in X_u} \iota_{\&}^{\oplus}(\iota_{\oplus\&}(B_x) \multimap \iota_{\oplus\&}(C_y))\right)$$

$$\cong \prod_{w \in W} \prod_{u \in U} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigwedge_{z \in Z_w} \iota_{\oplus\&}(A_z), \bigoplus_{v \in V} \bigwedge_{y \in Y_v} \bigoplus_{x \in X_u} \iota_{\&}^{\oplus}(\iota_{\oplus\&}(B_x) \multimap \iota_{\oplus\&}(C_y))\right) \quad (\dagger)$$

$$\cong \prod_{w \in W} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigwedge_{z \in Z_w} \iota_{\oplus\&}(A_z), B \multimap C\right)$$

$$\cong \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(A, B \multimap C)$$

Figure 8: Monoidal closure of $\mathcal{C}_{\oplus\&}$ (Theorem 3.53).

which holds for any family $(Z_x)_{x \in X}$ of objects of $\mathcal{C}$ and family $(D_i)_{i \in I}$ of objects of $\mathcal{C}_{\oplus\&}$. This is a formal computation corresponding to the tail end of Proposition 3.28, which applies because of the equivalence $\mathcal{C}_{\oplus\&} \cong (\mathcal{C}_{\&})_{\oplus}$.

The most subtle step, labelled with $(\heartsuit)$, necessitates a more detailed computation that relies crucially on the fact that $\iota_{\&}^{\oplus}(\iota_{\oplus}(A_x) \multimap \iota_{\oplus}(B_y))$ is &-free. This can be regarded as a generic isomorphism for arbitrary families of objects $(E_u)_{u \in U}$ and $(Z_x)_{x \in X}$ of $\mathcal{C}$

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigwedge_{x \in X} \iota_{\oplus\&}(Z_x), \bigoplus_{u \in U} \iota_{\oplus\&}(E_u)\right) \cong \sum_{x \in X} \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(Z_x, \bigoplus_{u \in U} \iota_{\oplus\&}(E_u)\right)$$

which is easy to compute explicitly.

In addition to checking that there is a chain of isomorphisms as per Figure 8, it should also be checked that they are natural in A. This requires identifying precisely what presheaves over $\mathcal{C}_{\oplus\&}$ are involved at every step and arguing that the maps are indeed natural. This

is not necessarily so straightforward, so we offer a more detailed proof of Theorem 3.53 in Appendix D.

3.7. **Proof of the main result on strings.** We can now give the proof of Theorem 1.1, which can be summarized as the equality

$$\lambda\ell^{\oplus\&}\text{-definable} \quad = \quad \mathfrak{SR}\text{-SSTs}$$

thanks to Fact 3.14. We start with the consequences of our syntactic analysis

$$\lambda\ell^{\oplus\&}\text{-definable} \quad \underset{\substack{\uparrow \\ \text{Lemma 3.23}}}{=} \quad \text{single-state } \mathfrak{L}\text{-SSTs} \quad \underset{\substack{\uparrow \\ \text{Lemma 3.36}}}{=} \quad \mathfrak{L}\text{-SSTs}$$

reducing our goal to

$$\mathfrak{L}\text{-SSTs} \quad = \quad \mathfrak{SR}\text{-SSTs}$$

For the above equality, the right-to-left inclusion is simpler than its converse: the existence of a morphism of streaming settings from $\mathfrak{SR}$ to $\mathfrak{L}$ (Lemma 3.25) entails that $\mathfrak{L}$-SSTs subsume $\mathfrak{SR}$-SSTs (by Lemma 3.9). (Were we only interested only proving that regular functions are $\lambda\ell^{\oplus\&}$-definable, our setting would be a complete overkill.)

On the other hand, the other direction mobilizes almost the whole development. First, our semantic evaluation argument combines Lemma 3.26 with Corollary 3.54 to get

$$\mathfrak{L}\text{-SSTs} \quad \subseteq \quad \mathfrak{SR}_{\oplus\&}\text{-SSTs}$$

We then finish proving Theorem 1.1 with automata-theoretic considerations:

$$\mathfrak{SR}_{\oplus\&}\text{-SSTs} \quad \underset{\substack{\uparrow \\ \text{Lemma 3.35}}}{=} \quad \text{sdm-}\mathfrak{SR}_{\&}\text{-SSTs} \quad \underset{\substack{\uparrow \\ \text{Theorem 3.48}}}{=} \quad \text{sdm-}\mathfrak{SR}\text{-SSTs}$$

$$\text{sdm-}\mathfrak{SR}\text{-SSTs} \quad \underset{\substack{\uparrow \\ \text{Corollary 3.33}}}{=} \quad \mathfrak{SR}_{\oplus}\text{-SSTs} \quad \underset{\substack{\uparrow \\ \text{Lemma 3.36}}}{=} \quad \mathfrak{SR}\text{-SSTs}$$

4. Some transducer-theoretic applications of $\mathfrak{C}$-SSTs and internal homsets

This section is devoted to showing that the notion of monoidal closure can be used to give a satisfying self-contained description of two important transformations of usual streaming string transducers, both of which are not entirely straightforward: the composition of two copyless SSTs and the uniformization of non-deterministic copyless SSTs.

We take advantage of our abstract setting to give proofs for categorical generalizations of those two statements, similarly in spirit to the generalized minimization argument found in [CP20]. The specialized theorem then follows easily from previous theorems in our developments having to do with monoidal closure, especially Theorem 3.53 and Lemma 3.38.

The ideas behind our arguments are not new; our main goal is to vindicate the view that the notion of internal homset is the key to showing those results, even if it does not appear explicitly in previous arguments.

4.1. **On closure under precomposition by regular functions.** First, let us recall right off the bat that the closure under composition of functions definable by copyless streaming string transducers follows from our main result on strings (Theorem 1.1) together with basic considerations on typed λ-calculi (see e.g. [NP20, Lemma 2.8] that applies *mutatis mutandis* to the $\lambda\ell^{\oplus\&}$-calculus) that entail the closure under composition of $\lambda\ell^{\oplus\&}$-definable functions. However, note that Theorem 1.1 relies on the syntactic analysis of Lemma 3.24, which is arguably a non-trivial result about the $\lambda\ell^{\oplus\&}$-calculus (as demonstrated by the size of Appendices B and C, which are both necessary to prove it).

The argument we give in this section circumvents that difficulty and does not appeal to Theorem 1.1 nor to Lemma 3.24. That said, it still shares some (non-syntactic) ingredients with our proof of Theorem 1.1, namely:
- the monoidal closure and quasi-affineness (see below) of $\mathcal{SR}_{\oplus\&}$;
- the fact that $\mathfrak{SR}_{\oplus\&}$-SSTs are no more expressive than $\mathfrak{SR}$-SSTs.

These results still require substantial developments – indeed, this composition property is quite non-trivial as mentioned in the introduction – but bypass the need of mentioning the $\lambda\ell^{\oplus\&}$-calculus.

Beyond this simplification, the main advantage of our approach here is that we get a more general theorem, that applies to many streaming settings; in particular, the final output does not have to be a string. Without further ado, let us state it.

Theorem 4.1. *Let $\mathfrak{C}$ be a string streaming setting with output set X. Suppose that the underlying category $\mathcal{C}$ is symmetric monoidal closed and quasi-affine. Furthermore, let us assume that $\mathbb{T}_{\mathfrak{C}}$ is equal to the monoidal unit $\mathbf{I}$.*

Then for any $f : \Gamma^* \to X$ computed by some $\mathfrak{C}$-SST, and any regular $g : \Sigma^* \to \Gamma^*$, the function $f \circ g : \Sigma^* \to X$ is computed by some (stateful) $\mathfrak{C}$-SST. In other words, the class of functions defined by $\mathfrak{C}$-SSTs is closed under precomposition by regular functions.

Before proving the above theorem, let us check that it entails known preservation and composition properties.

Corollary 4.2. *Let $L \subseteq \Gamma^*$ be a regular language and $g : \Sigma^* \to \Gamma^*$ be a regular function. Then the language $g^{-1}(L) \subseteq \Sigma^*$ is regular.*

Proof. That L is regular is equivalent to its indicator function $\chi_L : \Gamma^* \to \{0,1\}$ being computed by some single-state $\mathfrak{Finset}_2$-SST, see Example 3.5. The underlying category of finite sets is *cartesian closed*, and the monoidal structure given by a cartesian product is automatically symmetric and affine. According to Theorem 4.1, $\chi_L \circ g$ can therefore be computed by some $\mathfrak{Finset}_2$-SST. Observing that the category of finite sets has coproducts, and applying Lemma 3.36, we even have a *single-state* $\mathfrak{Finset}_2$-SST for $\chi_L \circ g$. Finally, the latter is none other than the indicator function of $g^{-1}(L)$. $\square$

Corollary 4.3. *Let $f : \Gamma^* \to \Delta^*$ and $g : \Sigma^* \to \Gamma^*$ be regular functions. Then $f \circ g$ is also a regular function.*

Proof. This is just the application of Theorem 4.1 to $\mathfrak{SR}_{\oplus\&}$-SSTs – indeed, we saw at the very end of Section 3 that the functions computed by $\mathfrak{SR}_{\oplus\&}$-SSTs are exactly the regular functions. By Theorem 1.3 / Corollary 3.54, the underlying category $\mathcal{SR}_{\oplus\&}$ is symmetric monoidal closed. Finally, $\mathcal{SR}_{\oplus\&}$ is quasi-affine since it has all cartesian products by construction. $\square$

We now come to the proof of this generalized preservation theorem.

Proof of Theorem 4.1. We give below a proof assuming that f is defined by some *single-state* $\mathfrak{C}$-SST (but beware: the $\mathfrak{C}$-SST computing $f \circ g$ will still be stateful!). The general case can be applied by considering the streaming setting $\mathfrak{C}_{\oplus \text{const}}$-SST, as was briefly mentioned in Remark 3.37, and using the fact that single-state $\mathfrak{C}_{\oplus \text{const}}$-SSTs, stateful $\mathfrak{C}_{\oplus \text{const}}$-SSTs and stateful $\mathfrak{C}$-SSTs are equally expressive. We leave it to the reader to check that the symmetric monoidal structure and the cartesian products in $\mathcal{C}$ can be lifted to $\mathcal{C}_{\oplus \text{const}}$, making this generalization possible.

Therefore, we may assume without loss of generality that f is computed by a single-state $\mathfrak{C}$-SST $T_f = (\{\bullet\}, A_f, \delta_f, i_f, o_f)$ where A_f is an object of $\mathcal{C}$ and

$$\delta_f : \Gamma \to \mathsf{Hom}_{\mathcal{C}}\left(A_f, A_f\right) \qquad i_f \in \mathsf{Hom}_{\mathcal{C}}\left(\mathbb{T}, A_f\right) \qquad o_f \in \mathsf{Hom}_{\mathcal{C}}\left(A_f, \perp\!\!\!\perp\right)$$

Let $T_g = (Q, q_0, R_g, \delta_g, i_g, o_g)$ be an usual copyless SST (i.e., a $\mathcal{SR}$-SST) computing the regular function g, where Q and R_g are finite sets and

$$q_0 \in Q \qquad \delta_g : \Sigma \times Q \to Q \times [R_g \to_{\mathcal{SR}} R_g]$$

$$i_g \in [\varnothing \to_{\mathcal{SR}} R_g] \qquad o_g : Q \to [R_g \to_{\mathcal{SR}} \{\bullet\}]$$

We will write $A = A_f$ and $R = R_g$ for short.

We want to build from this data a $\mathfrak{C}$-SST T defining $f \circ g$. Since $\mathcal{C}$ is quasi-affine and symmetric monoidal closed, we can apply Corollary 3.19 to the object $A \multimap A$ and to a family of morphisms $(\widetilde{\delta_f}(c))_{c \in \Gamma} \in \mathsf{Hom}_{\mathcal{C}}\left(A \multimap A, \ A \multimap A\right)^{\Gamma}$ that will be defined later. This gives us a functor $F_{\delta_f} : \mathcal{SR} \to \mathcal{C}$, enjoying various properties that will be progressively recalled, which is at the heart of our construction.

The set of states of our new $\mathfrak{C}$-SST T is Q, with initial state q_0, and its memory object is $F_{\delta_f}(R)$. The initialization morphism is defined as $i = F_{\delta_f}(i_g) \in \mathsf{Hom}_{\mathcal{C}}\left(\mathbf{I}, F_{\delta_f}(R)\right)$ – we use the assumption $\mathbb{T} = \mathbf{I}$, and the fact that $F_{\delta_f}(\varnothing) = \mathbf{I}$ (by Corollary 3.19). The transition function is

$$\delta : \quad \Sigma \times Q \quad \longrightarrow \quad Q \quad \times \quad \mathsf{Hom}_{\mathcal{C}}\left(F_{\delta_f}(R), F_{\delta_f}(R)\right)$$

$$c \ , \ q \quad \mapsto \quad \pi_1(\delta_g(c,q)) \ , \qquad F_{\delta_f}(\pi_2(\delta_g(c,q)))$$

Finally, using $j_f : F_{\delta_f}(\{\bullet\}) \to A$ to be defined later, we take as our new output function

$$o : q \in Q \mapsto o_f \circ j_f \circ F_{\delta_f}(o_g(q)) \in \mathsf{Hom}_{\mathcal{C}}\left(F_{\delta_f}(R), \perp\!\!\!\perp\right)$$

Let us now sketch the verification that this defines the intended function $f \circ g : \Sigma^* \to X$. In the process, we will fill the missing definitions to make everything work out.

Let $w = w_1 \ldots w_n \in \Sigma^*$ be an input string. The sequence $q_0, \ldots, q_n \in Q$ of states visited by both T_g and T when fed this input is obtained by $q_{i+1} = \pi_1(\delta_g(w_i, q_i))$ from the initial q_0. By definition of the output of T_g, we have:

$$\widehat{g(w)} \quad = \quad o_g(q_n) \circ \pi_2(\delta_g(w_n, q_{n-1})) \circ \cdots \circ \pi_2(\delta_g(w_1, q_0)) \circ i_g$$

where $\widehat{g(w)} \in [\varnothing \to_{\mathcal{SR}(\Gamma)} \{\bullet\}]$ corresponds to $g(w) \in \Gamma^*$ (cf. Theorem 3.17) and the '$\circ$' denotes a composition of register transitions (i.e. of $\mathcal{SR}$-morphisms). Similarly, the output $T(w)$ of the $\mathfrak{C}$-SST T that we built on the input w is defined as

$$T(w) \quad = \quad (\!|o(q_n) \circ F_{\delta_f}(\pi_2(\delta_g(w_n, q_{n-1}))) \circ \cdots \circ F_{\delta_f}(\pi_2(\delta_g(w_1, q_0))) \circ i|\!)$$

which, by unfolding the definitions of o and i, applying the functoriality of F_{δ_f} and comparing with the previous equality, one can simplify into

$$T(w) \quad = \quad (\!|o_f \circ j_f \circ F_{\delta_f}\left(\widehat{g(w)}\right)|\!)$$

It is now time to define $j_f \in \mathsf{Hom}_{\mathcal{C}} \left(F_{\delta_f}(\{\bullet\}), A \right)$. To do so, let us first introduce

$$\mathsf{appto}(\varphi) \ : \ B \multimap C \xrightarrow{\sim} (B \multimap C) \otimes \mathbf{I} \xrightarrow{\mathsf{id} \otimes \varphi} (B \multimap C) \otimes B \xrightarrow{\mathrm{ev}} C$$

where the last arrow is the evaluation map $\mathrm{ev}_{B,C}$, for any $B, C \in \mathsf{Obj}(\mathcal{C})$ and $\varphi : \mathbf{I} \to B$. An useful property, whose verification we leave to the reader, is

$$\mathsf{appto}(\varphi) \circ \Lambda'(\psi) \quad = \quad \psi \circ \varphi \qquad \text{for any } \psi : B \to C$$

where $\Lambda' : \mathsf{Hom}_{\mathcal{C}} (B, C) \xrightarrow{\sim} \mathsf{Hom}_{\mathcal{C}} (\mathbf{I}, B \multimap C)$ is defined in Proposition 2.25. We then take

$$j_f \ : \ F_{\delta_f}(\{\bullet\}) \xrightarrow{\pi_1} (A \multimap A) \multimap (A \multimap A) \xrightarrow{\mathsf{appto}(\Lambda'(\mathsf{id}_A))} A \multimap A \xrightarrow{\mathsf{appto}(i_f)} A$$

where π_1 is the left projection from $F_{\delta_f}(\{\bullet\}) = ((A \multimap A) \multimap (A \multimap A)) \,\&\, \mathbf{I}$ (this equality is guaranteed by Corollary 3.19) and i_f is the initialization morphism of T_f. Using the equation

$$F_{\delta_f}\left(\widehat{g(w)} \right) \quad = \quad \left\langle \Lambda'\left(\widetilde{\delta_f}(g(w)_1) \circ \cdots \circ \widetilde{\delta_f}(g(w)_m) \right), \ \mathsf{id}_{\mathbf{I}} \right\rangle \quad \text{where } m = |g(w)|$$

coming from Corollary 3.19, we then have

$$j_f \circ F_{\delta_f}\left(\widehat{g(w)} \right) \quad = \quad \mathsf{appto}(i_f) \circ \widetilde{\delta_f}(g(w)_1) \circ \cdots \circ \widetilde{\delta_f}(g(w)_m) \circ \Lambda'(\mathsf{id}_A)$$

Next, we define $\widetilde{\delta_f}(c) = \Lambda(\mathrm{ev}_{A,A} \circ (\mathsf{id}_{A \multimap A} \otimes \delta_f(c))) \in \mathsf{Hom}_{\mathcal{C}} (A \multimap A, \ A \multimap A)$ for $c \in \Gamma$. In other words, $\widetilde{\delta_f}(c)$ is the curryfication of

$$(A \multimap A) \otimes A \xrightarrow{\mathsf{id} \otimes \delta_f(c)} (A \multimap A) \otimes A \xrightarrow{\mathrm{ev}} A$$

One can then check that $\widetilde{\delta_f}(c) \circ \Lambda'(\psi) = \Lambda'(\psi \circ (\delta_f(c)))$ for any $\psi : A \to A$. Putting everything together, we finally have

$$T(w) = \Big(o_f \circ \mathsf{appto}(i_f) \circ \widetilde{\delta_f}(g(w)_1) \circ \cdots \circ \widetilde{\delta_f}(g(w)_m) \circ \Lambda'(\mathsf{id}_A) \Big)$$

$$= \Big(o_f \circ \mathsf{appto}(i_f) \circ \Lambda'(\delta_f(g(w)_m) \circ \cdots \circ \delta_f(g(w)_1)) \Big)$$

$$= \Big(o_f \circ \delta_f(g(w)_m) \circ \cdots \circ \delta_f(g(w)_1) \circ i_f \Big)$$

and this final expression is precisely the definition of the output of T_f on $g(w)$. Since T_f computes f, we end up with $T(w) = f(g(w))$, as we wanted. $\qquad\square$

To conclude this section, let us note that an analogous result for precomposition by regular tree functions can be shown by leveraging the results of Section 5; we leave it as an exercise. An important subtlety: since the presence of the additive conjunction is important to compute regular tree functions (as we stressed in the introduction), one must consider tree streaming settings whose underlying categories *have finite cartesian products* (which entails quasi-affineness).

4.2. Uniformization through monoidal closure.

We recall below the categorical uniformization theorem that we used in Section 3.5 and provide its proof.

Theorem 3.49. *Let $\mathfrak{C}$ and $\mathfrak{D}$ be streaming settings such that there is a morphism of streaming settings $\mathfrak{C} \to \mathfrak{D}$, whose underlying functor is $F : \mathcal{C} \to \mathcal{D}$. Assume further that $\mathcal{D}$ carries a symmetric monoidal affine structure and has internal homsets $F(C) \multimap F(C')$ for every pair of objects $C, C' \in \mathsf{Obj}(\mathcal{C})$.*

Then, sdm-$\mathfrak{C}_{\&}$-SSTs are subsumed by sdm-$\mathfrak{D}$-SSTs.

Recall that according to Lemma 3.47, the conclusion amounts to saying that non-deterministic sdm-$\mathfrak{C}$-SSTs are uniformizable by partial sdm-$\mathfrak{D}$-SSTs.

Proof. We do not prove the statement in excruciating details, but provide key formal definitions so that a reader familiar with a modicum of automata theory and category theory should be able to reconstitute a fully formal argument with ease. Let us stress once again that all of the combinatorics may be regarded as adaptation of known arguments.

The argument is based on a notion of *transformation forest*, a name that we borrow from [BC18, Chapter 13] for an extremely similar concept[22]. This gadget is also reminiscent of trees used in determinization procedures like the Muller-Schupp construction for automata over ω-words [MS95] (another exposition can be found in [BC18, Chapter 1]), and of the sharing techniques used in the original paper on SSTs [AČ10, §5.2]. In determinization procedures, this constitutes an elaboration of powerset constructions recalling not only reachable states, but also crucial information on *how* those states are reached. Here, the vertices v of such forests will be labelled by objects C_v of $\mathcal{C}$ and each edge (u, v) will be correspond to a "register containing a value of type $F(C_u) \multimap F(C_v)$".

We decompose this proof sketch in three parts: first, we introduce transformation forests, their semantic interpretation as families of maps in $\mathcal{D}$; we explain how they may be composed and that maps in $\mathcal{C}_{\&}$ may be regarded as depth-1 transformation forests. Then, we explain how to reduce the size of transformation forests in a sound way (this is the crucial part ensuring that the resulting sdm-$\mathfrak{D}$-SSTs will have finitely many states). Finally, we explain how to put all of this together to uniformize sdm-$\mathfrak{C}$-SSTs.

4.2.1. *Transformation forests and their semantics.* A transformation forest is defined as a tuple $\mathcal{F} = (V, E, O, (C_v)_{v \in V})$ where
- V is a non-empty finite set of vertices
- $E \subseteq V^2$ is a set of directed edges, pointing from parents to children
- O is a non-empty subset of V which we cal the set of *output nodes*
- every C_v is an object of $\mathcal{C}$

When a transformation forest $\mathcal{F}$ is fixed, we call $I_{\mathcal{F}}$ its set of roots (which we may sometimes *input nodes*; we drop the subscript when there is no ambiguity). Given a transformation forest $\mathcal{F} = (V, E, (v_o)_{o \in O}, (C_v)_{v \in V})$, we assign the following object of $\mathcal{D}$:

$$\mathrm{Ty}(\mathcal{F}) \quad = \quad \bigotimes_{(u,v) \in E} F(C_u) \multimap F(C_v)$$

An example of a transformation forest $\mathcal{F}$ and a computation of its type $\mathrm{Ty}(\mathcal{F})$ is pictured in Figure 9.

---

[22]There are two formal differences between our notions, which are not very big but worth mentioning for readers of [BC18]. First, edges of a transformation forest are intended to be associated with (elements of) a monoid, while ours should be associated with ("elements of") internal homsets $F(C_u) \multimap F(C_v)$. Were we trying to prove $\mathfrak{D}$-uniformization for $\mathfrak{C}$-SSTs, we would have necessarily $C_u = C_v$ and the aforementioned object would have a monoid structure internal to $\mathcal{D}$, so this distinction is more an artefact of our settings rather than an essential one. Second, what [BC18] calls transformation forests refers to a class of forest with "no junk", such as dangling leaves not referring to an intended output or spurious internal nodes, while we allow those in an initial definition; we add the adjective "normalized" for those containing "no junk" as we shall see later, so this is merely a terminological detail.

$$F(T) \multimap F(C_0) \; \bigotimes \; \begin{bmatrix} F(S) \multimap F(U) \\ \otimes \\ F(U) \multimap F(C_1) \\ \otimes \\ F(U) \multimap F(V) \; \otimes \; F(V) \multimap F(C_2) \; \otimes \; V \multimap F(C_3) \\ \otimes \\ F(U) \multimap F(C_4) \end{bmatrix}$$
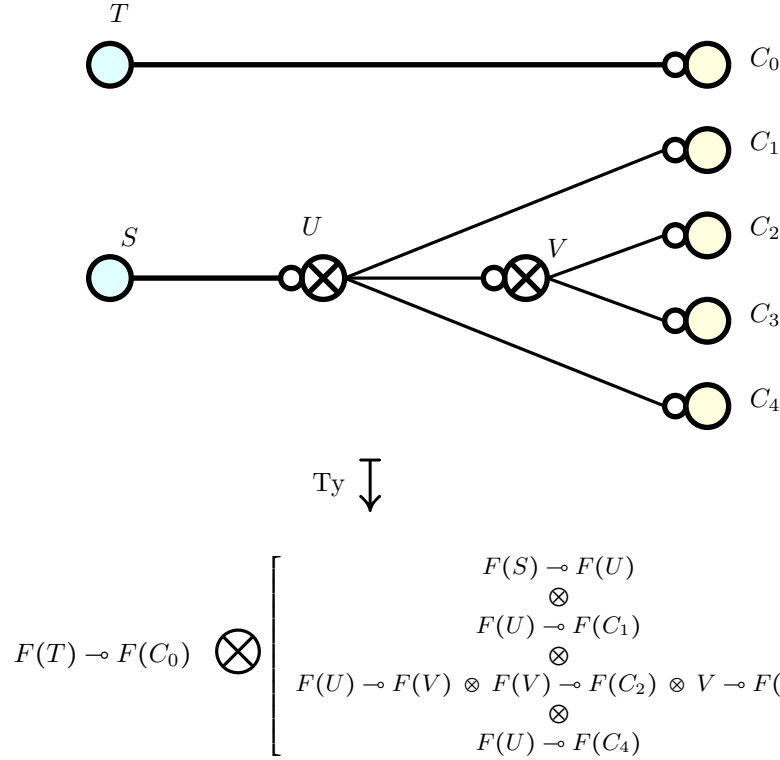
Figure 9: A transformation forest $\mathcal{F} : T \,\&\, S \to \displaystyle\bigwedge_{i=0}^{4} C_i$ $(S, T, U, V, C_0, \dots, C_4 \in \mathsf{Obj}(\mathcal{C}))$.

To guide the intuition, one may note that there is an embedding[23] of a set of suitable labellings of the forest $\mathcal{F}$ into $\mathsf{Hom}_{\mathcal{D}}(\top, \mathrm{Ty}(\mathcal{F}))$.

$$\prod_{(u,v)\in E} \mathsf{Hom}_{\mathcal{D}}\left(F(C_u), F(C_v)\right) \quad\longrightarrow\quad \mathsf{Hom}_{\mathcal{D}}\left(\top, \mathrm{Ty}(\mathcal{F})\right)$$

We will now call abusively the input of $\mathcal{F}$ the object $A = \bigwedge_{i\in I} C_i$ and the output $B = \bigwedge_{o\in O} C_o$; we write $\mathcal{F} : A \to B$ in the sequel. The justification for this notation is that there is a family of maps

$$[\![\mathcal{F}]\!] \; \in \; \prod_{o\in O} \sum_{i\in I} \mathsf{Hom}_{\mathcal{D}}\left(\mathrm{Ty}(\mathcal{F}), F(C_i) \multimap F(C_o)\right)$$

obtained by internalizing the composition of morphisms along branches of $\mathcal{F}$, which we call the *semantics of $\mathcal{F}$*.

We also note that this allow to interpret arbitrary $\mathcal{C}_{\&}$ morphisms: such a morphism $f : A \to B$ can be mapped to a pair $(\mathrm{Grph}(f), \widehat{f})$ consisting of

- a depth-1 transformation forest $\mathrm{Grph}(f) : \bigwedge_{i\in I} C_i \to \bigwedge_{o\in O} C_o$ (an example is depicted in Figure 10)

---

[23]This becomes an isomorphism when $\mathsf{Hom}_{\mathcal{D}}(\top, A \otimes B) \cong \mathsf{Hom}_{\mathcal{D}}(\top, A) \times \mathsf{Hom}_{\mathcal{D}}(\top, B)$; this is the case for our intended application $\mathcal{D} = \mathcal{SR}_{\oplus}$.

Figure 10: A depth-1 transformation forest.

- a morphism $\widehat{f} \in \mathsf{Hom}_{\mathcal{D}}\left(\top, \mathrm{Ty}(\mathcal{F})\right)$
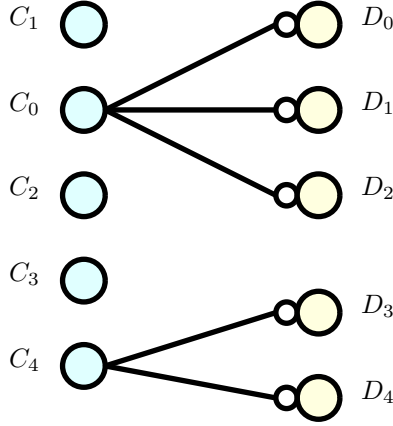
so that, if $f = (i_o, \alpha_o)_{o \in O}$, we have $[\![\mathrm{Grph}(f)]\!]_o \circ \widehat{f} = (i_o, \Lambda(\alpha_o \circ \rho^{-1}))$.

Given two forests $\mathcal{F} : A \to B$ and $\mathcal{F}' : B \to C$, there is a composition $\mathcal{F}' \circ \mathcal{F}$ obtained by gluing the input nodes of $\mathcal{F}'$ along the output nodes of $\mathcal{F}$. A crucial point is that the semantics of the composite $[\![-]\!]$ may be computed as follows

$$[\![\mathcal{F}' \circ \mathcal{F}]\!](o) \;=\; \left([\![\mathcal{F}]\!]_1([\![\mathcal{F}']\!]_1(o')), [\![\mathcal{F}']\!]_2(o') \circ [\![\mathcal{F}]\!]_2([\![\mathcal{F}']\!]_1)\right)$$

4.2.2. *Reducing transformation forests.* We now introduce two elementary transformations $\mathcal{F} \mapsto \mathcal{F}'$ over transformation forests, together with associated morphisms $\mathrm{Ty}(\mathcal{F}) \to \mathrm{Ty}(\mathcal{F}')$:

- **Pruning**: if $v \in V_{\mathcal{F}}$ is a leaf which is not an output node in the forest $\mathcal{F} : A \to B$, call $\mathrm{prune}(\mathcal{F}, v) : A \to B$ the forest obtained by removing v and adjacent edges from $\mathcal{F}$.

  This induces a canonical map

  $$prune(v) : \mathrm{Ty}(\mathcal{F}) \to \mathrm{Ty}(\mathrm{prune}(\mathcal{F}, v))$$

  by using the weakening maps on the components corresponding to the deleted edges.

- **Contraction**: If there is a vertex v with a unique child c and a (unique) parent p in the forest $\mathcal{F} : A \to B$, call $\mathrm{contract}(\mathcal{F}, v) : A \to B$ the forest obtained by replacing the edges (p, v) and (v, c) with a single edge (p, c) and removing v.

  There is a canonical map

  $$contract(v) : \mathrm{Ty}(\mathcal{F}) \to \mathrm{Ty}(\mathrm{contract}(\mathcal{F}, v))$$

  induced by the internal composition map

  $$\mathsf{Hom}_{\mathcal{D}}\left([F(C_p) \multimap F(C_v)] \otimes [F(C_v) \multimap F(C_c)], F(C_p) \multimap F(C_c)\right)$$

The auxiliary maps *prune* and *contract* operations are compatible with the semantic map $[\![\cdot]\!]$ in the sense that for every $o \in O$ and suitable vertices u, v of $\mathcal{F}$, we have

$$i = \pi_1([\![\mathcal{F}]\!](o)) = \pi_1([\![\mathrm{prune}(\mathcal{F}, u)]\!](o)) = \pi_1([\![\mathrm{contract}(\mathcal{F}, v)]\!](o))$$

and the following diagrams commute in $\mathcal{D}$

$$\begin{array}{ccc}
\mathrm{Ty}(\mathcal{F}) & \xrightarrow{\;[\![\mathcal{F}]\!](o)\;} & F(C_i) \multimap F(C_o) \\
{\scriptstyle prune(v)}\downarrow & {\scriptstyle [\![prune(\mathcal{F},v)]\!](o)} \nearrow & \\
\mathrm{Ty}(prune(\mathcal{F},v)) & &
\end{array}
\qquad
\begin{array}{ccc}
\mathrm{Ty}(\mathcal{F}) & \xrightarrow{\;[\![\mathcal{F}]\!](o)\;} & F(C_i) \multimap F(C_o) \\
{\scriptstyle contract(v)}\downarrow & {\scriptstyle [\![contract(\mathcal{F},v)]\!](o)} \nearrow & \\
\mathrm{Ty}(contract(\mathcal{F},v)) & &
\end{array}$$

Consider the rewrite system $\rightsquigarrow$ over forests $\mathcal{F} : A \to B$ induced by those two operations, and call $\rightsquigarrow^=$ its reflexive closure and $\rightsquigarrow^*$ the transitive closure of $\rightsquigarrow^=$. It can be shown that the reflexive closure $\rightsquigarrow^=$ satisfies the *diamond lemma*, i.e., that for every $\mathcal{F}, \mathcal{F}'$ and $\mathcal{F}''$ such that $\mathcal{F} \rightsquigarrow^= \mathcal{F}'$ and $\mathcal{F} \rightsquigarrow^= \mathcal{F}''$, there exists $\mathcal{F}'''$ such that $\mathcal{F}' \rightsquigarrow^= \mathcal{F}'''$ and $\mathcal{F}'' \rightsquigarrow^= \mathcal{F}'''$. This ensures that $\rightsquigarrow$ is confluent. Furthermore, given a rewrite $\mathcal{F} \rightsquigarrow^* \mathcal{F}'$, there is a map $\mathrm{Ty}(\mathcal{F}) \to \mathrm{Ty}(\mathcal{F}')$ obtained by composing maps $prune(v)$ and $contract(v)$ (and identities for trivial rewrites $\mathcal{F} \rightsquigarrow^* \mathcal{F}$). It can be shown that this map does *not* depend on the rewrite path. This is done first by arguing that if we have a rewrite square for $\rightsquigarrow^=$, the associated diagram in $\mathcal{D}$ is commutative, and then proceed by induction over the rewrite paths using the diamond lemma.

$$\begin{array}{ccc}
\mathcal{F} & \rightsquigarrow & \mathcal{F}' \\
\wr & & \wr \\
\mathcal{F}'' & \rightsquigarrow & \mathcal{F}'''
\end{array}
\qquad \longmapsto \qquad
\begin{array}{ccc}
\mathrm{Ty}(\mathcal{F}) & \longrightarrow & \mathrm{Ty}(\mathcal{F}') \\
\downarrow & & \downarrow \\
\mathrm{Ty}(\mathcal{F}'') & \longrightarrow & \mathrm{Ty}(\mathcal{F}''')
\end{array}$$

Defining the size of a forest as its number of vertices, it is clear that $prune(\mathcal{F}, u)$ and $contract(\mathcal{F}, v)$ have size strictly less than $\mathcal{F}$, so the rewrite system is also terminating. With confluence, it means that for every forest $\mathcal{F} : A \to B$, there is a unique forest $\mathrm{reduce}(\mathcal{F}) : A \to B$ such that $\mathcal{F} \rightsquigarrow^* \mathrm{reduce}(\mathcal{F})$ and there is no $\mathcal{F}'$ such that $\mathrm{reduce}(\mathcal{F}) \rightsquigarrow \mathcal{F}'$. We call $\mathrm{reduce}(\mathcal{F})$ the *normal form* of $\mathcal{F}$; a forest $\mathcal{F}$ is called *normal* if $\mathrm{reduce}(\mathcal{F}) = \mathcal{F}$. By the discussion above, there are canonical maps $reduce_{\mathcal{F}} : \mathrm{Ty}(\mathcal{F}) \to \mathrm{Ty}(\mathrm{reduce}(\mathcal{F}))$ coming from rewrites.

The last important thing to note is that, if the input $A = \bigwith_{i \in I} C_i$ and output $B = \bigwith_{o \in O} C_o$ are fixed, up to isomorphism, there are finitely many normal forests $\mathcal{F} : A \to B$ as their size is bounded by $2(|I| + |O|)$. We write $\mathrm{NF}(A, B)$ for a finite set of representative for all normal forests $A \to B$, and given $\mathcal{F} \in \mathrm{NF}(B, C)$ and $\mathcal{F}' \in \mathrm{NF}(A, B)$, we write $\mathcal{F} \circ_{\mathcal{N}} \mathcal{F}'$ for the unique forest in $\mathrm{NF}(A, C)$ which is isomorphic to $\mathrm{reduce}(\mathcal{F} \circ \mathcal{F}')$; an example of the full computation of a $\mathcal{F} \circ_{\mathcal{N}} \mathcal{F}'$ is given in Figure 11. Similarly, we assume that $\mathrm{Grph}(f) \in \mathrm{NF}(A, B)$ for every morphism $f \in \mathsf{Hom}_{\mathcal{C}_\&}(A, B)$.

4.2.3. *Putting everything together.* Let $\mathcal{T} = \left(Q, q_0, (A_q)_{q \in Q}, \delta, i, o\right)$ be a sdm-$\mathfrak{C}_\&$-SST with input Σ^*. We define the sdm-$\mathfrak{D}$-SST

$$\mathcal{T}' \;=\; \left(\sum_{q \in Q} \mathrm{NF}(A_{q_0}, A_q), (q_0, \mathrm{Grph}(\mathrm{id})), (\mathrm{Ty}(\mathcal{F}))_{q, \mathcal{F}}, \delta', i'o' \right)$$
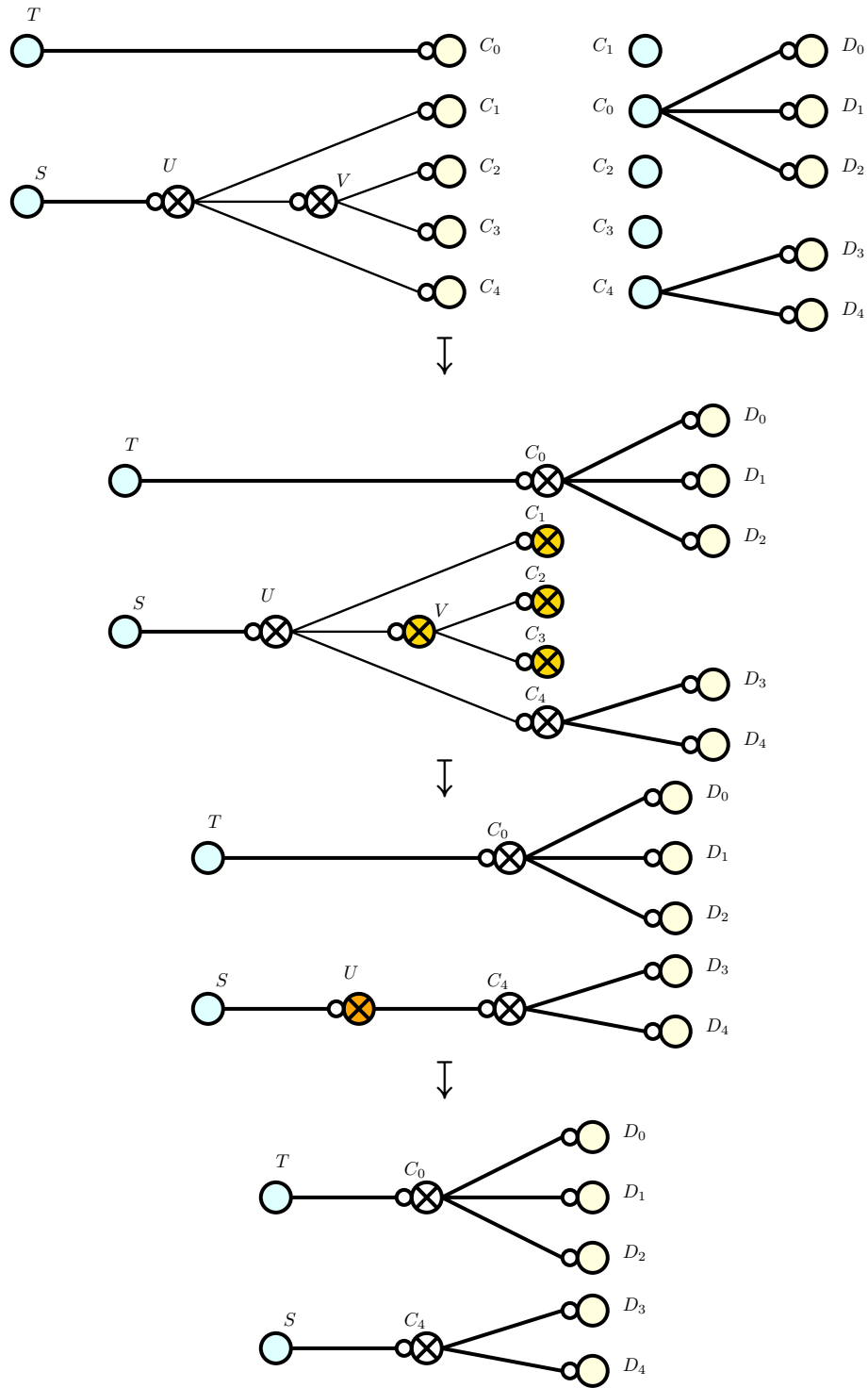
Figure 11: An example of composing normal transformation forests, where first the usual composition and then two big steps of reduction, corresponding respectively to pruning and contracting, are carried out in succession.

$\delta^{\mathcal{D}}(a)_{q,\mathcal{F}}$	$o'_{q,\mathcal{F}}$
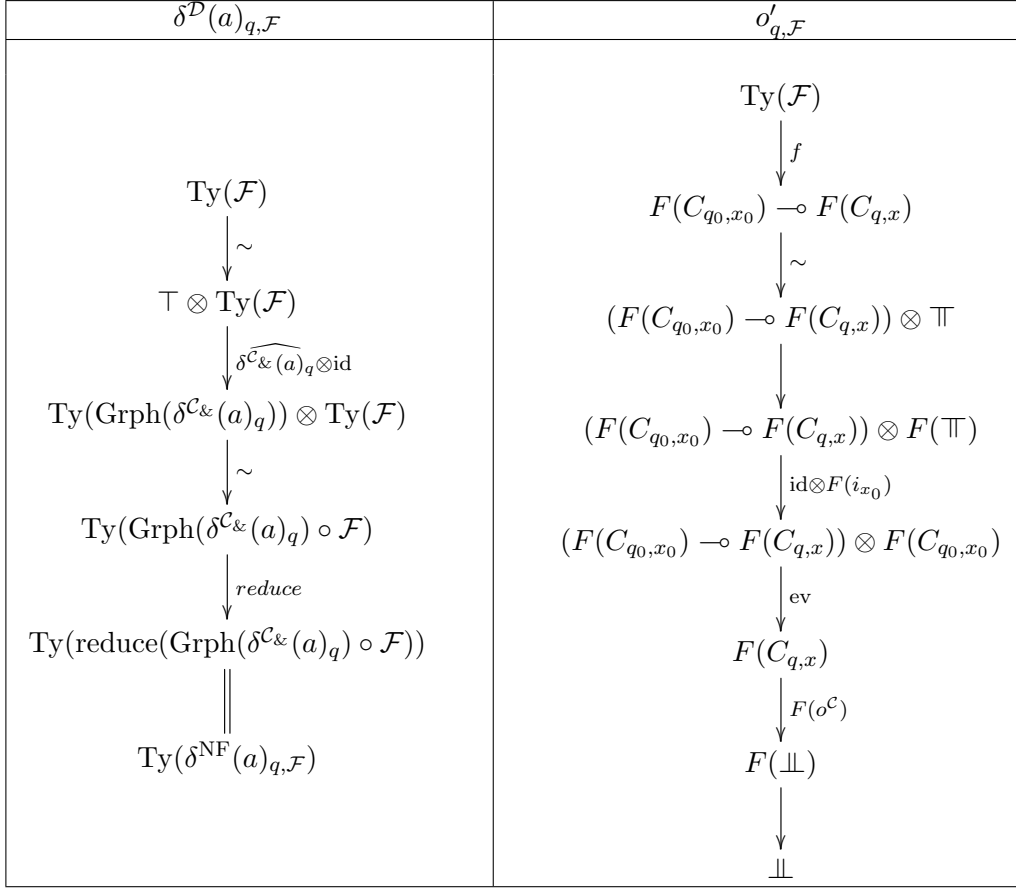
Figure 12: Definition of $\delta^{\mathcal{D}}(a)_{q,\mathcal{F}}$ and $o'_{q,\mathcal{F}}$.

where

$$\delta' \;:\; \Sigma \to \prod_{q,\mathcal{F}} \sum_{r,\mathcal{F}'} \mathsf{Hom}_{\mathcal{D}}\left(\mathrm{Ty}(\mathcal{F}), \mathrm{Ty}(\mathcal{F}')\right)$$

$$i' \in \mathsf{Hom}_{\mathcal{D}}\left(\mathbb{T}, \mathrm{Grph}(\mathrm{id})\right) \qquad\qquad o' \in \prod_{q,\mathcal{F}} \mathsf{Hom}_{\mathcal{D}}\left(\mathrm{Ty}(\mathcal{F}), \mathbb{\perp\!\!\!\perp}\right)$$

are given as follows, assuming that $A_q = \bigotimes\limits_{x \in X_q} C_{q,x}$

- Notice that $\mathrm{Ty}(\mathrm{Grph}(\mathrm{id})) = \bigotimes_{x \in X_{q_0}} F(C_{q_0,x}) \multimap F(C_{q_0,x})$; we simply take the constant map corresponding to the X_{q_0}-fold tensor of $\Lambda(\mathrm{id}_{C_{q_0,x}})$ for i'.
- Fix $a \in \Sigma$ and recall that $\delta(a)$ is a family of pairs

$$(\delta^{Q}(a)_q, \delta^{\mathcal{C}_\&}(a)_q)_{q \in Q} \;\in\; \prod_{q \in Q} \sum_{r \in Q} \mathsf{Hom}_{\mathcal{C}_\&}\left(A_q, A_r\right)$$

δ' is then defined by the equation

$$\delta'(a)_{q,\mathcal{F}} \;=\; \left(\left(\delta^{Q}(a)_q, \delta^{\mathrm{NF}}(a)_{q,\mathcal{F}}\right), \delta^{\mathcal{D}}(a)_{q,\mathcal{F}}\right)$$

where we set $\delta^{\mathrm{NF}}(a)_{q,\mathcal{F}} = \mathrm{Grph}(\delta^{\mathcal{C}_\&}(a)_q) \circ_{\mathcal{N}} \mathcal{F}$ and $\delta^{\mathcal{D}}(a)_{q,\mathcal{F}}$ is obtained as in Figure 12
- Finally, for $q, \mathcal{F}$ ranging over states of $\mathcal{T}'$, we define $o'_{q,\mathcal{F}}$. Recall that $\mathcal{F}$ determines a canonical family

$$\llbracket \mathcal{F} \rrbracket \in \prod_{x \in X_q} \sum_{x_0 \in X_{q_0}} \mathsf{Hom}_{\mathcal{D}}\left(\mathrm{Ty}(\mathcal{F}), F(C_{q_0,x_0}) \multimap F(C_{q,x})\right)$$

First, as $\perp\!\!\!\perp^{\mathfrak{C}_\&} = \iota_\&(\perp\!\!\!\perp^{\mathfrak{C}})$, note that there is a unique $x \in X_q$ such that $o_q = (x, o^{\mathcal{C}})_*$ for some $o^{\mathcal{C}} \in \mathsf{Hom}_{\mathcal{C}}\left(C_{q,x}, \perp\!\!\!\perp^{\mathfrak{C}}\right)$. Writing the pair $\llbracket \mathcal{F} \rrbracket_x$ as (x_0, f), $o'_{q,\mathcal{F}}$ is depicted in Figure 12.

We omit the proof that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{T}' \rrbracket$ by induction over the length of an input word. While spelling it out may be a bit notation-heavy, there is no particular difficulty considering the remarks above linking $\llbracket - \rrbracket$, the composition of transformation forests and their normal forms. $\qquad\square$

5. Regular tree functions in the $\lambda \ell^{\oplus\&}$-calculus

The goal of this section is now to prove our main theorem for tree functions (that we recall below), extending the case of strings (Theorem 1.1).

Theorem 1.2. *Let Σ and Γ be ranked alphabets. A function $\mathbf{Tree}(\Sigma) \to \mathbf{Tree}(\Gamma)$ is $\lambda \ell^{\oplus\&}$-definable if and only if it is regular.*

To prove this theorem, we follow a similar approach as in Section 3.

Remark 5.1. While the result on strings follows as a corollary of Theorem 1.2, this relies on the equivalence between regular string functions and regular tree functions when strings are regarded as particular trees. Given that we start from transducer-based characterization of these notions, this means that we would use e.g. [AD17, Theorem 3.16-3.17], which are themselves non-trivial results[24].

We thus first define a generalized notion of bottom-up ranked tree transducers (BRTT) parameterized by a notion of *tree streaming setting*.

Definition 5.2. Let X be a set. A *tree streaming setting* with output X is a tuple $\mathfrak{C} = (\mathcal{C}, \otimes, \mathbf{I}, \perp\!\!\!\perp, (\!|-|\!))$ where
- $(\mathcal{C}, \otimes, \mathbf{I})$ is a *symmetric monoidal* category
- $\perp\!\!\!\perp$ is an object of $\mathcal{C}$
- $(\!|-|\!)$ is a set-theoretic map $\mathsf{Hom}_{\mathcal{C}}(\mathbf{I}, \perp\!\!\!\perp) \to X$

This notion essentially differs by asking that the underlying category be equipped with a symmetric monoidal product, which is used in defining the semantics of $\mathfrak{C}$-BRTTs. The tensor product is used to fit the branching structure of trees and $\mathbf{I}$ is used for terminal nodes (so there is no need of a distinguished initial object $\top$ as in string streaming settings).

Definition 5.3. Let $\mathfrak{C} = (\mathcal{C}, \otimes, \mathbf{I}, \perp\!\!\!\perp, (\!|-|\!))$ be a tree streaming setting with output X. A $\mathfrak{C}$-BRTT with input (ranked) alphabet Σ and output X is a tuple (Q, R, δ, o) where
- Q is a finite set of states
- R is an object of $\mathcal{C}$

---

[24]A self-contained proof is also possible by building on our development by exploiting Theorem 3.48, arguably the most intricate argument presented here.

- δ is a function $\prod_{a \in \Sigma} \left(Q^{\mathrm{ar}(a)} \to Q \times \mathsf{Hom}_{\mathcal{C}} \left(\bigotimes_{\mathrm{ar}(a)} R \,,\, R \right) \right)$

- $o \in \mathsf{Hom}_{\mathcal{C}} (R, \perp\!\!\!\perp)$ is an output morphism

Its semantics is a set-theoretic map $\mathbf{Tree}(\Sigma) \to X$ defined as follows: writing $\delta_Q(a, (t_i)_{i \in \mathrm{ar}(a)})$ for $\pi_1(\delta(a, (t_i)_{i \in \mathrm{ar}(a)}))$ and $\delta_{\mathcal{C}}(a, (t_i)_{i \in \mathrm{ar}(a)})$ for $\pi_2(\delta(a, (t_i)_{i \in \mathrm{ar}(a)}))$, define auxiliary functions $\delta_Q^* : \mathbf{Tree}(\Sigma) \to Q$ and $\delta_{\mathcal{C}}^* : \mathbf{Tree}(\Sigma) \to \mathsf{Hom}_{\mathcal{C}} (\mathbf{I}, R)$ by iterating δ:

$$\delta_Q^* \left(a \left((t_i)_{i \in \mathrm{ar}(a)} \right) \right) \quad = \quad \delta_Q \left(a, (\delta_Q^*(t_i))_{i \in \mathrm{ar}(a)} \right)$$

$$\delta_{\mathcal{C}}^* \left(a \left((t_i)_{i \in \mathrm{ar}(a)} \right) \right) \quad = \quad \delta_{\mathcal{C}} \left(a, (\delta_Q^*(t_i))_{i \in \mathrm{ar}(a)} \right) \circ \left(\bigotimes_{i \in \mathrm{ar}(a)} \delta_{\mathcal{C}}^*(t_i) \right) \circ \varphi_{\mathrm{ar}(a)}$$

where $\varphi_{\mathrm{ar}(a)}$ is the unique isomorphism $\mathbf{I} \xrightarrow{\sim} \bigotimes_{i \in \mathrm{ar}(a)} \mathbf{I}$ generated by the associator and unitors of $(\mathcal{C}, \otimes, \mathbf{I})$. The output function $[\![\mathcal{T}]\!] : \mathbf{Tree}(\Sigma) \to X$ is defined as $[\![\mathcal{T}]\!] = \langle\!| o \circ \delta_{\mathcal{C}}^* |\!\rangle$.

Remark 5.4. Strictly speaking, we do not need the monoidal product to be symmetric for the notion to make sense, but it would require using a fixed order over the input ranked alphabet Σ. Although one could choose an arbitrary total order over Σ, different orders might define different classes of functions $\mathbf{Tree}(\Sigma)$ when the monoidal product is not symmetric. This is why we work in symmetric monoidal categories.

As with string streaming settings, it is convenient to define a notion of morphism of tree streaming settings to compare the expressiveness of classes of BRTTs.

Definition 5.5. Let $\mathfrak{C} = (\mathcal{C}, \mathbf{I}_{\mathcal{C}}, \otimes_{\mathcal{C}}, \perp\!\!\!\perp_{\mathcal{C}}, \langle\!|-|\!\rangle_{\mathfrak{C}})$ and $\mathfrak{D} = (\mathcal{D}, \mathbf{I}_{\mathcal{D}}, \otimes_{\mathcal{C}}, \perp\!\!\!\perp_{\mathcal{D}}, \langle\!|-|\!\rangle_{\mathfrak{D}})$ be tree streaming settings with output X. A morphism of tree streaming settings is given by a lax monoidal functor $F : (\mathcal{C}, \otimes_{\mathcal{C}}, \mathbf{I}_{\mathcal{C}}) \to (\mathcal{D}, \otimes_{\mathcal{D}}, \mathbf{I}_{\mathcal{D}})$ and a $\mathcal{D}$-arrow $o : F(\perp\!\!\!\perp_{\mathcal{C}}) \to \perp\!\!\!\perp_{\mathcal{D}}$ such that, for every $f \in \mathsf{Hom}_{\mathcal{C}} (\top_{\mathcal{C}}, \perp\!\!\!\perp_{\mathcal{C}})$, we have

$$\langle\!| o \circ F(f) \circ i |\!\rangle_{\mathcal{D}} = \langle\!| f |\!\rangle_{\mathcal{C}}$$

where $i : \mathbf{I}_{\mathcal{D}} \to F(\mathbf{I}_{\mathcal{C}})$ is obtained as part of the lax monoidal functor structure over F.

Observe that we do not require those functors to commute with the symmetry morphisms for the monoidal products, as promised in Section 2.4. This is consistent with the fact that the symmetries are not really involved in computing the image of a tree by a $\mathfrak{C}$-BRTT, according to Remark 5.4: it is only their mere existence that matters.

Lemma 5.6. *If there is a morphism of tree streaming settings $\mathfrak{C} \to \mathfrak{D}$, then $\mathfrak{D}$-BRTTs subsume $\mathfrak{C}$-BRTTs.*

Proof. Let us give the proof for single-state BRTTs; the proof for general BRTTs would be notationally heavier but not much more insightful. Suppose that we have some single-state $\mathfrak{C}$-BRTT $\mathcal{T} = (R, \delta, o)$ – where we leave the only state implicit and regard the transition function δ and output function o as elements of $\prod_{a \in \Sigma} \mathsf{Hom}_{\mathcal{C}} \left(\bigotimes_{\mathrm{ar}(a)} R \,,\, R \right)$ and $\mathsf{Hom}_{\mathcal{C}} (R, \perp\!\!\!\perp)$ respectively – and that the morphism under consideration is composed of a lax monoidal functor $F : \mathcal{C} \to \mathcal{D}$ and a $\mathcal{D}$-arrow $o' : F(\perp\!\!\!\perp_{\mathcal{C}}) \to \perp\!\!\!\perp_{\mathcal{D}}$.

Since F is lax monoidal, we have a family of natural transformations

$$m_{I,A} : \quad \bigotimes_I F(A) \quad \longrightarrow \quad F \left(\bigotimes_I A \right)$$

where I ranges over all finite sets[25] and A over objects of $\mathcal{C}$; this family is compatible with the associators and unitors in $\mathcal{C}$ and $\mathcal{D}$. Furthermore, $m_{\varnothing,A} : \mathbf{I} \to F(\mathbf{I})$ is the same for all $A \in \mathsf{Obj}(C)$, so we shall abbreviate it as $m_\varnothing$.

We claim that $[\![\mathcal{T}]\!] = [\![\mathcal{T}']\!]$, where $\mathcal{T}'$ is the single-state $\mathfrak{D}$-BRTT $(F(R), \delta', o' \circ F(o))$ with the same typing conventions and $\delta'_a = F(\delta_a) \circ m_{\mathrm{ar}(a),R}$. Let us prove this. To do so we first consider the iterations of the transition functions

$$\delta^* : \mathbf{Tree}(\mathbf{\Sigma}) \to \mathsf{Hom}_\mathcal{C}(\mathbf{I}, R) \qquad \text{and} \qquad \delta'^* : \mathbf{Tree}(\mathbf{\Sigma}) \to \mathsf{Hom}_\mathcal{D}(\mathbf{I}, F(R))$$

and show that $\delta'^*(t) = F(\delta(t)) \circ m_\varnothing$ by induction over $t \in \mathbf{Tree}(\mathbf{\Sigma})$. So suppose that $t = a((u_x)_{x \in \mathrm{ar}(a)})$ and that the inductive hypothesis holds (this also takes care of the base case: when $\mathrm{ar}(a) = \varnothing$, the inductive hypothesis is vacuous). In such a case, let us show that each face in the following diagram commutes (where all tensor products have arity $\mathrm{ar}(a)$):

Faces (a) and (f) commute by definition of iterated transition functions and face (e) corresponds to the definition of δ'. Face (d) commutes because of the naturality of $m_{-,-}$ and face (b) because of its compatibility with associators. Finally, face (c) corresponds to the inductive hypothesis. Therefore, the topmost and bottommost paths coincide, so we have $\delta'^*(t) = F(\delta(t)) \circ m_\varnothing$, which concludes our inductive argument. We can then conclude since we have, for every tree t,

$$
\begin{aligned}
[\![\mathcal{T}']\!](t) &= (\! | o' \circ F(o) \circ (\delta'^*(t)) | \!)_\mathfrak{D} & \text{by definition} \\
&= (\! | o' \circ F(o) \circ F(\delta^*(t)) \circ m_\varnothing | \!)_\mathfrak{D} & \text{inductive argument} \\
&= (\! | o' \circ F(o \circ \delta^*(t)) \circ m_\varnothing | \!)_\mathfrak{D} & \text{by functoriality} \\
&= (\! | o \circ (\delta^*(t)) | \!)_\mathfrak{C} & \text{since } (F, o') \text{ is part of a morphism } \mathfrak{C} \to \mathfrak{D} \\
&= [\![\mathcal{T}]\!](t) & \text{by definition}
\end{aligned}
$$

$\square$

From now on, we may omit the "tree" in when discussing streaming settings.

As for SSTs, linearity is an important concept for traditional BRTTs over ranked alphabets. Rather than starting from the category corresponding exactly to usual BRTTs, we will first study a more convenient, albeit less expressive, streaming setting based on the

---

[25]Recall from Section 2.1 that an operation $\bigotimes_{i \in I}(-)$ – here, a functor – is associated to every finite indexing set I by choosing an arbitrary total order over I.

idea of trees with multiple holes. For this, it will be convenient to introduce the notion of *multicategory*, which is essentially a notion of category where morphisms are allowed to have multiple input objects. We will thus first devote Section 5.1 to some preliminaries describing how to (freely) generate affine monoidal categories from multicategories.

After spelling out the universal properties that will allow us to easily define functors and a quick review of the generalization of the results of Section 3.4 pertaining to coproduct completions in Section 5.2, we present a basic multicategory of registers $\mathcal{TR}^{\mathrm{m}}$ containing "trees with holes" in Section 5.3, and the corresponding streaming setting $\mathfrak{TR}$ on top of a category $\mathcal{TR}$. The usual restriction to registers containing "trees with at most one hole" is also discussed and shown to be no less expressive thanks to our basic results on the coproduct completion. Then, we explain how the usual notion of BRTT presented in [AD17] can be shown to have the same expressiveness as $\mathfrak{TR}_{\&}$-BRTTs in Section 5.4.

Finally, in Section 5.5, we show that $\mathcal{TR}_{\oplus}$ has internal homsets $\iota_{\oplus}(R) \multimap \iota_{\oplus}(S)$ and conclude that $\mathcal{TR}_{\oplus\&}$ is monoidal-closed. We conclude the proof of Theorem 1.2 in Section 5.7.

For the rest of this part, we fix a ranked alphabet $\mathbf{\Gamma}$ so that we may focus on outputs contained in $\mathbf{Tree}(\mathbf{\Gamma})$, much like we focussed on outputs in some fixed Γ^* before.

5.1. Multicategorical preliminaries.
This section is devoted to spelling out the formal definition of the notion of multicategory that we use in the sequel, and their relation to symmetric affine monoidal category. While technically necessary for the sequel, it is rather dry and should maybe only be skimmed over at first reading.

Definition 5.7. A (weak symmetric) multicategory $\mathcal{M}$ consists of
- a class of objects $\mathsf{Obj}(\mathcal{M})$
- a class of *multimorphisms* going from pairs $(I, (A_i)_{i \in I})$ of a finite index set I and a family $(A_i)_{i \in I}$ of objects to objects B. We omit the first component of the source and write $\mathsf{Hom}_{\mathcal{M}}((A_i)_{i \in I}, B)$ for the set of these multimorphisms.
- for every object A, a distinguished identity multimorphism $\mathrm{id}_A \in \mathsf{Hom}_{\mathcal{M}}((A)_{* \in 1}, A)$.
- for every set-theoretic map $f : I \to J$, families $(A_i)_{i \in I}$, $(B_j)_{j \in J}$ and object C, a composition operation

$$\begin{array}{ccccc}
\mathsf{Hom}_{\mathcal{M}}((B_j)_{j \in J}, C) & \times & \left[\prod_{j \in J} \mathsf{Hom}_{\mathcal{M}}((A_i)_{i \in f^{-1}(j)}, B_j) \right] & \longrightarrow & \mathsf{Hom}_{\mathcal{M}}((A_i)_{i \in I}, C) \\
\alpha & , & (\beta_j)_{j \in J} & \longmapsto & \alpha *_f \beta
\end{array}$$

- for every bijection $\sigma : I' \to I$ between finite sets, a family of actions

$$\sigma^* : \mathsf{Hom}_{\mathcal{M}}((A_i)_{i \in I}, B) \to \mathsf{Hom}_{\mathcal{M}}((A_{\sigma(i')})_{i' \in I'}, B)$$

correspond to reindexing along symmetries.

Furthermore, the above data is required to obey the following laws.
- The identity morphism be a neutral for composition: for any $\alpha \in \mathsf{Hom}_{\mathcal{M}}((A_i)_{i \in I}, B)$,

$$\mathrm{id}_B *_! (\alpha)_{* \in 1} \quad = \quad \alpha \quad = \quad \alpha *_{\mathrm{id}_I} (\mathrm{id}_{A_i})_{i \in I}$$

- Composition is associative: for any finite sets I, J and K, functions $f : K \to J$ and $g : J \to I$, families of objects $(A_k)_{k \in K}$, $(B_j)_{j \in J}$, $(C_i)_{i \in I}$, D and families of morphisms

$$\begin{array}{l}
\alpha \in \prod_{j \in J} \mathsf{Hom}_{\mathcal{M}}((A_k)_{k \in f^{-1}(j)}, B_j) \\
\beta \in \prod_{i \in I} \mathsf{Hom}_{\mathcal{M}}((B_j)_{j \in g^{-1}(i)}, C_i) \\
\gamma \in \mathsf{Hom}_{\mathcal{M}}((C_i)_{i \in I}, D)
\end{array}$$

the following equation holds

$$\gamma *_g (\beta *_f \alpha) = \left(\gamma *_{g \restriction f^{-1}(j)} \beta\right)_j *_f \alpha$$

- Permutations act functorially: for any $(A_i)_{i \in I}$, B and bijections $\sigma : I' \to I$ and $\sigma' : I'' \to I'$, the following commute

$$\mathsf{Hom}_{\mathcal{M}} \left((A_i)_{i \in I}, B\right) \longrightarrow \mathsf{Hom}_{\mathcal{M}} \left((A_{\sigma(i')})_{i' \in I'}, B\right)$$
$$\mathsf{Hom}_{\mathcal{M}} \left((A_{\sigma(\sigma'(i''))})_{i'' \in I''}, B\right)$$

and $\mathrm{id}_I^* : \mathsf{Hom}_{\mathcal{M}} \left((A_i)_{i \in I}, B\right) \to \mathsf{Hom}_{\mathcal{M}} \left((A_i)_{i \in I}, B\right)$ is the identity.
- Composition is compatible with permutations: for every commuting square

$$
\begin{array}{ccc}
J' & \xrightarrow{\ g\ } & I' \\
\downarrow & & \downarrow{\scriptstyle \sigma} \\
J & \xrightarrow[\ f\ ]{} & I
\end{array}
$$

in FinSet such that the vertical arrows be bijections, note in particular that for every $i \in I$, there is a bijection $\sigma_i : g^{-1}(I') \to f^{-1}(I)$; we thus require that

$$\alpha *_f \left(\beta_{j \in f^{-1}(i)}\right)_{i \in I} \quad = \quad \sigma^*(\alpha) *_g \left(\sigma_i^* \left(\beta_{j \in f^{-1}(i)}\right)\right)_{i \in I}$$

for every suitable α and β_js.

Every symmetric monoidal category $\mathcal{C}$ can be mapped to a multicategory $\mathcal{C}_{\mathrm{mcat}}$ by taking

$$\mathsf{Hom}_{\mathcal{C}_{\mathrm{mcat}}} \left((A_i)_{i \in I}, B\right) \quad = \quad \mathsf{Hom}_{\mathcal{C}} \left(\bigotimes_{i \in I} A_i \, , \, B\right)$$

We may make this map functorial, provided we equip the class of multicategories and the class of symmetric monoidal categories with categorical structures.

Definition 5.8. Given two weak multicategories $\mathcal{M}$ and $\mathcal{N}$, a functor $F : \mathcal{M} \to \mathcal{N}$ consists of maps of objects $F : \mathsf{Obj}(\mathcal{M}) \to \mathsf{Obj}(\mathcal{N})$ and of multimorphisms

$$F \ : \ \mathsf{Hom}_{\mathcal{M}} \left((A_i)_{i \in I}, B\right) \ \longrightarrow \ \mathsf{Hom}_{\mathcal{N}} \left((F(A_i))_{i \in I}, F(B)\right)$$

such that $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$, $F\left(\alpha *_f (\beta_j)_{j \in f^{-1}(i)}\right) = F(\alpha) *_f (F(\beta_j))_{j \in f^{-1}(i)}$ and $F(\sigma * (\alpha)) = \sigma^*(F(\alpha))$ for all suitable objects, index sets, set-theoretic functions and morphisms.

Definition 5.8 gives a class of arrows for a large category MCat of multicategories. Calling Aff the category whose objects are symmetric affine monoidal categories and morphisms are strong monoidal functors, the map $\mathcal{C} \to \mathcal{C}_{\mathrm{mcat}}$ extends to a functor $\mathsf{Aff} \to \mathsf{MCat}$. We are now interested in the inverse process of generating freely a symmetric affine monoidal category out of a weak multicategory.

Definition 5.9. Let $\mathcal{M}$ be a weak multicategory. The *free affine symmetric monoidal category* generated by $\mathcal{M}$ is the category $\mathcal{M}_{\mathrm{aff}}$ such that
- objects are pairs $(I, (A_i)_{i \in I})$ of a finite set I and a family of objects of $\mathcal{M}$; write $\bigotimes_{i \in I} A_i$ for such objects.

- morphisms $\bigotimes_{i \in I} A_i \to \bigotimes_{j \in J} B_j$ are pairs $(f, (\alpha_j)_{j \in J})$ where f is a partial function $I \rightharpoonup J$ and α_j is a $\mathcal{M}$ multimorphism $(A_i)_{i \in f^{-1}(J)} \to B_j$.
- identities $\bigotimes_{i \in I} A_i \to \bigotimes_{i \in I} A_i$ are the pairs $(\mathrm{id}_I, (\mathrm{id}_{A_i})_{i \in I})$.
- the composition of

$$(f, (\alpha_j)_{j \in J}) : \bigotimes_{i \in I} A_i \to \bigotimes_{j \in J} B_j \quad \text{and} \quad (g, (\beta_k)_{k \in K}) : \bigotimes_{j \in J} B_j \to \bigotimes_{k \in K} C_k$$

is $\left(g \circ f, \left(\beta_k * (\alpha_j)_{j \in g^{-1}(k)} \right)_{k \in K} \right)$.

For any bijection $\sigma : I' \to I$, we have canonical isomorphisms $\bigotimes_{i \in I} A_i \to \bigotimes_{i \in I} A_{\sigma(i)}$. We take the binary tensor product to be

$$\left(\bigotimes_{i \in I} A_i \right) \otimes \left(\bigotimes_{j \in J} B_j \right) \quad = \quad \bigotimes_{x \in I+J} \left\{ \begin{array}{ll} A_i & \text{if } x = \mathrm{in}_1(i) \\ B_j & \text{if } x = \mathrm{in}_2(j) \end{array} \right.$$

and the unit to be the terminal object, which is the nullary family $\bigotimes_{\varnothing}$. The associator and symmetries are induced by the isomorphisms $(I + J) + K \cong I + (J + K)$ and $I + J \cong J + I$ respectively, and the units by $I + \varnothing \cong I \cong \varnothing + I$. The axioms of weak symmetric multicategories then imply that this indeed endows $\mathcal{M}_{\mathrm{aff}}$ with a symmetric affine monoidal structure. We skip checking the details.

5.2. **The coproduct completion.** Similarly as for strings, the coproduct completion of a category induces a map $\mathfrak{C} \mapsto \mathfrak{C}_\oplus$ over tree streaming settings. Furthermore, expressiveness of $\mathfrak{C}$ and $\mathfrak{C}_\oplus$ remain the same under similar hypotheses as Theorem 3.32.

Theorem 5.10. *Let $\mathfrak{C}$ be a tree streaming setting whose monoidal product is affine and such that all objects of the underlying category have unitary support. Then, $\mathfrak{C}$-BRTTs and $\mathfrak{C}_\oplus$-BRTTs are equi-expressive.*

The proof is an unsurprising adaptation of the one of Theorem 3.32. We leave it to the interested reader. Similarly, we define the notion of a state-dependent memory $\mathfrak{C}$-streaming tree transducer (sdm-$\mathfrak{C}$-BRTT), which can be shown to be as expressive as sdm-$\mathfrak{C}_\oplus$-BRTT. We only state the definition and the generalization of Lemma 3.35, whose proof we defer to the interested reader.

Definition 5.11. A $\mathfrak{C}$-*state-dependent memory BRTT* with input $\mathbf{Tree}(\Sigma)$ is a tuple $(Q, \delta, (C_q)_{q \in Q}, o)$ where

- Q is a finite set of states
- $(C_q)_{q \in Q}$ is a Q-indexed family of objects of $\mathcal{C}$
- $\delta \in \left[\prod_{a \in \Sigma} \prod_{q \in Q^{\mathrm{ar}(a)}} \sum_{r \in Q} \mathsf{Hom}_{\mathcal{C}} \left(\bigotimes_{x \in \mathrm{ar}(a)} C_{q(x)}, C_r \right) \right]$ is a transition function
- $o \in \prod_{q \in Q} \mathsf{Hom}_{\mathcal{C}} (C_q, \bot\!\!\!\bot)$ is the output family of morphisms

Lemma 5.12. *Let $\mathfrak{C}$ be a streaming setting. State-dependent memory $\mathfrak{C}$-BRTTs are as expressive as $\mathfrak{C}_\oplus$-BRTTs.*

5.3. **The combinatorial multicategory $\mathcal{TR}^{\mathrm{m}}$.** We are now ready to give a smooth definition of a category of register transition for trees, generalizing Proposition 3.12. As announced, we find it more convenient to first give a multicategory $\mathcal{TR}^{\mathrm{m}}$ and then move to monoidal categories by taking $\mathcal{TR} = (\mathcal{TR}^{\mathrm{m}})_{\mathrm{aff}}$. We then discuss the restriction consisting of limiting the number of holes in the tree expressions stored in register to at most one and show that it is not limiting.

Notations Recall that we regard ranked alphabets $\mathbf{R}$ as pairs (R, ar) where R is a finite set of letters and $(\mathrm{ar}(a))_{a \in R}$ is a family $R \to \mathsf{FinSet}$ of arities. Given two ranked alphabets $\mathbf{R}$ and $\mathbf{S}$, we suggestively write $\mathbf{R} \otimes \mathbf{S}$ for the ranked alphabet $(R + S, [\mathrm{ar}, \mathrm{ar}])$. Given a finite set U, call $\mathcal{O}(U)$ the ranked alphabet $(U, (\varnothing)_{u \in U})$ consisting of $|U|$-many terminal letters and $\mathcal{I}(U)$ for the ranked alphabet consisting of a single letter of arity U. Given a ranked alphabet $\boldsymbol{\Sigma} = (\Sigma, \mathrm{ar})$ and a subset $X \subseteq \Sigma$, we write $\boldsymbol{\Sigma} \upharpoonright X$ for the restriction $(X, \mathrm{ar} \upharpoonright X)$.

5.3.1. *Definition of $\mathcal{TR}$.* Before giving the definition of $\mathcal{TR}^{\mathrm{m}}$, we first need to make formal a notion of trees with linearly many occurrences of certain constructors.

Definition 5.13. Let $\mathbf{R}$ be a ranked alphabet. We define the set $\mathbf{LTree_{\Gamma}}(\mathbf{R})$ of $\mathbf{R}$-linear trees as the set of $(\boldsymbol{\Gamma} \otimes \mathbf{R})$-trees $\mathbf{Tree}(\boldsymbol{\Gamma} \otimes \mathbf{R})$ such that all constructors of $\mathbf{R}$ appear exactly once.

Definition 5.14. Define $\mathcal{TR}^{\mathrm{m}}(\boldsymbol{\Gamma})$ (abbreviated $\mathcal{TR}^{\mathrm{m}}$ in the sequel) as the multicategory
- whose class of objects $\mathsf{Obj}(\mathcal{TR}^{\mathrm{m}})$ is FinSet.
- whose class of multimorphisms from $(A_i)_{i \in I}$ to B is the set of linear trees over the joint alphabet $(I, A) \otimes \mathcal{O}(B)$ (recall that (I, A) can formally be regarded as a ranked alphabet: its set of letters is I and the arity of $i \in I$ is $A(i) = A_i$).
$$\mathsf{Hom}_{\mathcal{TR}^{\mathrm{m}}}((A_i)_{i \in I}, B) \quad = \quad \mathbf{LTree_{\Gamma}}((I, A) \otimes \mathcal{O}(B))$$
- whose composition operations are given by substitution: given a map $f : I \to J$ and multimorphisms
$$t \in \mathbf{LTree_{\Gamma}}((J, B) \otimes \mathcal{O}(C)) \qquad \text{and} \qquad u \in \prod_{j \in J} \mathbf{LTree_{\Gamma}}((f^{-1}(j), (A_i)_i) \otimes \mathcal{O}(B_j))$$

the composite $t *_f u$ is defined by recursion over t:
 - if $t = a((t'_k)_k)$ for some $a = \mathsf{in}_1(b)$ with $b \in \Gamma$ or $a = \mathsf{in}_2(\mathsf{in}_2(c))$ for $c \in C$, then
$$t *_f u = a((t'_k *_f u)_k)$$
 - otherwise $t = \mathsf{in}_2(\mathsf{in}_1(j))((t'_b)_{b \in B_j})$ with $j \in J$ and t'_b in some $\mathbf{LTree_{\Gamma}}((J_b, B_b) \otimes \mathcal{O}(C_j))$ for $\bigcup_{b \in B_j} J_b = J \setminus \{j\}$, $B_b = B \upharpoonright J_b$ and $\bigcup_{b \in B_j} C_b = C$. In such a case, we set
$$t *_f u = u_j[(t'_b *_{\mathrm{id}_{J_b}} (u_{j'})_{j' \in J_b})/b]_{b \in B_j}$$

where $[-/-]_{-\in -}$ denotes the more usual substitution of leaves by subtrees (recall that every $b \in B$, and a fortiori B_j has arity $\varnothing$).

While the definition of composition of multimorphisms in $\mathcal{TR}^{\mathrm{m}}$ looks daunting, we claim it is rather natural. Figure 13 depicts the composition $\alpha *_f (\beta_x)_{x \in \{t, u\}}$ with
$$\alpha = t(a(u(c()), *()), c()) \qquad \beta_t = a(x(p(), b(q())), y()) \quad \text{and} \quad \beta_u = z(c(), r(), c())$$

We now set $\mathcal{TR} = (\mathcal{TR}^{\mathrm{m}})_{\mathrm{aff}}$; while objects of $(\mathcal{TR}^{\mathrm{m}})_{\mathrm{aff}}$ are supposed to be families of finite sets $(A_i)_{i \in I}$, in the sequel, we sometimes identify them with the ranked alphabets
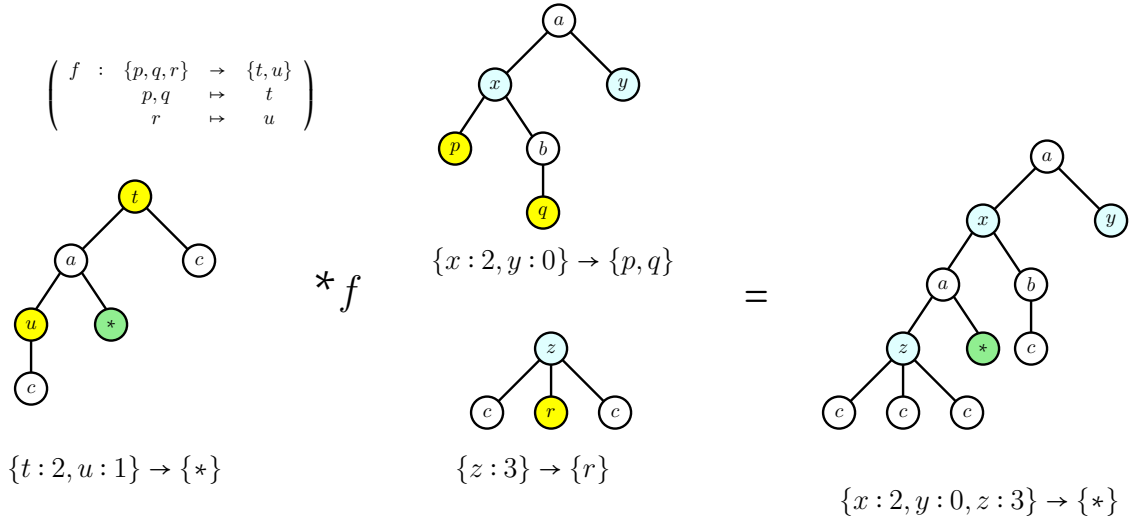
Figure 13: Composition of some multimorphisms of $\mathcal{TR}^{\mathrm{m}}$.

(I, A) in $\mathcal{TR}$ for notational convenience. As such, the notation $\mathbf{R} \otimes \mathbf{S}$ corresponds to the expected tensorial product in $\mathcal{TR}$.

We are now ready to define our first tree streaming setting.

Definition 5.15. $\mathfrak{TR}$ is the tree streaming setting $(\mathcal{TR}, \otimes, \top, \mathcal{I}(\varnothing), (\!|-\!|))$, where $(\!|-\!|)$ is the canonical isomorphism $\mathsf{Hom}_{\mathcal{TR}}(\top, \mathcal{I}(\varnothing)) \cong \mathsf{Hom}_{\mathcal{TR}^{\mathrm{m}}}((\!)_{\varnothing}, \varnothing) \cong \mathbf{LTree}_{\boldsymbol{\Gamma}}(\varnothing) \cong \mathbf{Tree}(\boldsymbol{\Gamma})$ (where $(\!)_{\varnothing}$ is the empty family).

Call $\mathcal{TR}^{\mathrm{m}, \leq 1}$ the full submulticategory of $\mathcal{TR}^{\mathrm{m}}$ whose objects are empty or singleton sets, and $\mathcal{TR}^{\leq 1} \cong \mathcal{TR}^{\mathrm{m}, \leq 1}_{\mathrm{aff}}$ to be the corresponding full subcategory of $\mathcal{TR}$. The monoidal structure of $\mathcal{TR}$ restricts to $\mathcal{TR}^{\leq 1}$ without any difficulty, and that $\mathcal{I}(\varnothing)$ is an object of $\mathcal{TR}^{\leq 1}$. This means that $\mathfrak{TR}$ has a restriction to a streaming setting $\mathfrak{TR}^{\leq 1}$.

While $\mathfrak{TR}^{\leq 1}$ turns out to be more elementary and a good building block toward the definition of usual BRTTs, it is easier to show the monoidal closure of $\mathfrak{TR}_{\oplus \&}$ than $\mathfrak{TR}^{\leq 1}$. Thankfully, it turns out that the expressiveness of BRTTs over $\mathfrak{TR}$ and $\mathfrak{TR}^{\leq 1}$ is the same.

For one direction, there is a morphism $\mathfrak{TR}^{\leq 1} \to \mathfrak{TR}$ corresponding to the embedding $\mathcal{TR}^{\leq 1} \to \mathcal{TR}$. For the other direction, we exploit Theorem 5.10. The proof involves some combinatorics, but nothing surprising as it amounts to the classical decomposition of multi-hole trees into families of single-hole trees as found in e.g. [AD17, §3.5].

Lemma 5.16. *There is a morphism of streaming settings* $\mathfrak{TR} \to \mathfrak{TR}^{\leq 1}_{\oplus}$.

Proof sketch. We focus on giving enough ingredients to define the underlying (strong) monoidal functor $F : \mathcal{TR} \to \mathcal{TR}^{\leq 1}_{\oplus}$, which is going to preserve $\bot$ (i.e., we will have $F(\mathcal{I}(\varnothing)) \cong \iota_{\oplus}(\mathcal{I}(\varnothing)))$.

Rather than giving a direct explicit construction of F (which is rather tedious over morphisms), we obtain it as a composition of two strong monoidal functors: the strong monoidal embedding $\iota_{\oplus} : \mathcal{TR} \to \mathcal{TR}_{\oplus}$ and a functor $R : \mathcal{TR}_{\oplus} \to \mathcal{TR}^{\leq 1}_{\oplus}$ right adjoint to the

inclusion $I : \mathcal{TR}_\oplus^{\leq 1} \to \mathcal{TR}_\oplus$.

$$\mathcal{TR} \xrightarrow{\iota_\oplus} \mathcal{TR}_\oplus \overset{R}{\underset{I}{\rightleftarrows}} \mathcal{TR}_\oplus^{\leq 1}$$

I is strong symmetric monoidal. Therefore, by [Mel09, Proposition 14, Section 5.17], once we construct R right adjoint to I, it comes equipped with a canonical lax monoidal structure. Furthermore, since we want $I \dashv R$, we can use the implicit characterization of adjoints given in [ML98, item (iv), Theorem 2, Section IV.1]: to define R, it suffices to give the value of $R(A)$ for every object $A \in \mathsf{Obj}(\mathcal{TR}_\oplus)$ and counit maps $\epsilon_A : I(R(A)) \to A$ such that, for every object $B \in \mathsf{Obj}(\mathcal{TR}_\oplus^{\leq 1})$ and map $h \in \mathsf{Hom}_{\mathcal{TR}_\oplus}(I(B), A)$, there is a unique $\widetilde{h} \in \mathsf{Hom}_{\mathcal{TR}_\oplus^{\leq 1}}(R(A), B)$ such that the following diagram commutes

$$I(B) \dashrightarrow^{I(\widetilde{h})} I(R(A))$$

with h going to A and ϵ_A from $I(R(A))$ to A.

So we only need to define $R(A)$ and ϵ_A to obtain our functor R; once those are defined, we leave checking that the universal property holds to the reader. We first focus on the case where $A = \iota_\oplus(\mathcal{I}(U))$ for some finite set U. Recall that a single-letter alphabet $\mathcal{I}(U)$, when seen as an object of $\mathcal{TR}$, should be should be intuitively regarded as a register containing a tree with U-many holes. If $|U| \leq 1$, we may simply take $R(\iota_\oplus(\mathcal{I}(U))) = \iota_\oplus(\mathcal{I}(U))$. Otherwise, $|U| \geq 2$ and $\mathcal{I}(U)$ is not an object $\mathcal{TR}^{\leq 1}$; in that case, we use the following recursive definition

$$R(\iota_\oplus(\mathcal{I}(U))) \quad = \quad \mathcal{I}(1) \quad \otimes \bigoplus_{b \in \Gamma} \bigoplus_{\substack{f : U \to \mathrm{ar}(b) \\ \text{nonconstant}}} \bigotimes_{x \in \mathrm{ar}(b)} R(\iota_\oplus(\mathcal{I}(f^{-1}(x))))$$

Note that this definition is well-founded because the function f in the second sum is taken to be non constant, so that $|f^{-1}(x)| < |U|$ for every x. While this suffices as a definition of $R(\iota_\oplus(\mathcal{I}(U)))$, this might be a bit opaque without having the definition of $\epsilon_{R(\iota_\oplus(\mathcal{I}(U)))}$. Before giving that, let us attempt to give an intuitive rationale behind this definition: there is an isomorphism[26]

$$\mathsf{Hom}_{\mathcal{TR}_\oplus^{\leq 1}}(\top, R(\iota_\oplus(\mathcal{I}(U)))) \cong \mathbf{LTree}_\Gamma(U)$$

which can be nicely pictured, provided we actually compute recursively $R(\iota_\oplus(\mathcal{I}(U)))$ and spell out a normal form

$$R(\iota_\oplus(\mathcal{I}(U))) \quad \cong \bigoplus_{t \in \mathbf{PT}(U)} \bigotimes_{n \in N(t)} A_n$$

with all $A_n = \mathcal{I}(\varnothing)$ or $A_n \cong \mathcal{I}(1)$. It is always possible to build a suitable set $\mathbf{PT}(U)$ simply because all objects of $\mathcal{TR}_\oplus^{\leq 1}$ have this shape, but an intuitive definition of what one might call a set of *partitioning trees over* U is also possible for $\mathbf{PT}$, and $N(t)$ would then

---

[26]Which we may later on define formally as a composite

$$\mathsf{Hom}_{\mathcal{TR}_\oplus^{\leq 1}}(\top, R(\iota_\oplus(\mathcal{I}(U)))) \xrightarrow{I} \mathsf{Hom}_{\mathcal{TR}_\oplus}(\top, R(\iota_\oplus(\mathcal{I}(U)))) \to \mathsf{Hom}_{\mathcal{TR}_\oplus}(\top, \mathcal{I}(U)) \xrightarrow{\sim} \mathbf{LTree}_\Gamma(U)$$

where the mediating arrow is the post-composition by $\epsilon_{\iota_\oplus(\mathcal{I}(U))}$.
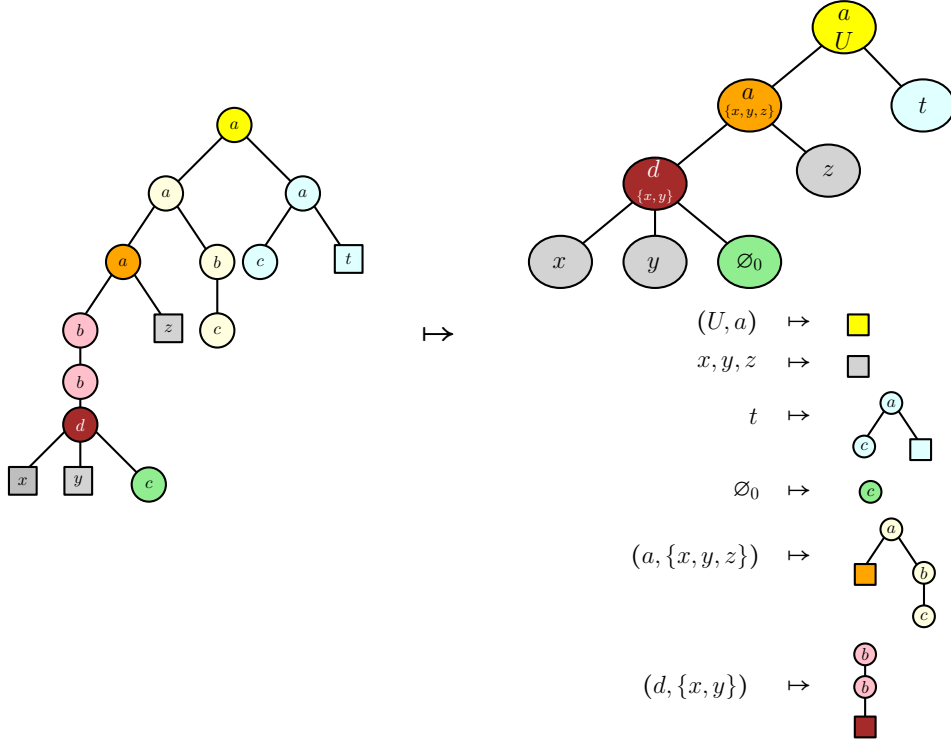
Figure 14: Decomposition of a multi-hole tree $\mathbf{LTree}_{\{a:2,b:1,c:0,d:3\}}(\{x,y,z,t\})$ as a tuple consisting of a partitioning tree and trees with at most one hole.

correspond to the nodes of the trees. We skip defining this notion formally, but note that the announced bijection would then match trees with U-many holes with pairs $(t, (u_i)_{i\in N(t)})$ of a partitioning tree t and a family of trees with at most one-hole $(u_i)_{i\in N(t)}$. This bijective correspondence is pictured in Figure 14.

Now, we define $\epsilon_{R(\iota_\oplus(\mathcal{I}(U)))} \in \mathsf{Hom}_{\mathcal{TR}_\oplus^{\leq 1}}(I(R(\iota_\oplus(\mathcal{I}(U)))), \iota_\oplus(\mathcal{I}(U)))$, by induction over the size of U. If $|U| \leq 1$, we take $\epsilon_{R(\iota_\oplus(\mathcal{I}(U)))} : \iota_\oplus(\mathcal{I}(U)) \to \iota_\oplus(\mathcal{I}(U))$ to be the identity. Otherwise, we need to define a map

$$g_U \quad : \quad I\left(\bigoplus_{b\in\Gamma} \bigoplus_{\substack{f:U\to\mathrm{ar}(b)\\ \text{nonconstant}}} \bigotimes_{x\in\mathrm{ar}(b)} R(\iota_\oplus(\mathcal{I}(f^{-1}(x))))\right) \quad \longrightarrow \quad \iota_\oplus(\mathcal{I}(U))$$

or, equivalently, a family of $\mathcal{TR}_\oplus$-maps indexed by $b \in \Gamma$ and $f : U \to \mathrm{ar}(b)$ non-constant

$$g_{b,f} \quad : \quad \bigotimes_{x\in\mathrm{ar}(b)} I(R(\iota_\oplus(\mathcal{I}(f^{-1}(x))))) \quad \longrightarrow \quad \iota_\oplus(\mathcal{I}(U))$$

By the induction hypothesis, we have a family of $\mathcal{TR}_\oplus$-maps $(g_x)_{x\in\mathrm{ar}(b)}$

$$g_x \quad : \quad I(R(\iota_\oplus(\mathcal{I}(f^{-1}(x))))) \quad \longrightarrow \quad \iota_\oplus(\mathcal{I}(f^{-1}(x)))$$

We define $g_{b,f}$ as the composite

$$\bigotimes_{x \in \mathrm{ar}(b)} I(R(\iota_\oplus(\mathcal{I}(f^{-1}(x))))) \xrightarrow{\bigotimes_x g_x} \bigotimes_{x \in \mathrm{ar}(b)} \iota_\oplus(\mathcal{I}(f^{-1}(x))) \xrightarrow{\overline{b}} \iota_\oplus(\mathcal{I}(U))$$

where $\overline{b}$ is obtained from the map of $\mathcal{TR}^{\mathrm{m}}$ which intuitively takes a family $(t_x)_{x \in \mathrm{ar}(b)}$ of trees into a single tree $b((t_x)_{x \in \mathrm{ar}(b)})$ (officially, the tree $b((*)_{x \in \mathrm{ar}(b)}) \in \mathbf{LTree}_{\boldsymbol{\Gamma}}(\mathcal{O}(U))$).

Now, R is defined on objects of the shape $\iota_\oplus(\mathcal{I}(U))$, as well as ϵ, so we need to extend this to the whole category $\mathcal{TR}_\oplus$. Recall that every object A of $\mathcal{TR}_\oplus$ can be written as $A = \bigoplus_{v \in V} \bigotimes_{j \in J_u} \iota_\oplus(\mathcal{I}(U_j))$. In the end, the functor R is expected to be strong monoidal, and we may force it to preserve coproducts, so we set

$$R(A) \;=\; \bigoplus_{v \in V} \bigotimes_{j \in J_u} R(\iota_\oplus(\mathcal{I}(U_j))) \qquad\qquad \epsilon_A \;=\; \bigoplus_{v \in V} \bigotimes_{j \in J_u} \epsilon_{\iota_\oplus(\mathcal{I}(U_j))}$$

$\square$

5.3.2. *Relationship to* $\lambda\ell^{\oplus\&}$. Recall that if $\boldsymbol{\Gamma} = \{a_1 : A_1, \ldots, a_k : A_k\}$ is an output alphabet, we call $\widetilde{\boldsymbol{\Gamma}}$ the context $a_1 : \mathbb{0} \multimap \ldots \multimap \mathbb{0}, \ldots, a_k : \mathbb{0} \multimap \ldots \multimap \mathbb{0}$ where the type of a_i has $|A_i|$ arguments. Definition 3.20 provides us with a suitable affine monoidal closed category $\mathcal{L}(\widetilde{\boldsymbol{\Gamma}})$, which we still call $\mathcal{L}$ when $\widetilde{\boldsymbol{\Gamma}}$ is clear from context. Since we have a monoidal product, we may easily adapt Definition 3.22 to get a tree streaming setting $\mathfrak{L}$. Then we may relate $\lambda\ell^{\oplus\&}$-definability to (single-state) $\mathfrak{L}$-BRTT.

Lemma 5.17. *Computability by single-state $\mathfrak{L}$-BRTTs and $\lambda\ell^{\oplus\&}$-definability are equivalent for functions* $\mathbf{Tree}(\boldsymbol{\Sigma}) \to \mathbf{Tree}(\boldsymbol{\Gamma})$.

Proof idea. This is proven in a similar way as Lemma 3.23, based on the syntactic Lemma 3.24. The proof is even more straightforward as there is no mismatch between the processing of trees by BRTTs and $\lambda\ell^{\oplus\&}$-terms working with Church encodings, contrary to SSTs for strings (which operate top-down rather than bottom-up when regarding strings as trees). $\square$

We can now notice that $\mathfrak{L}$-BRTTs are more expressive than $\mathcal{TR}^{\leq 1}$-BRTTs thanks to the notion of streaming setting morphisms, much like with strings (this generalizes Lemma 3.25).

Lemma 5.18. *There is a morphism of streaming settings* $\mathfrak{TR} \to \mathfrak{L}$.

Proof sketch. Let us focus on the underlying functor $F : \mathcal{TR} \to \mathcal{L}$. For objects (which are finite sets), we put

$$F\left(\bigotimes_{i \in I} U_i\right) = \bigotimes_{i \in I} \left((\mathbb{0}^{\otimes U_i} \multimap \mathbb{0}) \,\&\, \mathbf{I}\right)$$

A multimorphism $f \in \mathsf{Hom}_{\mathcal{TR}^{\mathrm{m}}}((U_i)_{i \in I}, V)$ is an element of $\mathbf{LTree}_{\boldsymbol{\Gamma}}\left(\left(\bigotimes_{i \in I} \mathcal{I}(U_i)\right) \otimes \mathcal{O}(V)\right)$ which has a Church encoding $\underline{f}$ which has a type isomorphic to $\bigotimes_{i \in I} (\mathbb{0}^{\otimes U_i} \multimap \mathbb{0}) \multimap \mathbb{0}$, and thus embeds into $F\left(\bigotimes_{i \in I} U_i\right)$ through well-typed term ι. We take $F(f_{\mathrm{aff}}) = \lambda x.\iota \, \underline{f}$, and extend this definition to arbitrary morphisms $(f, (\alpha_j)_j) : \bigotimes_{i \in I} U_i \to \bigotimes_{j \in J} V_j$ in $\mathcal{TR}$ by first using the second projection π_2 to restrict to the case where $\mathsf{dom}(f) = I$, and then by piecing together the $F((\alpha_j)_{\mathrm{aff}})$. $\square$

Corollary 5.19. *There is a morphism of streaming settings* $\mathfrak{TR}_{\oplus\&} \to \mathfrak{L}$.

Proof idea. Starting from Lemma 5.18, we have a functor $\mathcal{TR} \to \mathcal{L}$. Since $\mathcal{L}$ has all products and coproducts, the universal properties of the $(-)_\&$ and $(-)_\oplus$ completion yield a functor $F : \mathcal{TR}_{\oplus\&} \to \mathcal{L}$. The monoidal structure of the initial functor $\mathcal{TR} \to \mathcal{L}$ can be lifted accordingly. For any finite family of objects $(((A_k)_{k \in K_j})_{j \in J_i})_{i \in I}$ sitting in a symmetric monoidal closed category with products and coproducts, there are canonical morphisms

$$\left(\bigoplus_{u \in U} \bigwith_{x \in X_u} A_x \right) \otimes \left(\bigoplus_{v \in V} \bigwith_{v \in Y_v} B_y \right) \longrightarrow \bigoplus_{(u,v) \in U \times V} \bigwith_{(x,y) \in X_u \times Y_v} A_x \otimes B_y$$

which are not isomorphisms in general, but constitute the non-trivial part of the lax monoidal structure of F; $m^0 : \mathbf{I} \to F(\mathbf{I})$ is actually the identity. $\square$

5.4. $\mathfrak{TR}_\&$-BRTTs coincide with regular functions, via coherence spaces.

We define a streaming setting $\mathfrak{TR}^\asymp$ and its restriction $\mathfrak{TR}^{\asymp, \leq 1}$ (with respective underlying categories $\mathcal{TR}^\asymp$, $\mathcal{TR}^{\asymp, \leq 1}$) so that $\mathfrak{TR}^{\asymp, \leq 1}$-BRTTs coincide with Alur and D'Antoni's notion of single-use-restricted BRTT [AD17], which they showed to characterize regular tree functions. We then show that there are morphisms of streaming settings $\mathfrak{TR}_\& \to \mathfrak{TR}^\asymp \to \mathfrak{TR}_\&$ and thus establish that $\mathfrak{TR}_\&$-BRTTs capture exactly regular tree functions.

Much like $\mathcal{TR}$, the category $\mathcal{TR}^\asymp$ is obtained by applying a generic construction to $\mathcal{TR}^m$, taking weak symmetric multicategories to symmetric affine monoidal categories. In particular, objects of $\mathcal{TR}^\asymp$ will consist of formal tensor products of objects of $\mathcal{TR}^m$. The main difference is that morphisms of $\mathcal{TR}^\asymp$ will induce a dependency *relation* $D \subseteq I \times J$ over indexing sets, rather than a partial function $J \rightharpoonup I$. This corresponds to a relaxation of the copylessness condition. However, objects of $\mathcal{TR}^\asymp$ will also be equipped with a *conflict relation* $\asymp$ over their indexed sets, and D will be required to satisfy a linearity constraint. Calling $\smile$ the dual *coherence relation* such that $x \smile y$ is equivalent to $x = y \vee \neg(x \asymp y)$, if we have $(i, j) \in D$ and $(i', j') \in D$, the linearity constraint enforces

$$j \smile_J j' \;\Rightarrow\; i \smile_I i' \qquad \text{and} \qquad i \asymp_I i' \;\Rightarrow\; j \asymp_J j'$$

This corresponds to the *single use restriction* imposed on BRTTs [AD17, §2.1], whose introduction was motivated in Section 2.2.3.

Example 5.20. The BRTT that we gave for the "conditional swap" function in Example 2.7 is single-use-restricted according to the above by taking its two registers (i.e. objects of $\mathcal{TR}^m$) to be in conflict.

But as our choice of notation and vocabulary suggests, this is also related to the category of (finite) *coherence spaces*, the first denotational model of linear logic [Gir87] (predated by a similar semantics for system F [Gir86]). As far as we know, this observation is new (the conflict relation is denoted by η in [AD17], while $\asymp$ comes from the linear logic literature). The coherence semantics of the linear λ-calculus has been used in particular by Gallot, Lemay and Salvati [GLS20] to analyze a top-down tree transducer model containing linear λ-terms. Unlike them, we do not use coherence spaces *as a semantics* here; what happens here is much closer the use of a coherence/conflict relation to handle additive connectives – we will indeed show a connection with the $\&$-completion – in proof nets, see [Gir96, Appendix A.1] and [HH16].

Definition 5.21 (see e.g. [Gir95, §2.2.3])**.** A coherence space I is a pair $(\|I\|, \frown_I)$ of a set $\|I\|$, called the *web*, and a binary reflexive symmetric relation $\frown_I$ over $\|I\|$ called the *coherence relation*. As usual, given a coherence relation $\frown$, we write $\asymp$ for the dual defined by $i \asymp i' \Leftrightarrow (i = i' \vee \neg(i \frown i'))$. Finite coherence spaces are those coherence spaces whose webs are finite. A *linear map* of coherence spaces $f : I \to J$ is a relation $f \subseteq \|I\| \times \|J\|$ such that, whenever $(i,j) \in f$ and $(i',j') \in f$, we have

$$ i \frown_I i' \quad \Rightarrow \quad j \frown_J j' \qquad \text{and} \qquad j \asymp_J j' \quad \Rightarrow \quad i \asymp_I i' $$

Note that these are the *converse implications* of those stated above for BRTTs.

The diagonal $\{(i,i) \mid i \in \|I\|\}$ is a linear map $I \to I$ and the relational composition of two linear maps $I \to J \to K$ is again a linear map, so that we have a category FinCoh whose objects are coherence spaces and morphisms are linear maps.

FinCoh, equipped with the tensorial product

$$ (\|I\|, \frown_I) \otimes (\|J\|, \frown_J) = (\|I\| \times \|J\|, \frown_I \times \frown_J) $$

and dualizing object $(1, 1 \times 1)$, is a well-studied $*$-autonomous category with cartesian products and coproducts. The latter may be defined pointwise as

$$ (\|I\|, \frown_I) \oplus (\|J\|, \frown_J) = (\|I\| + \|J\|, \ \frown_{I \oplus J}) $$

where $\frown_{I \oplus J}$ is the *smallest* relation such that

$$ \mathsf{in}_1(i) \frown_{I \oplus J} \mathsf{in}_1(i') \text{ when } i \frown_I i' \qquad \text{and} \qquad \mathsf{in}_2(j) \frown_{I \oplus J} \mathsf{in}_2(j') \text{ when } j \frown_J j' $$

Dualizing an object corresponds to moving from $\frown$ to $\asymp$, i.e. $(\|I\|, \frown_I)^\perp = (\|I\|, \asymp_I)$, and the product is $I \mathbin{\&} J = (I^\perp \oplus J^\perp)^\perp$.

With this in mind, we can describe how to turn a multicategory into an affine monoidal category where monoidal products may be indexed by coherence spaces. The construction has a vague family resemblance with the *coherence completion of categories* introduced by Hu and Joyal [HJ99], but appears to have quite different properties.

Definition 5.22. Let $\mathcal{M}$ be a weak symmetric multicategory. We define $\mathcal{M}_{\mathrm{coh}}$ to be the category

- whose objects are pairs $(X, (R_x)_{x \in \|X\|})$ where X is a finite coherence space and $(R_x)_{x \in \|X\|}$ a family of objects of $\mathcal{M}$. We suggestively write them $\bigodot_{x \in X} R_x$.
- whose morphisms

$$ (f, (\alpha_y)_{y \in \|Y\|}) \in \mathsf{Hom}_{\mathcal{M}_{\mathrm{coh}}}\left(\bigodot_{x \in X} R_x, \bigodot_{y \in Y} S_y \right) $$

  are pairs consisting of a linear map $f \in \mathsf{Hom}_{\mathsf{FinCoh}}(Y, X)$ and a family of multimorphisms $\alpha_y \in \mathsf{Hom}_{\mathcal{M}}\left((R_x)_{x \in f(y)}, S_y \right)$.
- whose identities are pairs $(\mathrm{id}_X, (\mathrm{id}_{R_x})_{x \in \|X\|})$.
- where the composition of

$$ (f, (\alpha_y)_{y \in \|Y\|}) \in \mathsf{Hom}\left(\bigodot_{x \in X} R_x, \bigodot_{y \in Y} S_y \right) \quad \text{and} \quad (g, (\beta_z)_{z \in \|Z\|}) \in \mathsf{Hom}\left(\bigodot_{y \in Y} S_y, \bigodot_{z \in Z} T_z \right) $$

  is $(f \circ g, (\beta_z * (\alpha_y)_{y \in g(z)})_{z \in Z})$.

Definition 5.23. We set $\mathcal{TR}^{\succ} = (\mathcal{TR}^{\mathrm{m}})_{\mathrm{coh}}$ and $\mathcal{TR}^{\succ,\leq 1}$ to be its full subcategory consisting of objects $\bigodot_{x \in X} A_x$ where each A_x is either empty or a singleton (so that $\mathcal{TR}^{\succ,\leq 1}$ is isomorphic to $(\mathcal{TR}^{\mathrm{m},\leq 1})_{\mathrm{coh}}$).

Proposition 5.24. *BRTTs over the restricted tree streaming setting $\mathfrak{TR}^{\succ,\leq 1}$ compute exactly the regular tree functions.*

Proof. By virtue of being equivalent to Alur and D'Antoni's notion of single-use-restricted BRTT [AD17]. We point the reader to Appendix A for a self-contained definition of those not involving categories, and leave it as an exercise to formally match those two descriptions. Although [AD17] and our Appendix A only consider BRTTs over *binary* trees, the proof of equivalence between the latter and regular tree functions goes through *macro tree transducers* (with regular look-ahead and single use restriction) which are known to compute regular functions for trees over arbitrary ranked alphabets [EM99], so everything can be made to work out with arbitrary arities in the end. $\qquad\square$

This being done, the remainder of this section does not depend on $\mathcal{TR}^{\mathrm{m}}$; the arguments apply to any weak symmetric multicategory $\mathcal{M}$ and designated object $\bot\!\!\!\bot \in \mathsf{Obj}(\mathcal{M})$.

Accordingly, fix such an $\mathcal{M}$ and a $\bot\!\!\!\bot$ for the remainder of this section. Fix also a set O and a map $(\!|-|\!) : \mathsf{Hom}_{\mathcal{M}}((\cdot)_\varnothing, \bot\!\!\!\bot) \to O$.

Proposition 5.25. $\mathcal{M}_{\mathrm{coh}}$ *has a terminal object, given by the unique family over the empty coherence space, and can be equipped with a symmetric monoidal affine structure $(\otimes, \top)$ where*

$$\left(\bigodot_{i \in I} A_i \right) \otimes \left(\bigodot_{j \in J} B_j \right) \;=\; \bigodot_{x \in I \& J} \left\{ \begin{array}{ll} A_i & \textit{if } x = \mathsf{in}_1(i) \\ B_j & \textit{if } x = \mathsf{in}_2(j) \end{array} \right.$$

and $I \& J$ designates the cartesian product in FinCoh.

Proof. Left to the reader. Strictly speaking, later developments will depend on the precise structure itself and not merely on its existence, but there is a single sensible choice of bifunctorial action and structural morphisms making the above a monoidal product. $\qquad\square$

Remark 5.26. To start making sense of the use of the cartesian product of FinCoh, there is a useful analogy with $\mathcal{M}_{\mathrm{aff}}$ here. There is a projection functor $\mathcal{M}_{\mathrm{aff}} \to \mathsf{PartFinSet}$ where $\mathsf{PartFinSet}$ is the category of finite sets and partial functions. The tensorial product of $\mathcal{M}_{\mathrm{aff}}$ required a coproduct at the level of indices. Here, we have a projection functor $\mathcal{M}_{\mathrm{coh}} \to \mathsf{FinCoh}^{\mathrm{op}}$, and we again use a coproduct at the level of indices (which becomes a product due to the contravariance).

We call $\mathfrak{M}_{\mathrm{aff}}$ the tree streaming setting based on $\mathcal{M}_{\mathrm{aff}}$, $\bot\!\!\!\bot$ and $(\!|-|\!)$), and $\mathfrak{M}_{\mathrm{coh}}$ the corresponding tree streaming setting based on $\mathcal{M}_{\mathrm{coh}}$.

Proposition 5.27. *There is a full and faithful strong monoidal functor $\mathcal{M}_{\mathrm{aff}} \to \mathcal{M}_{\mathrm{coh}}$ extending to a morphism of streaming setting $\mathfrak{M}_{\mathrm{aff}} \to \mathfrak{M}_{\mathrm{coh}}$.*

Proof. Call F this functor, and, for any set I, write Δ for the functor $\mathsf{PartFinSet} \to \mathsf{FinCoh}$ taking a set I to the discrete coherence space $\Delta(I) = (I, \{(i,i) \mid i \in I\})$. Note that we have $\Delta(I)^\bot = (I, I \times I)$, which may be regarded as the codiscrete coherence space generated by I. On objects of $\mathcal{M}_{\mathrm{aff}}$, we define F as

$$F\left(\bigotimes_{i \in I} A_i \right) = \bigodot_{i \in \Delta(I)^\bot} A_i$$

For morphisms $(f, (\alpha_j)_{j \in J}) \in \mathsf{Hom}_{\mathcal{M}_{\mathrm{aff}}} \left(\bigotimes_{i \in I} A_i, \bigotimes_{j \in J} B_j \right)$, we set

$$F(f, (\alpha_j)_{j \in J}) = (\{(j, i) \mid j = f(i)\}, (\alpha_j)_{j \in J})$$

It is rather straightforward to check that F is indeed full, faithful and strong monoidal, and the extension to a morphism $\mathfrak{M}_{\mathrm{aff}} \to \mathfrak{M}_{\mathrm{coh}}$ is immediate. $\qquad \square$

Proposition 5.28. $\mathcal{M}_{\mathrm{coh}}$ *also has cartesian products, which may be defined as*

$$\left(\bigodot_{i \in I} A_i \right) \& \left(\bigodot_{j \in J} B_j \right) \quad = \quad \bigodot_{x \in I \oplus J} \left\{ \begin{array}{ll} A_i & \textit{if } x = \mathsf{in}_1(i) \\ B_j & \textit{if } x = \mathsf{in}_2(j) \end{array} \right.$$

(The proof is left to the reader.) Therefore, we can extend Proposition 5.27:

Corollary 5.29. *There is a functor $E : (\mathcal{M}_{\mathrm{aff}})_\& \to \mathcal{M}_{\mathrm{coh}}$ that is full, faithful and lax (but not strong) monoidal, extending to a morphism of streaming settings $(\mathfrak{M}_{\mathrm{aff}})_\& \to \mathfrak{M}_{\mathrm{coh}}$.*

In the following proof and the rest of this section, we write explicitly $\mathbf{I}_{\mathrm{coh}}$ for the monoidal unit of $\otimes$ in $\mathcal{M}_{\mathrm{coh}}$ and $\mathbf{I}_{\mathrm{aff}\&}$ for the unit in $(\mathcal{M}_{\mathrm{aff}})_\&$.

Proof idea. The universal property of the free product completion defines E as the unique product-preserving functor extending the functor of Proposition 5.27. It remains to equip it with a lax monoidal functor structure. The map $m^0 : \mathbf{I}_{\mathrm{coh}} \to E(\mathbf{I}_{\mathrm{aff}\&})$ is an obvious isomorphism, while the natural transformation $m^2_{A,B} : E(A) \otimes E(B) \to E(A \otimes B)$ can be obtained via the canonical map

$$\left(\underset{i \in I}{\&} A_i \right) \otimes \left(\underset{j \in J}{\&} B_j \right) \to \underset{(i,j) \in I \times J}{\&} A_i \otimes B_j$$

in $\mathcal{M}_{\mathrm{coh}}$ (it exists in all monoidal categories with products). $\qquad \square$

We can now go the other way around.

Lemma 5.30. *There is a strong monoidal functor $\mathcal{M}_{\mathrm{coh}} \to (\mathcal{M}_{\mathrm{aff}})_\&$, which extends to a morphism of streaming settings $\mathfrak{M}_{\mathrm{coh}} \to (\mathfrak{M}_{\mathrm{aff}})_\&$.*

Proof. For a coherence space $(\|X\|, \frown_X)$, write $\mathrm{Cl}(X) \subseteq \mathcal{P}(X)$ the set of *cliques* of X

$$\mathrm{Cl}(X) \quad = \quad \{S \in \mathcal{P}(X) \mid \forall x\, y \in S.\ x \frown_X y\}$$

We now define the functor $F : \mathcal{M}_{\mathrm{coh}} \to (\mathcal{M}_{\mathrm{aff}})_\&$ on objects as

$$F \left(\bigodot_{x \in X} A_x \right) \quad = \quad \underset{S \in \mathrm{Cl}(X)}{\&} \bigotimes_{x \in S} A_x$$

As for morphisms, first recall that a morphism $(R, \alpha) \in \mathsf{Hom}_{\mathcal{M}_{\mathrm{coh}}} \left(\bigodot_{x \in X} A_x, \bigodot_{y \in Y} B_y \right)$ consists of a linear map $R \in \mathsf{Hom}_{\mathsf{FinCoh}}(Y, X)$ and a family

$$(\alpha_y)_{y \in \|Y\|} \in \prod_{y \in \|Y\|} \mathsf{Hom}_{\mathcal{M}} \left((A_x)_{(y,x) \in R}, B_y \right)$$

We set out to define

$$F(R, \alpha) \in \mathsf{Hom}_{(\mathcal{M}_{\mathrm{aff}})_\&} \left(F \left(\bigotimes_{x \in S} A_x \right), F \left(\bigotimes_{y \in S'} B_y \right) \right)$$

$$
\begin{aligned}
F\left(\underset{x\in X}{\bigodot} A_x\right) \otimes F\left(\underset{y\in Y}{\bigodot} B_y\right) \quad &\cong\quad \left(\underset{S\in\mathrm{Cl}(X)}{\Large\&}\ \underset{x\in S}{\bigotimes} A_x\right) \otimes \left(\underset{S'\in\mathrm{Cl}(Y)}{\Large\&}\ \underset{y\in S'}{\bigotimes} B_y\right) \\[2mm]
&\cong\quad \underset{\substack{S\in\mathrm{Cl}(X)\\ S'\in\mathrm{Cl}(Y)}}{\Large\&}\ \left(\underset{x\in X}{\bigotimes} A_x\right)\otimes\left(\underset{y\in Y}{\bigotimes} B_y\right) \\[2mm]
&\cong\quad \underset{S\in\mathrm{Cl}(X\&Y)}{\Large\&}\ \underset{z\in S}{\bigotimes} C_z \\[2mm]
&\cong\quad F\left(\underset{z\in X\&Y}{\bigodot} C_z\right) \\[2mm]
&\cong\quad F\left(\left(\underset{x\in X}{\bigodot} A_x\right)\otimes\left(\underset{y\in Y}{\bigodot} B_y\right)\right)
\end{aligned}
$$

Figure 15: The main structural natural isomorphism making the functor $\mathcal{M}_{\mathrm{coh}} \to (\mathcal{M}_{\mathrm{aff}})_\&$ strong monoidal, writing A_x for $C_{\mathsf{in}_1(x)}$ and B_y for $C_{\mathsf{in}_2(y)}$.

recalling that

$$
\mathsf{Hom}_{(\mathcal{M}_{\mathrm{aff}})_\&}\left(F\left(\bigotimes_{x\in S} A_x\right), F\left(\bigotimes_{y\in S'} B_y\right)\right)
$$

$$
= \prod_{S'\in\mathrm{Cl}(Y)}\ \sum_{S\in\mathrm{Cl}(X)}\ \sum_{f:S\rightharpoonup S'}\ \prod_{y\in S'} \mathsf{Hom}_{\mathcal{M}}\left((A_x)_{x\in f^{-1}(x)}, B_y\right)
$$

So fixing $S' \in \mathrm{Cl}(Y)$ and recall that R being linear means that we have

$$
(y,x)\in R \wedge (y',x)\in R \qquad \Rightarrow \qquad \left\{\begin{array}{lll} y \frown_Y y' & \Rightarrow & x \frown_X x' \quad (1) \\ x = x' & \Rightarrow & y \asymp_Y y' \quad (2) \end{array}\right.
$$

In particular, (1) implies that $\{x \in \|X\| \mid (y,x) \in R\}$ is a clique; we take that to be S. (2), and the fact that $y \frown_Y y' \wedge y \asymp_Y y' \Rightarrow y = y'$, imply that R determines a (total) function $S \rightharpoonup S'$, which we take to be f. Finally, once $y \in S'$ is fixed, we pick the component α_y to complete the definition, which makes sense as $f^{-1}(y) = \{x \in S \mid (y,x) \in R\} = \{x \in \|X\| \mid (y,x) \in R)\}$. This completes the definition of $F(R,\alpha)$; we leave checking functoriality to the reader.

Now, we turn to defining a morphism of streaming setting $\mathfrak{M}_{\mathrm{coh}} \to (\mathfrak{M}_{\mathrm{aff}})_\&$ from F. To this end, we must first equip F with a lax monoidal structure, that is to define $(\mathcal{M}_{\mathrm{aff}})_\&$-maps

$$
m^0 : \mathbf{I}_{\mathrm{aff}\&} \to F(\mathbf{I}_{\mathrm{coh}}) \quad m^2 : F\left(\underset{x\in X}{\bigodot} A_x\right) \otimes F\left(\underset{y\in Y}{\bigodot} B_y\right) \to F\left(\left[\underset{x\in X}{\bigodot} A_x\right] \otimes \left[\underset{y\in Y}{\bigodot} B_y\right]\right)
$$

satisfying the relevant coherence diagrams. We do not check them here, but indicate how to build those two maps. m^0 arises as an obvious isomorphism $F(\mathbf{I}_{\mathrm{coh}}) \cong \mathbf{I}_{\mathrm{aff}}$. m^2 is also an isomorphism, which may be computed as per Figure 15. Finally, there is a

canonical isomorphism $F(\bot\!\!\!\bot) \cong \bot\!\!\!\bot$ which allows to complete the definition of the morphism $\mathfrak{M}_{\mathrm{coh}} \to (\mathfrak{M}_{\mathrm{aff}})_{\&}$. $\qquad\square$

We thus conclude this section by first specializing the above to the case $\mathcal{M} = \mathcal{TR}^{\leq 1}$, and then making a final tangential observation.

Lemma 5.31. *There are morphisms of streaming settings* $\mathfrak{TR}^{\leq 1}_{\overline{\&}} \to \mathfrak{TR}^{\succ, \leq 1} \to \mathfrak{TR}^{\leq 1}_{\overline{\&}}$. *In particular,* $\mathfrak{TR}^{\leq 1}_{\overline{\&}}$*-BRTTs compute exactly the regular functions.*

Remark 5.32. One idea that one could take from Hu and Joyal's coherence completion [HJ99] – but that we do not explore further here – is to look at objects whose indexing coherence spaces are (up to isomorphism) generated from singletons by the $\&/\oplus$ operations of FinCoh. (Those are called "contractible" in [HJ99, Section 4], and considering coherence spaces as undirected graphs, this corresponds to the classical notion of *cograph* in combinatorics.)

In the case of the coherence completion of some category $\mathcal{C}$, the full subcategory spanned by such objects turns out to be the free completion of $\mathcal{C}$ under finite products and coproducts (which differs from our $(-)_{\oplus\&}$ in not making '$\&$' distribute over '$\oplus$'); this is formalized as a universal property in [HJ99, Theorem 4.3]. In the same vein, we conjecture that the full subcategory of $\mathcal{M}_{\mathrm{coh}}$ consisting of cograph-indexed objects – that is, of objects that are generated from those of $\mathcal{M}$ by means of the operations $\otimes/\&$ in $\mathcal{M}_{\mathrm{coh}}$ – is in some way the *free affine symmetric monoidal category with products generated by the multicategory* $\mathcal{M}$.

5.5. $\mathcal{TR}_{\oplus\&}$ is monoidal closed.

Now, we consider the category $\mathcal{TR}_{\oplus\&}$ in the context of trees. Much like with strings, this category is symmetric monoidal monoidal closed with finite products and coproducts, which makes it an ideal target to compile $\lambda\ell^{\oplus\&}$-terms. This structure over $\mathcal{TR}_{\oplus\&}$ is obtained in the same way as for strings: the monoidal product over $\mathcal{TR}$ is defined as distributing over formal sums and products and the usual products and coproducts are created by the $(-)_{\oplus\&}$ completion. Similarly, monoidal closure can be obtained in a generic way once we show that the objects coming from $\mathcal{TR}$ have internal homsets $\mathcal{TR}_{\oplus}$ (echoing Lemma 3.38). This section is mostly dedicated to proving this fact, whose proof relies on decomposing linear trees in a similar way as in Lemma 5.16.

Lemma 5.33. *For any two objects* $\mathbf{R}$ *and* $\mathbf{S}$ *of* $\mathcal{TR}$, *there is an internal hom* $\iota_{\oplus}(\mathbf{R}) \multimap \iota_{\oplus}(\mathbf{S})$ *in* $\mathcal{TR}_{\oplus}$.

Proof. First, we treat the special case where $\mathbf{S} = \mathcal{I}(U)$ for some finite set U. To make sense of the definition of $\iota_{\oplus}(\mathbf{R}) \multimap \iota_{\oplus}(\mathcal{I}(U))$ it is helpful to notice that it will ultimately induce an isomorphism

$$\mathsf{Hom}_{\mathcal{TR}_{\oplus}}(\top, \iota_{\oplus}(\mathbf{R}) \multimap \iota_{\oplus}(\mathcal{I}(U))) \cong \mathsf{Hom}_{\mathcal{TR}}(\mathbf{R}, \mathcal{I}(U)) \cong \mathbf{LTree}_{\Gamma}(\mathbf{R} \otimes \mathcal{O}(U))$$

so, recalling that objects of $\mathcal{TR}_{\oplus}$ are of the shape $\bigoplus_{i \in I} \bigotimes_{j \in J_i} \mathcal{I}(V_j)$ for V_j being finite sets, the operational intuition is that one may code trees with "holes with arity" into some bounded finitary data (which we may informally call a partitioning tree) plus finitely many trees containing holes "without arity"; this bijection is pictured in Figure 16. As with Lemma 5.16, we will not use this as our official definition for the internal homset, but rather use the following recursive definition:

- If $\mathbf{R} = \top$, set $\iota_{\oplus}(\mathbf{R}) \multimap \iota_{\oplus}(\mathcal{I}(U)) = \iota_{\oplus}(\mathcal{I}(U))$.
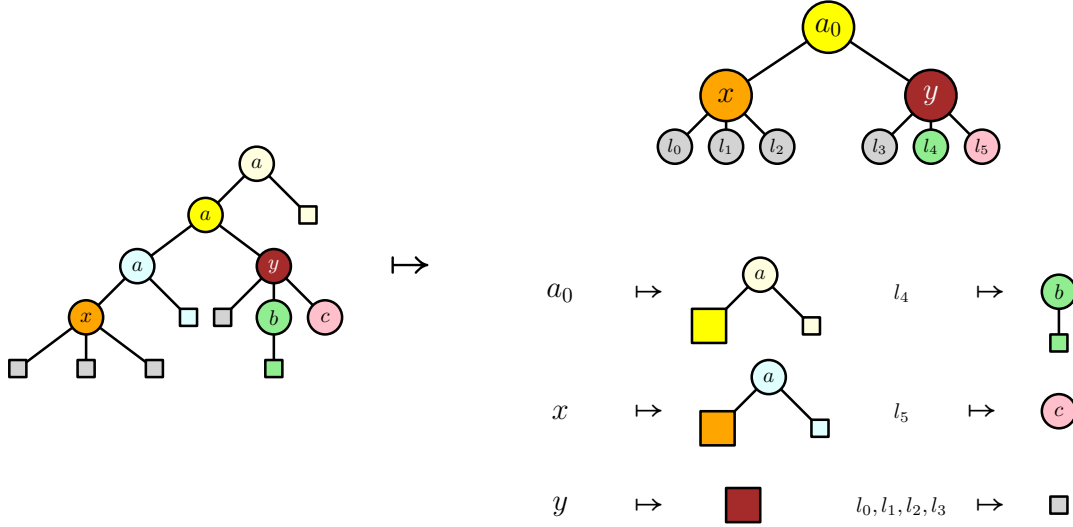
Figure 16: Decomposition of a map of $\mathsf{Hom}_{\mathcal{TR}(\{a:2,b:1,c:0\})}(\{x:3,y:3\},\mathcal{I}(7))$ (which is defined as $\mathbf{LTree}_{\{a:2,b:1,c:0\}}(\{x:3,y:3\}\otimes\mathcal{O}(7)))$ as a tuple consisting of a partitioning tree and trees without the letters x and y.

- Otherwise, define $\iota_\oplus(\mathbf{R}) \multimap \iota_\oplus(\mathcal{I}(U))$ as

$$\bigoplus_{U=V\uplus W} \iota_\oplus(\mathcal{I}(V+1)) \otimes \left(\bigoplus_{b\in\mathbf{\Gamma}} (\mathbf{R} \multimap_b \mathcal{I}(W)) \oplus \bigoplus_{r\in R} (\mathbf{R} \multimap_r \mathcal{I}(W)) \right)$$

where $\mathbf{R} \multimap_r \mathcal{I}(W)$ and $\mathbf{R} \multimap_b \mathcal{I}(W)$ are auxiliary definitions which correspond to the following situations (recalling that morphisms can be regarded as trees):

- $\iota_\oplus(\mathcal{I}(V+1)) \otimes (\mathbf{R} \multimap_r \mathcal{I}(W))$ correspond to morphisms such that there is a *unique* minimal path leading from the root to a node labelled by a letter in R, and that letter is r. The second component $\mathbf{R} \multimap_r \mathcal{I}(W)$ is meant to include the immediate subtrees of that node while the first $\iota_\oplus(\mathcal{I}(V+1))$ contains the tree where a nullary node labeled is inserted instead of that node. The combination of both these data and r allows to recover the original morphism.
- $\bigoplus_{b\in\mathbf{\Gamma}} \iota_\oplus(\mathcal{I}(V+1)) \otimes \mathbf{R} \multimap_b \mathcal{I}(W)$ correspond to all the other morphisms. In such a case there is a topmost node labelled by some letter b of $\mathbf{\Gamma}$ with which has at least two distinct immediate subtrees which have at least one node of R each. Similarly to the first subcase, $\mathbf{R} \multimap_b \mathcal{I}(W)$ is intended to include the immediate subtrees of that node labeled by b while $\iota_\oplus(\mathcal{I}(V+1))$ contains the tree where a nullary node labeled is inserted instead of that node. The combination of these data and b allows to recover the original morphism.

Their formal definition is as follows:

$$\mathbf{R} \multimap_b \mathcal{I}(W) \;=\; \bigoplus_{\substack{f:W\to\mathrm{ar}(b)\\ g:R\to\mathrm{ar}(b)\\ g \text{ nonconstant}}} \bigotimes_{x\in\mathrm{ar}(\Gamma)} \iota_\oplus(\mathbf{R}\upharpoonright g^{-1}(x)) \multimap \iota_\oplus(f^{-1}(x))$$

$$\mathbf{R} \multimap_r \mathcal{I}(W) \;=\; \bigoplus_{\substack{f:W\to\mathrm{ar}(r)\\ g:R\setminus\{r\}\to\mathrm{ar}(r)}} \bigotimes_{x\in\mathrm{ar}(r)} \iota_\oplus(\mathbf{R}\upharpoonright g^{-1}(x)) \multimap \iota_\oplus(f^{-1}(x))$$

Note that the definitions of $\iota_\oplus(\mathbf{R}) \multimap \iota_\oplus(\mathcal{I}(U))$, $\mathbf{R} \multimap_b \mathcal{I}(W)$ and $\mathbf{R} \multimap_r \mathcal{I}(W)$ mutually depend on one another. Still this is is well-defined as the definitions of $\mathbf{R} \multimap_b \mathcal{I}(W)$ and $\mathbf{R} \multimap_r \mathcal{I}(W)$ only require $\iota_\oplus(\mathbf{S}) \multimap \iota_\oplus(\mathcal{I}(V))$ for $\mathbf{S}$ strictly smaller than $\mathbf{R}$.

We now describe the associated evaluation map

$$\mathrm{ev}_{\mathbf{R},\mathcal{I}(U)} \;:\; (\iota_\oplus(\mathbf{R}) \multimap \iota_\oplus(\mathcal{I}(U))) \otimes \mathbf{R} \;\longrightarrow\; \mathcal{I}(U)$$

also by recursion over $\mathbf{R}$.

- If $\mathbf{R} = \top$, it is the identity.
- Otherwise, we need to provide maps

$$\left(\mathcal{I}(V+1) \otimes \left(\bigoplus_{b\in\Gamma} (\mathbf{R}\multimap_b\mathcal{I}(W)) \oplus \bigoplus_{r\in R}(\mathbf{R}\multimap_r\mathcal{I}(W))\right)\right) \otimes \mathbf{R} \;\longrightarrow\; \mathcal{I}(U)$$

for every decomposition $U = V \uplus W$, the intuition being that $\mathcal{I}(V+1)$ is a context containing the top of the tree corresponding to the function we want to apply. Therefore, once we provide a map

$$\left(\bigoplus_{b\in\Gamma} (\mathbf{R}\multimap_b\mathcal{I}(W)) \oplus \bigoplus_{r\in R}(\mathbf{R}\multimap_r\mathcal{I}(W))\right) \otimes \mathbf{R} \;\longrightarrow\; \mathcal{I}(W)$$

we may post-compose it with $\mathcal{I}(V+1)\otimes\mathcal{I}(W) \to \mathcal{I}(V+W) \cong \mathcal{I}(V\uplus W) = \mathcal{I}(U)$ to define $\mathrm{ev}_{\mathbf{R}\multimap\mathcal{I}(U)}$. Recalling that $\otimes$ distributes over $\oplus$, it suffices to provide specialized maps

$$\mathrm{ev}^b_{\mathbf{R},\mathcal{I}(W)} : \mathbf{R} \multimap_b \mathcal{I}(W) \otimes \mathbf{R} \to \mathcal{I}(W) \quad \text{and} \quad \mathrm{ev}^r_{\mathbf{R},\mathcal{I}(W)} : \mathbf{R} \multimap_r \mathcal{I}(W) \otimes \mathbf{R} \to \mathcal{I}(W)$$

for $b \in \Gamma$ and $r \in R$, which we describe now.

- For $\mathrm{ev}^b_{\mathbf{R},\mathcal{I}(W)}$, it suffices to define a family of $\mathcal{TR}$-maps indexed by $f : W \to \mathrm{ar}(b)$ and $g : R \to \mathrm{ar}(b)$ with g non-constant

$$\left(\bigotimes_{x\in\mathrm{ar}(\Gamma)} (\mathbf{R}\upharpoonright g^{-1}(x)) \multimap \mathcal{I}(f^{-1}(x))\right) \otimes \mathbf{R} \;\longrightarrow\; \mathcal{I}(W)$$

Recall that b can be seen as tree constructor and induces a canonical map

$$\bar{b} \;:\; \bigotimes_{x\in\mathrm{ar}(b)} \mathcal{I}(f^{-1}(x)) \;\longrightarrow\; \mathcal{I}(W)$$

Using the induction hypothesis, we have evaluation maps

$$((\mathbf{R}\upharpoonright g^{-1}(x)) \multimap \mathcal{I}(f^{-1}(x))) \otimes (\mathbf{R}\upharpoonright g^{-1}(x)) \;\longrightarrow\; \mathcal{I}(W)$$

We can then compose $\bar{b}$ with the product of those maps over $x \in \mathrm{ar}(b)$ and then the isomorphism $\mathbf{R} \cong \bigotimes_{x\in\mathrm{ar}(b)} \mathbf{R} \upharpoonright f^{-1}(x)$ to conclude the definition of $\mathrm{ev}^b_{\mathbf{R},\mathcal{I}(W)}$.

– For $\mathrm{ev}^r_{\mathbf{R},\mathcal{I}(W)}$, it suffices to define a family of $\mathcal{TR}$-maps indexed by $f : W \to \mathrm{ar}(r)$ and $g : R \setminus \{r\} \to \mathrm{ar}(r)$

$$\left(\bigotimes_{x \in \mathrm{ar}(\Gamma)} (\mathbf{R} \upharpoonright g^{-1}(x)) \multimap \mathcal{I}(f^{-1}(x)) \right) \otimes \mathbf{R} \quad \longrightarrow \quad \mathcal{I}(W)$$

By exploiting the isomorphism $\mathbf{R} \cong \mathcal{I}(\mathrm{ar}(r)) \otimes \bigotimes_{x \in \mathrm{ar}(r)}(\mathbf{R} \upharpoonright g^{-1}(x))$ and using the inductive hypothesis as in the previous case, we obtain a map

$$\left(\bigotimes_{x \in \mathrm{ar}(\Gamma)} (\mathbf{R} \upharpoonright g^{-1}(x)) \multimap \mathcal{I}(f^{-1}(x)) \right) \otimes \mathbf{R} \quad \longrightarrow \quad \mathcal{I}(\mathrm{ar}(r)) \otimes \bigotimes_{x \in \mathrm{ar}(r)} \mathcal{I}(f^{-1}(x))$$

and we may conclude by post-composing by the map

$$\mathcal{I}(\mathrm{ar}(r)) \otimes \bigotimes_{x \in \mathrm{ar}(r)} \mathcal{I}(f^{-1}(x)) \to \mathcal{I}(W)$$

which is induced by the depth-2 tree whose root corresponds to the first component, whose children correspond to the successive elements of $\bigotimes_{x \in \mathrm{ar}(r)} \mathcal{I}(f^{-1}(x))$ and other leaves are in W.

While the definition is a bit wordy, there is then little difficulty in checking that this yields the expected universal property.

$$\begin{array}{ccc}
\iota_\oplus(\mathbf{R} \multimap \mathcal{I}(U)) \otimes \iota_\oplus(\mathcal{I}(U)) & \longrightarrow & \iota_\oplus(\mathcal{I}(U)) \\
\Lambda(h)\otimes\mathrm{id} \uparrow & \nearrow & \\
A \otimes \iota_\oplus(\mathbf{R}) & h &
\end{array}$$

One needs then to extend the definition for the general case where $\mathbf{S}$ is not necessarily $\mathcal{I}(U)$; this is done using a similar approach as for strings, by using a coproduct over partial maps $R \rightharpoonup S$ tracking which letter of the input participates in which letter of the output, and employing the particular case where there is one letter in the output[27]

$$\iota_\oplus(\mathbf{R}) \multimap \iota_\oplus(\mathbf{S}) \quad = \quad \bigoplus_{f:R\rightharpoonup S} \bigotimes_{s \in S} \iota_\oplus(\mathbf{R} \upharpoonright f^{-1}(s)) \multimap \iota_\oplus(\mathcal{I}(\mathrm{ar}(s)))$$

The evaluation function can then be extended and the universal property accordingly lifted to the more general case. $\qquad\square$

The above lemma tells us that $\mathcal{TR}_{\oplus\&}$ satisfies the premises of Theorem 3.53, hence:

Theorem 5.34. *The category $\mathcal{TR}_{\oplus\&}$ has cartesian products, coproducts and a symmetric monoidal* closed *structure.*

Remark 5.35. Given the close relationship between the constructions $((-)_{\mathrm{aff}})_{\&}$ and $(-)_{\mathrm{coh}}$, it seems plausible that $(\mathcal{TR}_{\mathrm{coh}})_\oplus$ could be monoidal closed (we know that it has products, coproducts and a symmetric monoidal structure). We leave this question to further work.

---

[27]This could be factored out as a more general result concerning the existence of internal homsets in categories of the shape $\mathcal{M}_{\mathrm{aff}}$ for multicategories $\mathcal{M}$.

In other words, $\mathcal{TR}_{\oplus\&}$ provides a categorical semantics for the purely linear fragment of the $\lambda\ell^{\oplus\&}$-calculus. Concretely, by a suitable adaptation of Lemma 3.26 to tree streaming settings, we finally have:

Corollary 5.36. *There is a morphism of streaming settings $\mathfrak{L} \to \mathfrak{TR}_{\oplus\&}$.*

5.6. **Preservation properties of finite completions.** We have now proven all of the combinatorial results leading to the main theorem for trees. At this stage, we only need to make explicit a few preservation properties of the completions $(-)_\oplus$ and $(-)_\&$. We only state the minimal requirements that we need to proceed[28].

Lemma 5.37. *Let $\square \in \{\oplus, \&, \oplus\&\}$ and $\mathfrak{C}, \mathfrak{D}$ be streaming settings. Then if there is a morphism $\mathfrak{C} \to \mathfrak{D}$, there is also a morphism $\mathfrak{C}_\square \to \mathfrak{C}_\square$.*

Proof idea. Let us only discuss the case $\square = \oplus$ and assume that we have a streaming setting morphism whose underlying lax monoidal functor is $F : \mathcal{C} \to \mathcal{D}$. Then we consider $F_\oplus : \mathcal{C}_\oplus \to \mathcal{D}_\oplus$ defined using the universal property of $\mathcal{C}_\oplus$ from the functor $\iota_\oplus \circ F : \mathcal{C} \to \mathcal{D}_\oplus$ such that

$$F_\oplus \left(\bigoplus_{u \in U} \iota_\oplus(C_u) \right) = \bigoplus_{u \in U} \iota_\oplus(F(C_u))$$

so that $F_\oplus$ inherits a lax monoidal structure from F by lifting the relevant maps functorially. For instance, we have $\iota_\oplus(\mathbf{I}) \to \iota_\oplus(F(\mathbf{I})) = F_\oplus(\mathbf{I})$ for the unit. We leave checking the coherence diagram, the case of the binary tensor to the reader, as well as checking that the rest of the structure of morphisms of streaming settings lift accordingly. $\square$

Lemma 5.38. *Let $\mathfrak{C}$ be a streaming setting. Then there are morphisms of streaming settings $\mathfrak{C}_{\oplus\&} \to (\mathfrak{C}_\oplus)_{\oplus\&}$ and $(\mathfrak{C}_\oplus)_{\oplus\&} \to \mathfrak{C}_{\oplus\&}$.*

Proof idea. The first morphism can be obtained using Lemma 5.37 with the morphism of streaming setting $\mathfrak{C} \to \mathfrak{C}_\oplus$ corresponding to $\iota_\oplus$. The second morphism can be written as a composite

$$(\mathfrak{C}_\oplus)_{\oplus\&} \cong ((\mathfrak{C}_\oplus)_\&)_\oplus \xrightarrow{(\mathfrak{Dist}_{\mathfrak{C}})_\oplus} ((\mathfrak{C}_\&)_\oplus)_\oplus \xrightarrow{\mathfrak{Mu}_{\mathfrak{C}_\&}} (\mathfrak{C}_\&)_\oplus \cong \mathfrak{C}_{\oplus\&}$$

where $(\mathfrak{Dist}_{\mathfrak{C}})_\oplus$ is obtained from a morphism $\mathfrak{Dist}_{\mathfrak{C}} : (\mathfrak{C}_\oplus)_\& \to (\mathfrak{C}_\&)_\oplus$ by Lemma 5.37. $\mathfrak{Dist}_{\mathfrak{C}}$ itself is built from a functor $D_{\mathcal{C}} : (\mathcal{C}_\oplus)_\& \to (\mathcal{C}_\&)_\oplus$ defined on object following the mantra "products distribute over coproducts".

$$D_{\mathcal{C}} \left(\underset{x \in X}{\&} \iota_\& \left(\bigoplus_{u \in U_x} \iota_\oplus(C_{x,u}) \right) \right) = \bigoplus_{F \in \prod_x U_x} \iota_\oplus \left(\underset{x \in X}{\&} \iota_\&(C_{x,u}) \right)$$

On the other hand, the functor $M : \mathcal{C}$ which is part of $\mathfrak{Mu}_{\mathfrak{C}}$ is obtained by using the universal property of $(-)_\oplus$ so that

$$M \left(\bigoplus_{u \in U} \iota_\oplus \left(\bigoplus_{v \in V} \iota_\oplus(C_{u,v}) \right) \right) = \bigoplus_{(u,v)} \iota_\oplus(C_{u,v})$$

$\square$

---

[28]The following, which would entail these requirements, should hold (but we have not checked it): the completions $(-)_\oplus$ and $(-)_\&$ should behave as pseudo-monads over the 2-category of tree streaming settings, admitting a pseudo-distributive law $((-)_\oplus)_\& \to ((-)_\&)_\oplus$ giving rise to the pseudo-monad $(-)_{\oplus\&}$.

5.7. **Proof of the main result on trees.** We can now give the proof of Theorem 1.2, which can be summarized as the string of equalities

$$
\begin{aligned}
\lambda\ell^{\oplus\&} \quad &= \quad \text{single-state } \mathfrak{L}\text{-BRTTs} && \text{by Lemma 5.17} \\
&= \quad \text{single-state } \mathfrak{TR}_{\oplus}^{\succ,\leq 1}\text{-BRTTs} && (\dagger) \\
&= \quad \mathfrak{TR}^{\succ,\leq 1}\text{-BRTTs} && \text{by Theorem 5.10} \\
&= \quad \text{regular tree functions} && \text{by Proposition 5.24}
\end{aligned}
$$

Note that applying Theorem 5.10 in the last step requires that objects of $\mathcal{TR}^{\succ,\leq 1}$ have unitary support, which is easily checked for output alphabets $\mathbf{\Gamma}$ containing at least one letter of arity $\varnothing$. It remains to justify step ($\dagger$), i.e., that single-state $\mathfrak{L}$-BRTTs and $\mathfrak{TR}_{\oplus}^{\succ,\leq 1}$-BRTTs are equi-expressive. By Lemma 5.6, it suffices to have morphisms of streaming settings both ways $\mathfrak{L} \leftrightarrow \mathfrak{TR}_{\oplus}^{\succ,\leq 1}$ to conclude. They may be obtained as the following composites (where we leave implicit the uses of Lemma 5.37 and some easily inferrable steps).

$$
\mathfrak{L} \quad\underset{\text{Cor. 5.19}}{\overset{\text{Cor. 5.36}}{\rightleftarrows}}\quad \mathfrak{TR}_{\oplus\&} \quad\overset{\text{Lem. 5.38}}{\rightleftarrows}\quad (\mathfrak{TR}_{\oplus})_{\oplus\&} \quad\overset{\text{Lem. 5.16}}{\rightleftarrows}\quad (\mathfrak{TR}_{\oplus}^{\leq 1})_{\oplus\&} \quad\overset{\text{Lem. 5.38}}{\rightleftarrows}\quad \mathfrak{TR}_{\oplus\&}^{\leq 1} \cong (\mathfrak{TR}_{\&}^{\leq 1})_{\oplus} \quad\overset{\text{Lem. 5.31}}{\rightleftarrows}\quad \mathfrak{TR}_{\oplus}^{\succ,\leq 1}
$$

6. Conclusion & further work

We have proven that the tree (and string) functions definable at type $\text{Tree}_{\mathbf{\Sigma}}[A] \multimap \text{Tree}_{\mathbf{\Gamma}}$ in the $\lambda\ell^{\oplus\&}$-calculus correspond exactly to regular functions. To prove the non-trivial left-to-right inclusion, we used a syntactic lemma for $\lambda\ell^{\oplus\&}$-terms allowing to compile those terms into a kind of transducer model using purely linear $\lambda\ell^{\oplus\&}$-terms as its memory ($\mathfrak{L}$-BRTTs). We then showed in a principled way that those transducers are equivalent to the more standard single-use-restricted Bottom-up Ranked Tree Transducers (expressed as $\mathfrak{TR}^{\succ,\leq 1}$-BRTTs in our framework) which capture regular tree functions. Along the way, we revisited a few results on string transducers under the light of our categorical framework, exhibiting some points of convergence with our semantics for $\lambda\ell^{\oplus\&}$-terms.

There are a number of ways one could plan on expanding this work. We list some of the most relevant problems, regarding the definable functions between strings and trees in various λ-calculi, that we expect could be tackled using similar methods or minor variations of our setting.

Removing the additive connectives. $\lambda\ell^{\oplus\&}$ features the additive connectives $\oplus$ and $\&$ (as well as the relevant units 0 and $\top$). A natural question is whether all regular functions are still encodable if we remove access to those connectives and use the λa-calculus, an *affine* version of $\lambda\ell^{\oplus\&}$ without those connectives; for strings, this is first half of [NP20, Claim 6.2].

In a sense, we went further than that in showing that $\lambda\ell^{\oplus\&}$-definable functions are regular, since it can be checked that all λa-definable string-to-string functions are also $\lambda\ell^{\oplus\&}$-definable. However, we did not give evidence for the converse direction, as our encoding of states of BRTTs use the additive connectives. We believe this can be remedied by leveraging our previous work on encoding sequential transducers [NP20, Sections 4-5] (that relies on the highly non-trivial Krohn-Rhodes theorem).

As for functions taking trees as input, we suspect that the λa-calculus is *strictly less expressive* than copyless BRTTs ($\mathfrak{TR}$-BRTTs) – in fact, that λa-terms are even unable to

recognize all regular tree *languages*. We expect that this can be shown by interpreting purely affine λa-terms into (a variant of) the geometry of interaction semantics mentioned in Example 3.7. Our intent is to deduce from this that λa-definable languages can be recognized by *tree-walking automata* – which are indeed a natural extension to trees of the two-way automata from Example 3.7 – and some regular tree languages are known to be unrecognizable by such devices [BC08].

Anecdotally, we show in Appendix F that removing the additive disjunction '$\oplus$' from the $\lambda\ell^{\oplus\&}$-calculus while keeping the additive conjunction '$\&$' does not affect the definability of tree functions. While this is a routine application of a continuation-passing-style transformation, the nice thing is that it can be entirely carried out at the level of streaming settings.

Dropping commutativity. The second half of [NP20, Claim 6.2] raised the issue of definable string-to-string transductions in the $\lambda\wp$-calculus, the restriction of the λa-calculus to a *non-commutative* multiplicative structure. By analogy with the main result of [NP20], which was that $\lambda\wp$-definable languages are star-free (i.e. first-order definable) while λa defines regular languages, the natural guess is that they correspond to *first-order regular functions*. We expect non-commutative typing to translate into a *planarity* condition in the above-mentioned category of diagrams (close to a free (non-symmetric) compact-closed category [KL80]).

The situation with trees looks more delicate, although trying to compare the expressiveness of the $\lambda\wp$-calculus with the functional combinators described in [BD20] might be a good starting point.

Duplicating the input. Going to higher complexities, but without escalating to the towers of exponentials definable in the simply-typed λ-calculus, one natural question is: how expressive are $\lambda\ell^{\oplus\&}$-terms of type $\mathrm{Tree}_{\boldsymbol{\Sigma}}[\kappa] \to \mathrm{Tree}_{\boldsymbol{\Gamma}}$, for purely linear κ? We expect that this problem could be tackled by reusing our semantic interpretation of purely linear $\lambda\ell^{\oplus\&}$-terms together with a suitable refinement of Lemma 3.24. Concerning the case of strings, our current conjecture is that the expressible functions have polynomial growth and form a strict subclass of *polyregular functions* [Boj18, BKL19].

Semantic interpretation of exponentials. One frustrating aspect of our approach is that we need to rely on the normalization of $\lambda\ell^{\oplus\&}$-terms and a special-purpose syntactic lemma to dissect the term under consideration and extract the relevant purely linear subterms before being able to go to semantic interpretations and more high-level arguments. The reason behind this is that our semantics does not interpret the non-linear arrow '$\to$' or, equivalently, a version of the exponential modality '!' of linear logic. It might be more pleasant to have more general semantic settings with those features that still allow us to show our characterizations. Furthermore, such a setting would be *necessary* if we hope to tackle the question of the expressiveness of functions $\mathrm{Tree}_{\boldsymbol{\Sigma}}[A] \to \mathrm{Tree}_{\boldsymbol{\Sigma}}$ for general A using semantic tools (this last variant of the problem is equivalent to the setting of the simply-typed λ-calculus without linearity constraints).

References

[AČ10]   Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In Kamal
             Lodaya and Meena 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. `doi:10.4230/LIPIcs.FSTTCS.2010.1`.

[AD11]   Rajeev Alur and Jyotirmoy V. Deshmukh. Nondeterministic streaming string transducers. In
             Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming -
             38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings,
             Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
             `doi:10.1007/978-3-642-22012-8_1`.

[AD17]   Rajeev Alur and Loris D'Antoni. Streaming Tree Transducers. *Journal of the ACM*, 64(5):1–55,
             August 2017. `doi:10.1145/3092842`.

[ADHS01]   Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott. Normalization by
             evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on
             Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*,
             pages 303–310. IEEE Computer Society, 2001. `doi:10.1109/LICS.2001.932506`.

[AFR14]   Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string
             transformations. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual
             Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE
             Symposium on Logic in Computer Science (LICS) - CSL-LICS '14*, pages 1–10, Vienna,
             Austria, 2014. ACM Press. `doi:10.1145/2603088.2603151`.

[AFT12]   Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular Transformations of Infinite
             Strings. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 65–74,
             Dubrovnik, Croatia, June 2012. IEEE. `doi:10.1109/LICS.2012.18`.

[Bar96]   Andrew Barber. Dual Intuitionistic Linear Logic. Technical report ECS-LFCS-96-347,
             LFCS, University of Edinburgh, 1996. URL: `http://www.lfcs.inf.ed.ac.uk/reports/96/
             ECS-LFCS-96-347/`.

[BB85]   Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed λ-programs on term al-
             gebras. *Theoretical Computer Science*, 39:135–154, January 1985. `doi:10.1016/0304-3975(85)
             90135-5`.

[BC08]   Mikołaj Bojańczyk and Thomas Colcombet. Tree-walking automata do not recognize all regular
             languages. *SIAM Journal on Computing*, 38(2):658–701, 2008. `doi:10.1137/050645427`.

[BC18]   Mikołaj Bojańczyk and Wojciech Czerwiński. An automata toolbox. Lecture notes for a
             course at the University of Warsaw, 2018. URL: `https://www.mimuw.edu.pl/~bojan/paper/
             automata-toolbox-book`.

[BD20]   Mikołaj Bojańczyk and Amina Doumane. First-order tree-to-tree functions. In Holger Her-
             manns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual
             ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany (online con-
             ference), July 8-11, 2020*, pages 252–265. ACM, 2020. `doi:10.1145/3373718.3394785`.

[BDBRDR18]   Patrick Baillot, Erika De Benedetti, and Simona Ronchi Della Rocca. Characterizing poly-
             nomial and exponential complexity classes in elementary lambda-calculus. *Information and
             Computation*, 261:55–77, August 2018. `doi:10.1016/j.ic.2018.05.005`.

[BDK18]   Mikołaj Bojańczyk, Laure Daviaud, and Shankara Narayanan 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, Oxford, United Kingdom, 2018. ACM Press.
             `doi:10.1145/3209108.3209163`.

[BDSW17]   Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata:
             Zeroness and applications. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer
             Science (LICS)*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE. `doi:10.1109/LICS.2017.
             8005101`.

[BE00]   Roderick Bloem and Joost Engelfriet. A Comparison of Tree Transductions Defined by Monadic
             Second Order Logic and by Attribute Grammars. *Journal of Computer and System Sciences*,
             61(1):1–50, August 2000. `doi:10.1006/jcss.1999.1684`.

[Bie94]   Gavin M. Bierman. On Intuitionistic Linear Logic. Technical report UCAM-CL-TR-346, University of Cambridge, Computer Laboratory, August 1994. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.588&rep=rep1&type=pdf`.

[BKL19]   Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-String Interpretations With Polynomial-Size Output. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.106`.

[BM10]   Patrick Baillot and Damiano Mazza. Linear Logic by Levels and Bounded Time Complexity. *Theoretical Computer Science*, 411(2):470–503, January 2010. `doi:10.1016/j.tcs.2009.09.015`.

[Boj18]   Mikołaj Bojańczyk. Polyregular functions, 2018. `arXiv:1810.08760`.

[Boj21]   Mikołaj Bojańczyk. Algebra for trees. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, chapter 22, pages 801–838. European Mathematical Society Publishing House, 2021. `doi:10.4171/Automata-1/22`.

[CM19]   Pierre Clairambault and Andrzej S. Murawski. On the Expressivity of Linear Recursion Schemes. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.MFCS.2019.50`.

[CP17a]   Thomas Colcombet and Daniela Petrişan. Automata and minimization. *ACM SIGLOG News*, 4(2):4–27, May 2017. `doi:10.1145/3090064.3090066`.

[CP17b]   Thomas Colcombet and Daniela Petrişan. Automata in the Category of Glued Vector Spaces. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.MFCS.2017.52`.

[CP20]   Thomas Colcombet and Daniela Petrişan. Automata Minimization: a Functorial Approach. *Logical Methods in Computer Science*, 16(1), March 2020. `doi:10.23638/LMCS-16(1:32)2020`.

[DGK18]   Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular Transducer Expressions for Regular Transformations. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, pages 315–324, Oxford, United Kingdom, 2018. ACM Press. `doi:10.1145/3209108.3209182`.

[DJR18]   Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic String Transducers. *International Journal of Foundations of Computer Science*, 29(05):801–824, August 2018. `doi:10.1142/S0129054118420054`.

[dP89]   Valeria C. V. de Paiva. The Dialectica categories. In John W. Gray and Andre Scedrov, editors, *Contemporary Mathematics*, volume 92, pages 47–62. American Mathematical Society, Providence, Rhode Island, 1989. `doi:10.1090/conm/092/1003194`.

[DP16]   Henry DeYoung and Frank Pfenning. Substructural proofs as automata. In Atsushi Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 3–22, 2016. `doi:10.1007/978-3-319-47958-3_1`.

[EH01]   Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, April 2001. `doi:10.1145/371316.371512`.

[EM99]   Joost Engelfriet and Sebastian Maneth. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations. *Information and Computation*, 154(1):34–91, October 1999. `doi:10.1006/inco.1999.2807`.

[ERS80]   Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20(2):150–202, 1980. `doi:10.1016/0022-0000(80)90058-6`.

[FLO83]   Steven Fortune, Daniel Leivant, and Michael O'Donnell. The Expressiveness of Simple and Second-Order Type Structures. *Journal of the ACM*, 30(1):151–185, January 1983. `doi: 10.1145/322358.322370`.

[FR16]   Emmanuel Filiot and Pierre-Alain Reynier. Transducers, Logic and Algebra for Functions of Finite Words. *ACM SIGLOG News*, 3(3):4–19, August 2016. `doi:10.1145/2984450.2984453`.

[FR21]   Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. *Fundamenta Informaticae*, 178(1-2):59–76, January 2021. `doi:10.3233/FI-2021-1998`.

[Gal20]   Zeinab Galal. A profunctorial scott semantics. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.FSCD.2020.16`.

[Gir86]   Jean-Yves Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, January 1986. `doi:10.1016/0304-3975(86)90044-7`.

[Gir87]   Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, January 1987. `doi:10.1016/0304-3975(87)90045-4`.

[Gir89]   Jean-Yves Girard. 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. `doi:10.1090/conm/092/1003197`.

[Gir95]   Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 1–42. Cambridge University Press, 1995. `doi:10.1017/CBO9780511629150.002`.

[Gir96]   Jean-Yves Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*, pages 97–124. Marcel Dekker, 1996.

[Gir98]   Jean-Yves Girard. Light Linear Logic. *Information and Computation*, 143(2):175–204, June 1998. `doi:10.1006/inco.1998.2700`.

[GK13]   Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 154, pages 153–192. Cambridge University Press, 2013. `doi:10.1017/S0305004112000394`.

[GLS20]   Paul Gallot, Aurélien Lemay, and Sylvain Salvati. Linear high-order deterministic tree transducers with regular look-ahead. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 38:1–38:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.MFCS.2020.38`.

[Göd58]   Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12(3-4):280–287, 1958. `doi:10.1111/j.1746-8361.1958.tb01464.x`.

[Gre16]   Charles Grellois. *Semantics of linear logic and higher-order model-checking*. PhD thesis, Université Paris 7, April 2016. URL: `https://tel.archives-ouvertes.fr/tel-01311150/`.

[GRV09]   Marco Gaboardi, Luca Roversi, and Luca Vercelli. A By-Level Analysis of Multiplicative Exponential Linear Logic. In *Mathematical Foundations of Computer Science 2009*, Lecture Notes in Computer Science, pages 344–355. Springer, Berlin, Heidelberg, August 2009. `doi:10.1007/978-3-642-03816-7_30`.

[GSS92]   Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97(1):1–66, April 1992. `doi:10.1016/0304-3975(92)90386-T`.

[Hed18]   Jules Hedges. Morphisms of open games. In Sam Staton, editor, *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018, Dalhousie University, Halifax, Canada, June 6-9, 2018*, volume 341 of *Electronic Notes in Theoretical Computer Science*, pages 151–177, 2018. `doi:10.1016/j.entcs.2018.11.008`.

[HH16]   Dominic Hughes and Willem Heijltjes. Conflict nets: Efficient locally canonical MALL proof nets. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), 2016*, pages 437–446, New York, U. S. A., July 2016. ACM. `doi:10.1145/2933575.2934559`.

[Hin03]   Peter Hines. A categorical framework for finite state machines. *Mathematical Structures in Computer Science*, 13(3):451–480, 2003. `doi:10.1017/S0960129503003931`.

[HJ99]   Hongde Hu and André Joyal. Coherence completions of categories. *Theoretical Computer Science*, 227(1):153–184, September 1999. `doi:10.1016/S0304-3975(99)00051-1`.

[HK96]   Gerd G. Hillebrand and Paris 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. `doi:10.1109/LICS.1996.561337`.

[HKM96]   Gerd G. Hillebrand, Paris C. Kanellakis, and Harry G. Mairson. Database Query Languages Embedded in the Typed Lambda Calculus. *Information and Computation*, 127(2):117–144, June 1996. `doi:10.1006/inco.1996.0055`.

[Hof11]   Pieter Hofstra. The dialectica monad and its cousins. In Bradd Hart, Thomas Kucera, Anand Pillay, Philip Scott, and Robert Seely, editors, *Models, Logics, and Higher-Dimensional Categories: A Tribute to the Work of Mihály Makkai*, volume 53 of *CRM Proceedings and Lecture Notes*. American Mathematical Society, Providence, Rhode Island, September 2011. `doi:10.1090/crmp/053`.

[KL80]   Gregory M. Kelly and Miguel L. Laplaza. Coherence for compact closed categories. *Journal of pure and applied algebra*, 19:193–213, 1980. `doi:10.1016/0022-4049(80)90101-2`.

[Lam58]   Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958. URL: `http://www.jstor.org/stable/2310058`.

[Lin07]   Sam Lindley. Extensional rewriting with sums. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2007. `doi:10.1007/978-3-540-73228-0_19`.

[Mel09]   Paul-André Melliès. Categorical semantics of linear logic. In *Interactive models of computation and program behaviour*, volume 27 of *Panoramas et Synthèses*, pages 1–196. Société Mathématique de France, 2009. URL: `https://www.irif.fr/~mellies/papers/panorama.pdf`.

[Mel17]   Paul-André Melliès. Higher-order parity automata. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE. `doi:10.1109/LICS.2017.8005077`.

[ML98]   Saunders Mac Lane. *Categories for the Working Mathematician (2nd edition*. Graduate Texts in Mathematics. Springer, 1998.

[MP19]   Anca Muscholl and Gabriele Puppis. The Many Facets of String Transducers. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.STACS.2019.2`.

[MS95]   David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1–2):69 – 107, 1995. `doi:10.1016/0304-3975(94)00214-4`.

[MT03]   Harry G. Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. In Carlo Blundo and Cosimo Laneve, editors, *Theoretical Computer Science, 8th Italian Conference, ICTCS 2003, Bertinoro, Italy, October 13-15, 2003, Proceedings*, volume 2841 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2003. `doi:10.1007/978-3-540-45208-9_4`.

[MvG18]   Sean K. Moss and Tamara von Glehn. Dialectica models of type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 739–748. ACM Press, 2018. `doi:10.1145/3209108.3209207`.

[Ngu19]   Lê Thành Dũng Nguyễn. On the Elementary Affine Lambda-Calculus with and Without Fixed Points. *Electronic Proceedings in Theoretical Computer Science*, 298:15–29, August 2019. In Proceedings DICE-FOPARA 2019. `doi:10.4204/EPTCS.298.2`.

[nLa20]   nLab authors. Semicartesian monoidal category. `http://ncatlab.org/nlab/show/semicartesian%20monoidal%20category`, March 2020. Revision 24.

[NP19]   Lê Thành Dũng Nguyễn and Cécilia Pradic. From normal functors to logarithmic space queries. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th*

International Colloquium on Automata, Languages and Programming (ICALP 2019), volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 123:1–123:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.123`.

[NP20] Lê Thành Dũng Nguyễn and Cécilia Pradic. Implicit automata in typed λ-calculi I: aperiodicity in a non-commutative logic. In Artur Czumaj, Anuj Dawar, and Emanuela 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. `doi:10.4230/LIPIcs.ICALP.2020.135`.

[PR18] Cécilia Pradic and Colin Riba. LMSO: A Curry-Howard Approach to Church's Synthesis via Linear Logic. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 849–858, New York, NY, USA, 2018. ACM. `doi:10.1145/3209108.3209195`.

[PR19] Cécilia Pradic and Colin Riba. A dialectica-like interpretation of a linear MSO on infinite words. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 470–487. Springer, 2019. `doi:10.1007/978-3-030-17127-8_27`.

[RDRR97] Simona Ronchi Della Rocca and Luca Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 59(3):417–448, 1997. `doi:10.1023/A:1005092630115`.

[Rey93] John C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3):233–247, Nov 1993. `doi:10.1007/BF01019459`.

[Rib20] Colin Riba. Monoidal-closed categories of tree automata. *Mathematical Structures in Computer Science*, 30(1):62–117, January 2020. `doi:10.1017/S0960129519000173`.

[Sch16] Gabriel Scherer. *Which types have a unique inhabitant? : Focusing on pure program equivalence.* PhD thesis, Paris Diderot University, France, 2016. URL: `https://tel.archives-ouvertes.fr/tel-01309712`.

[Ter12] Kazushige Terui. Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, pages 323–338, 2012. `doi:10.4230/LIPIcs.RTA.2012.323`.

[Til87] Bret Tilson. Categories as algebra: An essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48(1):83–198, September 1987. `doi:10.1016/0022-4049(87)90108-3`.

[vHKR+19] Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino, and Alexandra Silva. Tree automata as algebras: Minimisation and determinisation. In Markus Roggenbach and Ana Sokolova, editors, *8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019, June 3-6, 2019, London, United Kingdom*, volume 139 of *LIPIcs*, pages 6:1–6:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CALCO.2019.6`.

[Wad07] Philip Wadler. The Girard–Reynolds isomorphism (second edition). *Theoretical Computer Science*, 375(1-3):201–226, May 2007. `doi:10.1016/j.tcs.2006.12.042`.

[Zai87] Marek Zaionc. Word operation definable in the typed λ-calculus. *Theoretical Computer Science*, 52(1):1–14, January 1987. `doi:10.1016/0304-3975(87)90077-6`.

Appendix A. Alur and D'Antoni's Bottom-Up Ranked Tree Transducers

We give here a self-contained definition of the notion of BRTT corresponding to regular tree functions, as they were designed in [AD17]. Since the paper [AD17] is mainly concerned with transducers over unranked trees, the information concerning the definition of BRTTs is spread over its sections 2.1, 3.7 and 3.8. We restrict here to binary trees (as in [AD17, Sections 3.7 and 3.8]) and avoid using multicategories.

Definition A.1 ([AD17, p. 31:36])**.** The set $\mathrm{BinTree}(\Sigma)$ of *binary trees* over the alphabet Σ, and the set $\partial\mathrm{BinTree}(\Sigma)$ of *one-hole binary trees* are generated by the respective grammars

$$T, U ::= \lceil\cdot\rceil \mid a\lceil T, U\rceil \quad (a \in \Sigma) \qquad\qquad T', U' ::= \square \mid a\lceil T', U\rceil \mid a\lceil T, U'\rceil \quad (a \in \Sigma)$$

That is, $\mathrm{BinTree}(\Sigma)$ consists of binary trees whose leaves are all equal to $\langle\rangle$ and whose nodes are labeled with letters in Σ. As for $\partial\mathrm{BinTree}(\Sigma)$, it contains trees with exactly one leaf labeled $\square$ instead of $\lceil\cdot\rceil$. This "hole" $\square$ is intended to be substituted by a tree: for $T' \in \partial\mathrm{BinTree}(\Sigma)$ and $U \in \mathrm{BinTree}(\Sigma)$, $T'[U]$ denotes T' where $\square$ has been replaced by U.

Definition A.2 ([AD17, p. 31:40])**.** The *binary tree expressions* (E, F below, forming the set $\mathrm{ExprBT}(\Sigma, V, V')$) and *one-hole binary tree expressions* (E', F' below, forming the set $\mathrm{Expr}\partial\mathrm{BT}(\Sigma, V, V')$) over the variable sets V and V' are generated by the grammar (with $x \in V$, $x' \in V'$ and $a \in \Sigma$)

$$E, F ::= \lceil\cdot\rceil \mid x \mid a\lceil E, F\rceil \mid E'[F] \qquad E', F' := \square \mid x' \mid a\lceil E', F\rceil \mid a\lceil E, F'\rceil \mid E'[F']$$

Given $\rho : V \to \mathrm{BinTree}(\Sigma)$ and $\rho' : V' \to \partial\mathrm{BinTree}(\Sigma)$, one defines $E(\rho, \rho') \in \mathrm{BinTree}(\Sigma)$ for $E \in \mathrm{ExprBT}(\Sigma)$ and $E'(\rho, \rho') \in \partial\mathrm{BinTree}(\Sigma)$ for $E \in \mathrm{Expr}\partial\mathrm{BT}(\Sigma)$ in the obvious way.

Definition A.3 ([AD17, §3.7])**.** Let us fix an input alphabet Γ and output alphabet Σ. The set of *register assignments* over two disjoint sets R, R', whose elements are called *registers*, is

$$\mathcal{A}(\Sigma, R, R') = \mathrm{ExprBT}(\Sigma, R_{\lhd\rhd}, R'_{\lhd\rhd})^R \times \mathrm{Expr}\partial\mathrm{BT}(\Sigma, R_{\lhd\rhd}, R'_{\lhd\rhd})^{R'} \quad \text{where } R_{\lhd\rhd} = R \times \{\lhd, \rhd\}$$

A *register tree transducer* consists of a *finite set* Q of *states* with an *initial state* $q_I \in Q$, two disjoint *finite* sets R, R' of *registers*, a *transition function* $\delta : Q \times Q \times \Gamma \to Q \times \mathcal{A}(\Sigma, R, R')$ and an *output function* $F : Q \to \mathrm{ExprBT}(\Sigma, R, R')$. To each tree $T \in \mathrm{BinTree}(\Gamma)$, it associates inductively a *configuration* $\mathrm{Conf}(T) \in Q \times \mathrm{BinTree}(\Sigma)^R \times \partial\mathrm{BinTree}(\Sigma)^{R'}$:

- The base case is $\mathrm{Conf}(\lceil\cdot\rceil) = (q_I, (r \mapsto \lceil\cdot\rceil), (r' \mapsto \square))$.
- When $\mathrm{Conf}(T) = (q_{\lhd}, \rho_{\lhd}, \rho'_{\lhd})$, $\mathrm{Conf}(U) = (q_{\rhd}, \rho_{\rhd}, \rho'_{\rhd})$ and $\delta(c, q_{\lhd}, q_{\rhd}) = (q, (\varepsilon, \varepsilon'))$, we set $\mathrm{Conf}(c\lceil T, U\rceil) = (q, (r \mapsto \varepsilon(r)(\rho, \rho')), (r' \mapsto \varepsilon'(r')(\rho, \rho')))$ where $\rho(r, d) = \rho_d(r)$ for $(r, d) \in R \times \{\lhd, \rhd\}$ and similarly for ρ'.

The function defined by the transducer is $T \in \mathrm{BinTree}(\Gamma) \mapsto F(q_{\mathrm{fin}}(T))(\rho_{\mathrm{fin}}(T), \rho'_{\mathrm{fin}}(T))$ where $(q_{\mathrm{fin}}(T), \rho_{\mathrm{fin}}(T), \rho'_{\mathrm{fin}}(T)) = \mathrm{Conf}(T)$ (recall that F is the output function).

Example A.4 (illustrated by Figure 17)**.** Let us consider a transducer over the alphabets $\Gamma = \Sigma = \{a, b\}$, with a single state ($|Q| = 1$) and two registers, both tree-valued (so $R = \{r_1, r_2\}$ and $R' = \varnothing$). This simplifies the transition function δ into a function $\Gamma \to \mathrm{ExprBT}(\Sigma, R_{\lhd\rhd}, \varnothing)^R$ – equivalently, we will consider $\delta : \Gamma \times R \to \mathrm{ExprBT}(\Sigma, R_{\lhd\rhd}, \varnothing)$.

We take $\delta(c, r_1) = c\lceil (r_1)_{\lhd}, (r_1)_{\rhd}\rceil$ and $\delta(c, r_2) = c\lceil (r_2)_{\rhd}, (r_2)_{\lhd}\rceil$ for $c \in \{a, b\}$, where $r_{\lhd}$ is a notation for $(r, \lhd) \in R_{\lhd\rhd} = R \times \{\lhd, \rhd\}$. If we write $\hat{r}_i(T)$ ($i \in \{1, 2\}$) for the contents of the register r_i at the end of a run of the transducer on $T \in \mathrm{BT}(\Gamma)$, then this δ translates into:

$$\hat{r}_1(c\lceil T, U\rceil) = c\lceil \hat{r}_1(T), \hat{r}_1(U)\rceil \qquad \hat{r}_2(c\lceil T, U\rceil) = c\lceil \hat{r}_2(U), \hat{r}_2(T)\rceil \qquad (c \in \{a, b\})$$

And the initial condition is $\hat{r}_1(\lceil\cdot\rceil) = \hat{r}_2(\lceil\cdot\rceil) = \lceil\cdot\rceil$. Therefore $\hat{r}_1(T) = T$ and $\hat{r}_2(T)$ is T "mirrored" by exchanging left and right; let us write $\hat{r}_2(T) = \mathtt{reverse}(T)$.

The output function F is also simplified into an expression in $\mathrm{ExprBT}(\Sigma, R, \varnothing)$. By taking $F = a\lceil r_1, r_2\rceil$, we define a transducer computing the regular function $T \mapsto a\lceil T, \mathtt{reverse}(T)\rceil$.

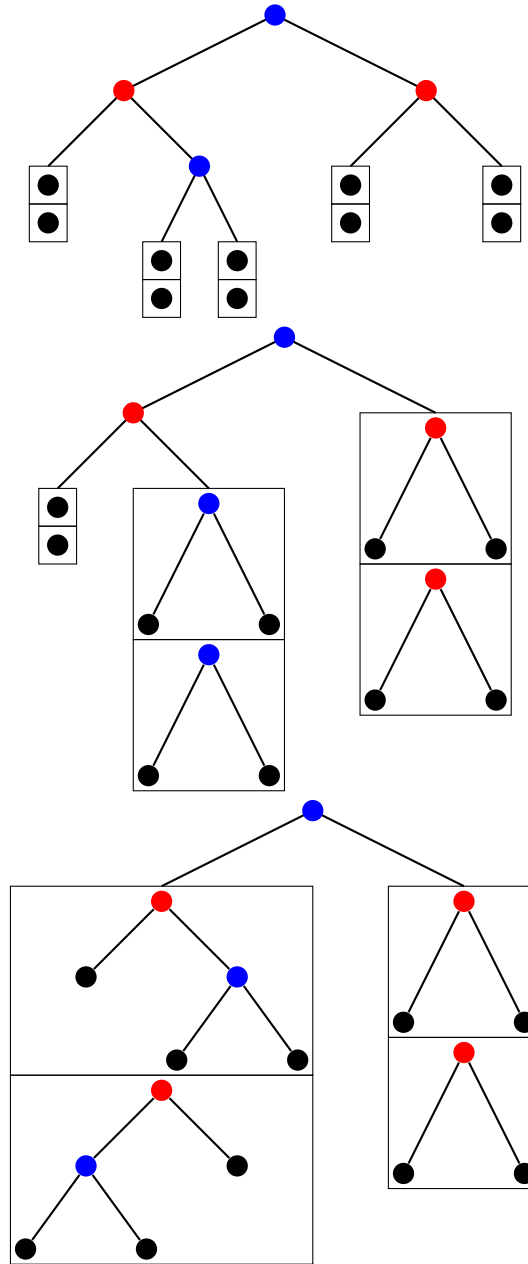Figure 17: First few steps of the run of the bottom-up ranked tree transducer of Figure A.4 over a tree whose alphabet of node labels is $\Sigma = \{\bullet, \bullet\} \cong \{a, b\}$.
The configuration at each subtree is represented by two boxes; the top (resp. bottom) box displays the contents of r_1 (resp. r_2). (The single state is omitted from the visual representation of the configuration.)

To characterize regular functions, a condition must be imposed[29] on the register assignments: the *single use restriction*. It is not intrinsic and depends on an additional piece of data, namely a *conflict relation* between the registers.

Definition A.5 ([AD17, §2.1]). A *conflict relation* is a binary reflexive and symmetric relation.

An expression $E \in \mathrm{ExprBT}(\Sigma, V, V') \cup \mathrm{Expr}\partial\mathrm{BT}(\Sigma, V, V')$ is said to be *consistent with* a conflict relation $\asymp$ over $V \cup V'$ when each variable in $V \cup V'$ appears at most once in E, and for all $x, y \in V \cup V'$, if $x \neq y$ and $x \asymp y$, then E does not contain both x and y.

A register assignment $(\varepsilon, \varepsilon') \in \mathcal{A}(\Sigma, R, R')$ is *single use restricted* with respect to a conflict relation $\asymp$ over $R \cup R'$ when:

- all $\varepsilon(r)$ for $r \in R$ and all $\varepsilon'(r')$ for $r' \in R'$ are consistent with $\asymp$;
- if $x_1, x_2, y_1, y_2 \in R \cup R'$, $x_1 \asymp x_2$ and, for some $d \in \{\triangleleft, \triangleright\}$, (x_1, d) appears in[30] $(\varepsilon \cup \varepsilon')(y_1)$ and (x_2, d) appears in $(\varepsilon \cup \varepsilon')(y_2)$, then $y_1 \asymp y_2$ (note that this includes the case $x_1 = x_2$).

Definition A.6. A *bottom-up ranked tree transducer (BRTT)* is a register tree transducer $(Q, q_I, R, R', \delta, F)$ endowed with a conflict relation $\asymp$ on $R \cup R'$, such that $F(q)$ is consistent with $\asymp$ and all register assignments in the image of δ are single use restricted w.r.t. $\asymp$.

A *regular tree function* is a function computed by a BRTT.

When the conflict relation is trivial (i.e. coincides with equality), we say that the BRTT is *copyless*. We also say that a register tree transducer is copyless if it becomes a BRTT when endowed with a trivial conflict relation.

Appendix B. Normalization of the $\lambda\ell^{\oplus\&}$-calculus

This section is devoted to proving that the $\lambda\ell^{\oplus\&}$-calculus is strongly normalizing. What it means is that any $\lambda\ell^{\oplus\&}$-term t admitting a typing derivation $\Psi; \Delta \vdash t : A$ can be shown to be $\beta\eta$-equivalent to a *normal* term u. The notion of normal (NF) and *neutral* (NE) are defined via the typing system presented in Figure 20. The intuition is that a normal term cannot be β-reduced further, and that neutral terms substituted in normal terms produce terms that stay normal.

The purpose of this section is to show that *any* typed term t can be turned into a normal term t' of the same type via a sequence of β-reductions. Because $\lambda\ell^{\oplus\&}$ features *positive* type constructors $\otimes$ and $\oplus$, the reality is not quite so straightforward: it is not sufficient to β-reduce a term to reach a normal form. In certain circumstances, one must additionally use η-conversion to make certain β-redexes appear and proceed with the computation of normal terms. This difficulty is rather well-known in the context of λ-calculus extended with coproducts [Lin07, ADHS01, Sch16]. However we are not aware of a text treating exactly $\lambda\ell^{\oplus\&}$ (e.g., incorporating additives, a native $\otimes$ and units), so we include a proof using reducibility candidates along the lines of [RR97, Appendix A.1].

To describe said reducibility candidates, we first need to give an oriented version of $\beta\eta$-equality. The β-reduction relation $\rightarrow_\beta$ is obtained by closing the relations given in Figure 18 by congruence. We write $\rightarrow_\beta^*$ for the reflexive transitive closure relation. Much like with $=_\beta$, we assume that terms related by $\rightarrow_\beta$ have the same type in the same context.

---

[29]Without this restriction, one could have outputs of exponential size, whereas regular functions have linearly bounded output. Filiot and Reynier [FR21] have shown that the analogous unrestricted model for strings corresponds to HDT0L systems; we are not aware of any similar result on trees.

[30]By $\varepsilon \cup \varepsilon'$ we mean the map on the *disjoint* union $R \cup R'$ induced in the obvious way by ε and ε'.

$$\begin{array}{lll}
\beta\text{-redexes} & (\lambda x.t)\ u\ \to_\beta\ t[u/x] & (\lambda^! x.t)\ u\ \to_\beta\ t[u/x] \\
& \pi_1(\langle t, u\rangle)\ \to_\beta\ t & \pi_2(\langle t, u\rangle)\ \to_\beta\ u \\
& \mathsf{case}(\mathsf{in}_1(t), x.u, x.v)\ \to_\beta\ u[t/x] & \mathsf{case}(\mathsf{in}_2(t), x.u, x.v)\ \to_\beta\ v[t/x] \\
& \mathsf{let}\ x \otimes y = t \otimes u\ \mathsf{in}\ v\ \to_\beta\ v[t/x][u/y] & \mathsf{let}\ () = ()\ \mathsf{in}\ t\ \to_\beta\ t
\end{array}$$

Figure 18: β-redexes.

As it is not the case that every typable term β-reduces to a normal form, we need to describe another set of reduction rules which involve $=_\eta$. Those *extrusion rules* are listed in Figure 19; we also write $\to_\varepsilon$ for the congruence closure of the relation described there. While the number of cases is daunting, it should be remarked that these rules are obtained mechanically by considering the nesting of an eliminator for a positive type (i.e., the $\mathsf{let}\ \cdot = \cdot\ \mathsf{in}\ \cdot$ constructions $(\otimes, \mathbf{I})$, case $(\oplus)$ and abort (0)) within another eliminator (the aforementioned constructions plus function application $(\to, \multimap)$ and projections π_1, π_2 $(\&)$). For a more careful discussion of (a subset) of these rules, we point the reader to [Sch16, Section 3.3]. We write $\to_\varepsilon^*$ for the reflexive transitive closure of $\to_\varepsilon$, $\to_{\beta\varepsilon}$ for the union of $\to_\beta$ and $\to_\beta$ and $\to_\varepsilon$, and $\to_{\beta\varepsilon}^*$ for its reflexive transitive closure. With these notations, we can state the finer version of the normalization theorem for $\lambda\ell^{\oplus\&}$.

Theorem B.1. *For every term term t such that $\Psi;\ \Delta \vdash t : \tau$, there exists t' such that $t \to_{\beta\varepsilon}^* t'$ and $\Psi;\ \Delta \vdash_{\mathrm{NF}} t' : \tau$.*

Before embarking on the definitions of the reducibility candidates and the proof of Theorem B.1 itself, we first make a couple of observations relating $\to_{\beta\varepsilon}^*$ and $=_{\beta\eta}$.

Lemma B.2. *Suppose that we terms t and t' with matching types and that $t \to_{\beta\varepsilon} t'$. Then, we have $t =_{\beta\eta} t'$. Furthermore if t is normal, so is t'. Similarly, if t is neutral, so is t'.*

Lemma B.3. *If t is normal, then there is no t' such that $t \to_\beta t'$.*

Both of these Lemmas are proved by straightforward induction on the relations $\to_\beta$ and $\to_{\beta\varepsilon}$.

Another crucial ingredient is the confluence of the reduction relation $\to_{\beta\varepsilon}$ (or Church-Rosser property). Alas, this does not hold for syntactic equality (up to α-equivalence. However, it holds up to *commuting conversions*, an equivalence relation $\approx_c$ inductively defined by the clauses in Figure 21 and closure under congruence. We merely state the confluence property that we will use.

Theorem B.4. *If we have $t \to_{\beta\varepsilon}^* u$ and $t \to_{\beta\varepsilon}^* v$, then there exists u' and v' such that $u \to_{\beta\varepsilon}^* u'$, $v \to_{\beta\varepsilon}^* v'$ and $u' \approx_c v'$.*

A first observation is these are compatible with $=_{\beta\eta}$ and that neutral and normal term are preserved by commutative conversions.

Lemma B.5. *If $t \approx_c t'$, then $t =_{\beta\eta} t'$.*

Lemma B.6. *If t is neutral (resp. normal) and $t \approx_c t'$, then t' is also neutral (resp. normal).*

We can now turn to the definition of the reducibility candidates, where we write $t \to_{\beta\varepsilon}^* \approx_c t'$ when there is some t'' such that $t \to_{\beta\varepsilon}^* t''$ and $t'' \approx_c t'$ hold.

Nested $\otimes/\mathbf{I}$ eliminator

$$\begin{aligned}
(\text{let } p = t \text{ in } u) \; v &\;\to_\varepsilon\; \text{let } p = t \text{ in } (u \; v) \\
\pi_i(\text{let } p = t \text{ in } u) &\;\to_\varepsilon\; \text{let } p = t \text{ in } \pi_i(u) \\
\text{let } q = \text{let } p = t \text{ in } u \text{ in } v &\;\to_\varepsilon\; \text{let } p = t \text{ in let } q = u \text{ in } v \\
\text{abort}(\text{let } p = t \text{ in } u) &\;\to_\varepsilon\; \text{let } p = t \text{ in abort}(u) \\
\text{case}(\text{let } p = t \text{ in } u, x.v, y.w) &\;\to_\varepsilon\; \text{let } p = t \text{ in case}(u, x.v, y.w)
\end{aligned}$$

Nested 0 eliminator

$$\begin{aligned}
\text{abort}(t) \; u &\;\to_\varepsilon\; \text{abort}(t) \\
\pi_i(\text{abort}(t)) &\;\to_\varepsilon\; \text{abort}(t) \\
\text{let } p = \text{abort}(t) \text{ in } u &\;\to_\varepsilon\; \text{abort}(t) \\
\text{abort}(\text{abort}(t)) &\;\to_\varepsilon\; \text{abort}(t) \\
\text{case}(\text{abort}(t), x.u, y.v) &\;\to_\varepsilon\; \text{abort}(t)
\end{aligned}$$

Nested $\oplus$ eliminator

$$\begin{aligned}
\text{case}(t, x.u, y.v) \; w &\;\to_\varepsilon\; \text{case}(t, x.u \; w, y.v \; w) \\
\pi_i(\text{case}(t, x.u, y.v)) &\;\to_\varepsilon\; \text{case}(t, x.\pi_i(u), y.\pi_i(v)) \\
\text{let } p = \text{case}(t, x.u, y.v) \text{ in } w &\;\to_\varepsilon\; \text{case}(t, x.\text{let } p = u \text{ in } w, y.\text{let } p = v \text{ in } w) \\
\text{abort}(\text{case}(t, x.u, y.v)) &\;\to_\varepsilon\; \text{case}(t, x.\text{abort}(u), y.\text{abort}(v)) \\
\text{case}(\text{case}(t, x.u, y.v), x'.u', y'.v') &\;\to_\varepsilon\; \text{case}(t, x.\text{case}(u, x'.u', y'.v'), y.\text{case}(v, x'.u', y'.v'))
\end{aligned}$$

Figure 19: The extrusion relation $\to_\varepsilon$ ($i = 1, 2$ and p, q are patterns () or $z \otimes z'$).

Definition B.7. Define a judgment $\Psi; \Delta \models t : \tau$ by induction over the type τ as follows:

- for $\tau = \mathbb{o}, \mathbf{I}, 0$ or $\top$, we have $\Psi; \Delta \models t : \tau$ if and only if there is t' such that $t \to^*_{\beta\varepsilon} t'$ and $\Psi; \Delta \vdash_{\text{NE}} t' : \tau$
- $\Psi; \Delta \models t : \tau \multimap \sigma$ holds if and only if
  - there is t' such that $t \to^*_{\beta\varepsilon} t'$ and $\Psi; \Delta \vdash_{\text{NF}} t' : \tau \multimap \sigma$
  - for every u, Δ' such that $\Psi; \Delta' \models u : \tau$, we have $\Psi; \Delta, \Delta' \models t \; u : \sigma$
- $\Psi; \Delta \models t : \tau \to \sigma$ holds if and only if
  - there is t' such that $t \to^*_{\beta\varepsilon} t'$ and $\Psi; \Delta \vdash_{\text{NF}} t' : \tau \to \sigma$
  - for every u such that $\Psi; \cdot \models u : \tau$, we have $\Psi; \Delta \models t \; u : \sigma$
- $\Psi; \Delta \models t : \tau \otimes \sigma$ holds if and only if
  - there is t' such that $t \to^*_{\beta\varepsilon} t'$ and $\Psi; \Delta \vdash_{\text{NF}} t' : \tau \otimes \sigma$
  - if there are t_1, t_2 such that and $t \to^*_{\beta\varepsilon} t_1 \otimes t_2$, then there are Δ_1 and Δ_2 such that $\Delta = \Delta_1, \Delta_2$ and

$$\Psi; \Delta_1 \models t_1 : \tau \qquad \text{and} \qquad \Psi; \Delta_2 \models t_2 : \sigma$$

- $\Psi; \Delta \models t : \tau \mathbin{\&} \sigma$ holds if and only if
  - there is t' such that $t \to^*_{\beta\varepsilon} t'$ and $\Psi; \Delta \vdash_{\text{NF}} t' : \tau \mathbin{\&} \sigma$
  - $\Psi; \Delta \models \pi_1(t) : \tau$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t:\tau}{\Psi;\ \Delta \vdash_{\mathrm{NF}} t:\tau}$$

$$\overline{\Psi;\ x:\tau \vdash_{\mathrm{NE}} x:\tau} \qquad\qquad\qquad \overline{\Psi, x:\tau;\ \cdot \vdash_{\mathrm{NE}} x:\tau}$$

$$\frac{\Psi;\ \Delta,\ x:\tau \vdash_{\mathrm{NF}} t:\sigma}{\Psi;\ \Delta \vdash_{\mathrm{NF}} \lambda x.t : \tau \multimap \sigma}$$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t:\tau \multimap \sigma \qquad \Psi;\ \Delta' \vdash_{\mathrm{NF}} u:\tau}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} t\,u:\sigma}$$

$$\frac{\Psi,\ x:\tau;\ \Delta \vdash_{\mathrm{NF}} t:\sigma}{\Psi;\ \Delta \vdash_{\mathrm{NF}} \lambda^! x.t : \tau \to \sigma}$$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NF}} t:\tau \qquad \Psi;\ \Delta' \vdash_{\mathrm{NF}} u:\sigma}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NF}} t \otimes u : \tau \otimes \sigma} \qquad \frac{\Psi;\ \Delta' \vdash_{\mathrm{NE}} t:\tau \otimes \sigma \qquad \Psi;\ \Delta,\ x:\tau,\ y:\sigma \vdash_X u:\kappa}{\Psi;\ \Delta,\ \Delta' \vdash_X \mathsf{let}\ x \otimes y = t\ \mathsf{in}\ u : \kappa}$$

$$\frac{\Psi;\ \Delta' \vdash_{\mathrm{NE}} t:\mathbf{I} \qquad \Psi;\ \Delta \vdash_X u:\kappa}{\Psi;\ \Delta,\ \Delta' \vdash_X \mathsf{let}\ () = t\ \mathsf{in}\ u : \kappa}$$

$$\overline{\Psi;\ \cdot \vdash_{\mathrm{NF}} () : \mathbf{I}}$$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NF}} t:\tau \qquad \Psi;\ \Delta \vdash_{\mathrm{NF}} u:\sigma}{\Psi;\ \Delta \vdash_{\mathrm{NF}} \langle t,u \rangle : \tau \,\&\, \sigma} \qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t:\tau \,\&\, \sigma}{\Psi;\ \Delta \vdash_{\mathrm{NE}} \pi_1(t):\tau} \qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t:\tau \,\&\, \sigma}{\Psi;\ \Delta \vdash_{\mathrm{NE}} \pi_2(t):\sigma}$$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NF}} t:\tau}{\Psi;\ \Delta \vdash_{\mathrm{NF}} \mathsf{in}_1(t): \tau \oplus \sigma} \qquad\qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NF}} t:\sigma}{\Psi;\ \Delta \vdash_{\mathrm{NF}} \mathsf{in}_2(t): \tau \oplus \sigma}$$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t:\sigma \oplus \tau \qquad \Psi;\ \Delta',\ x:\sigma \vdash_X u:\kappa \qquad \Psi;\ \Delta',\ y:\tau \vdash_X v:\kappa}{\Psi;\ \Delta,\ \Delta' \vdash_X \mathsf{case}(t, x.u, y.v):\kappa}$$

$$\overline{\Psi;\ \Delta \vdash_{\mathrm{NF}} \langle\rangle : \top} \qquad\qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t:0}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} \mathsf{abort}(t):\tau}$$

Figure 20: Normal forms for $\lambda \ell^{\oplus \&}$-terms ($\vdash_{\mathrm{NF}}$ for normal forms and $\vdash_{\mathrm{NE}}$ for neutral forms and $X \in \{\mathrm{NE}, \mathrm{NF}\}$).

  – $\Psi;\ \Delta \models \pi_2(t):\sigma$
- $\Psi;\ \Delta \models t : \tau \oplus \sigma$ holds if and only if
  – there is t' such that $t \to_{\beta\varepsilon}^* t'$ and $\Psi;\ \Delta \vdash_{\mathrm{NF}} t' : \tau \oplus \sigma$
  – if there is u such that $t \to_{\beta\varepsilon}^* \approx_c \mathsf{in}_1(u)$, then $\Psi;\ \Delta \models u : \tau$
  – if there is v such that $t \to_{\beta\varepsilon}^* \approx_c \mathsf{in}_2(v)$, then $\Psi;\ \Delta \models v : \sigma$

The set of terms t such that $\Psi;\ \Delta \models t:\tau$ constitutes our set of reducibility candidates at type τ in the context $\Psi;\ \Delta$. They are defined in such a way that if $\Psi;\ \Delta \models t:\tau$, then there is t' such that $t \to_{\beta\varepsilon}^* t'$ and $\Psi;\ \Delta \vdash_{\mathrm{NF}} t':\tau$. We shall be able to conclude this section if

$$
\begin{array}{rcl}
\mathsf{let}\ q = u\ \mathsf{in}\ \mathsf{let}\ p = t\ \mathsf{in}\ v & \approx_c & \mathsf{let}\ p = t\ \mathsf{in}\ \mathsf{let}\ q = u\ \mathsf{in}\ v \\
\mathsf{let}\ p = t\ \mathsf{in}\ \mathsf{abort}(u) & \approx_c & \mathsf{abort}(\mathsf{let}\ p = t\ \mathsf{in}\ u) \\
\mathsf{let}\ p = t\ \mathsf{in}\ \mathsf{case}(u, x.v, y.w) & \approx_c & \mathsf{case}(u, x.\mathsf{let}\ p = t\ \mathsf{in}\ v, y.\mathsf{let}\ p = t\ \mathsf{in}\ w) \\
\mathsf{case}(u, x.\mathsf{abort}(t), y.\mathsf{abort}(v)) & \approx_c & \mathsf{abort}(\mathsf{case}(u, x.t, y.v))
\end{array}
$$

$$
\mathsf{case}(t, x.\mathsf{case}(u, x'.v, y'.w), y.\mathsf{case}(u, x'.v', y'.w'))
$$
$$
\approx_c
$$
$$
\mathsf{case}(u, x'.\mathsf{case}(t, x.v, y.v'), y'.\mathsf{case}(t, x.w, y.w'))
$$

Figure 21: Commutative conversions $\approx_c$ (p, q are patterns () or $z \otimes z'$ and both sides are assumed to be well-scoped).

we show an *adequacy lemma* stating that every typable term lies in a reducibility candidate. Before doing that, we first need a couple of stability properties: closure under anti-reduction, and the fact that every neutral term lies in a reducibility candidate.

Lemma B.8. *If t is a neutral term, then t cannot be $\beta\eta$-equivalent to one of the following*

$$
\lambda x.u \qquad \lambda^! x.u \qquad (u, v) \qquad u \otimes v \qquad \langle\rangle \qquad () \qquad \mathsf{in}_1(u) \qquad \mathsf{in}_2(u)
$$

Proof. Trivial case-analysis. □

Theorem B.9. *Suppose that $\Psi; \Delta \models t' : \tau$. Then the following hold:*
- *There exists t'' such that $t' \to^*_{\beta\varepsilon} t''$ and $\Psi; \Delta \vdash_{\mathrm{NF}} t'' : \tau$.*
- *If we have $t \to^*_{\beta\varepsilon} t'$, then we also have $\Psi; \Delta \models t : \tau$.*
- *If $\Psi; \Delta \vdash_{\mathrm{NE}} t : \tau$, then $\Psi; \Delta \models t : \tau$.*
- *If $t' \approx_c t''$, then $\Psi; \Delta \models t'' : \tau$.*
- *If $t' \to^*_{\beta\varepsilon} t''$, then we also have $\Psi; \Delta \models t'' : \tau$.*

Proof. The first point can be proven via an easy case analysis on τ that we skip. The second point we may prove by induction over τ; let us sketch a few representative cases:
- If τ is $\mathbb{0}$, 0, $\mathbf{I}$ or $\top$, this is immediate.
- Suppose that $\tau = \sigma \multimap \kappa$ and that $t \to^*_{\beta\varepsilon} t'$. By definition of $\models$ at $\tau \multimap \kappa$, there is some normal t'' such that $t' \to^*_{\beta\varepsilon} t''$, so we also have $t \to^*_{\beta\varepsilon} t''$ by transitivity. Now suppose that we are given some u and Δ' such that $\Psi; \Delta' \models u : \tau$. By definition, we have $\Psi; \Delta, \Delta' \models t' u : \kappa$. Using the induction hypothesis at κ and the fact that $t u \to^*_{\beta\varepsilon} t' u$, we thus have that $\Psi; \Delta, \Delta' \models t u : \kappa$. We can thus conclude that $\Psi; \Delta \models t : \kappa$.
- Suppose that $\tau = \sigma \oplus \kappa$ and that $t \to^*_{\beta\varepsilon} t'$. By definition of $\models$, there is some normal t'' such that $t' \to^*_{\beta\varepsilon} t''$, so we also have $t \to^*_{\beta\varepsilon} t''$ by transitivity. Now if we have u (resp. v) such that $t \to^*_{\beta\varepsilon}\approx_c \mathsf{in}_1(u)$ (resp. $\mathsf{in}_2(v)$), then, thanks to confluence (Theorem B.4) we also have $t' \to^*_{\beta\varepsilon}\approx_c \mathsf{in}_1(u)$ (resp. $\mathsf{in}_2(v)$), so we have $\Psi; \Delta \models u : \sigma$ (resp. $\Psi; \Delta \models u : \kappa$) by definition. Therefore, we may conclude that $\Psi; \Delta \models t : \sigma \oplus \kappa$.

The third point is also proved via a straightforward induction over τ by leveraging Lemma B.8. The last two points follow from induction over τ combined with Theorem B.4 and Lemma B.6. □

Corollary B.10. *If x is a variable of type τ in either Ψ or Δ, we have $\Psi; \Delta \models x : \tau$.*

Proof. Immediate as variables are neutral. □

Lemma B.11. *Suppose that we have* $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau \otimes \sigma$ *and* $\Psi;\ \Delta', x : \tau, y : \sigma \models u : \kappa$. *Then we have* $\Psi;\ \Delta,\ \Delta' \models \mathsf{let}\ x \otimes y = t\ \mathsf{in}\ u : \kappa$.

Similarly, if $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau \oplus \sigma$, $\Psi;\ \Delta', x : \tau \models u : \kappa$ *and* $\Psi;\ \Delta', y : \sigma \models u' : \kappa$, *we have* $\Psi;\ \Delta, \Delta' \models \mathsf{case}(t, x.u, y.u') : \kappa$.

Finally, if $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \mathbf{I}$ *and* $\Psi;\ \Delta' \models u : \kappa$, *we also have* $\Psi;\ \Delta,\ \Delta' \models \mathsf{let}\ () = t\ \mathsf{in}\ u : \kappa$.

Proof. By induction over κ. □

Theorem B.12 (Adequacy). *Suppose that we have a non-linear context* $\Psi = x_1 : \sigma_1,\ \ldots,\ x_k : \sigma_k$, *a linear context* $\Delta = a_1 : \tau_1, \ldots,\ a_n : \tau_n$ *and a term* v *such that* $\Psi;\ \Delta \vdash v : \kappa$ *for some type* κ. *Further, assume that* $\Psi', \Delta'_1, \ldots, \Delta'_n$ *and terms* $t_1, \ldots t_k, u_1, \ldots, u_n$ *such that* $\Psi';\ \cdot \models t_i : \sigma_i$ *for* $1 \le i \le k$ *and* $\Psi';\ \Delta'_j \models u_j : \tau_j$ *for* $1 \le j \le n$. *Then we have*

$$\Psi';\ \Delta'_1, \ldots,\ \Delta'_n \models v[t_1/x_1, \ldots, t_k/x_k, u_1/a_1, \ldots, u_n/a_n] : \kappa$$

Proof. The proof goes by induction over the typing derivation $\Psi;\ \Delta \vdash v : \kappa$; we sketch a few representative subcases below. To keep notations short, we write γ (respectively δ) instead of the sequence of assignments $t_1/x_1, \ldots, t_k/x_k$ (respectively $u_1/a_1, \ldots, u_n/a_n$) and $\Delta' = \Delta'_1, \ldots,\ \Delta'_n$.

- If the last rule used in an axiom, the conclusion is immediate.
- If the last rule used is a linear function application

$$\frac{\Psi;\ \Delta_1 \vdash v : \kappa \multimap \kappa' \qquad \Psi;\ \Delta_2 \vdash v' : \kappa}{\Psi;\ \Delta_1,\ \Delta_2 \vdash v\ v' : \kappa'}$$

  with δ_1, δ_2 and $\Delta''_1,\ \Delta''_2$ the obvious decomposition of δ and Δ', the induction hypothesis yields

$$\Psi';\ \Delta''_1 \models v[\gamma, \delta_1] : \kappa \multimap \kappa' \qquad \text{and} \qquad \Psi';\ \Delta''_2 \models v'[\gamma, \delta_2] : \kappa$$

  By definition of $\models$ for type $\kappa \multimap \kappa'$, we thus have $\Psi';\ \Delta' \models v[\gamma, \delta_1]\ v'[\gamma, \delta_2] : \kappa'$, so we may conclude.
- The case of non-linear function application is entirely analogous.
- If the last rule is the typing of a linear λ-abstraction

$$\frac{\Psi;\ \Delta,\ c : \kappa \vdash v : \kappa'}{\Psi;\ \Delta \vdash \lambda c.v : \kappa \multimap \kappa'}$$

  by the inductive hypothesis, we have $\Psi';\ \Delta', \Delta'' \vdash v[\gamma, \delta, v'/c] : \kappa'$ for any v' and Δ'' such that $\Psi';\ \Delta'' \models v' : \kappa$. We can prove the conjunct defining $\Psi';\ \Delta' \models (\lambda c.v)[\gamma, \delta] : \kappa \multimap \kappa'$ as follows:
  - First, by taking $\Delta'' = c : \kappa$ (Corollary B.10), we obtain that there exists some v'' such that $v[\gamma, \delta] \to^*_{\beta\varepsilon} v''$ and $\Psi;\ \Delta,\ c : \kappa \vdash_{\mathrm{NF}} v'' : \kappa'$. Therefore we have $\lambda c.v[\gamma, \delta] \to^*_{\beta\varepsilon} \lambda c.v'$ and $\Psi;\ \Delta \vdash_{\mathrm{NF}} \lambda c.v' : \kappa \multimap \kappa'$.
  - Then, assume we have some v' and $\Psi;\ \Delta'' \models v' : \kappa$, so that we have $\Psi';\ \Delta', \Delta'' \vdash v[\gamma, \delta, v'/c] : \kappa'$ by the inductive hypothesis. Because

$$(\lambda c.v)[\gamma, \delta]\ v' = (\lambda c.v[\gamma, \delta])\ v' \to_\beta v[\gamma, \delta, v'/c]$$

    we may apply Theorem B.9 to conclude that $\Psi';\ \Delta',\ \Delta'' \models (\lambda c.v)[\gamma, \delta]\ v' : \kappa'$
- The case of the non-linear λ-abstraction for $\to$ is similar.

- If the last rule applied is an introduction of $\otimes$,

$$\frac{\Psi;\ \Delta_1 \vdash t : \tau \qquad \Psi;\ \Delta_2 \vdash u : \sigma}{\Psi;\ \Delta_1,\ \Delta_2 \vdash t \otimes u : \tau \otimes \sigma}$$

call δ_1, δ_2 the splitting of δ according to the decomposition $\Delta = \Delta_1,\ \Delta_2$. The induction hypothesis yields

$$\Psi';\ \Delta_1' \models t[\gamma, \delta_1] : \tau \qquad \text{and} \qquad \Psi';\ \Delta_2' \models u[\gamma, \delta_2] : \sigma$$

By definition it means that we have normal terms t' and u' such that $t[\gamma, \delta_1] \to_{\beta\varepsilon}^* t'$ and $u[\gamma, \delta_2] \to_{\beta\varepsilon}^* u'$, so $(t \otimes u)[\gamma, \delta] \to_{\beta\varepsilon}^* t \otimes u'$. Now suppose that we have $(t \otimes u)[\gamma, \delta] \to_{\beta\varepsilon}^* \approx_c t'' \otimes u''$. It is not difficult to check (by induction over the length of the reductions $\to_{\beta\varepsilon}^*$ and derivation of $\approx_c$) that we have $t[\gamma, \delta_1] \to_{\beta\varepsilon}^* \approx_c t''$ and $u[\gamma, \delta_2] \to_{\beta\varepsilon}^* \approx_c u''$. So by Theorem B.9, we have that $\Psi;\ \Delta_1 \models t'' : \tau$ and $\Psi;\ \Delta_2 \models u'' : \sigma$,so we may conclude.
- If the last rule applied is an elimination of $\otimes$

$$\frac{\Psi;\ \Delta_1 \vdash u : \tau \otimes \sigma \qquad \Psi;\ \Delta_2, x : \tau, y : \sigma \vdash t : \kappa}{\Psi;\ \Delta_1,\ \Delta_2 \vdash \mathsf{let}\ x \otimes y = u\ \mathsf{in}\ t : \kappa}$$

with δ_1, δ_2 the obvious decomposition of δ along $\Delta_1,\ \Delta_2$, the induction hypothesis applied to the first premise yields $\Psi';\ \Delta_1' \models u[\gamma, \delta_1] : \tau \otimes \sigma$. In particular, this means we have $u[\gamma, \delta_1] \to_{\beta\varepsilon}^* u'$ such that $\Psi';\ \Delta_1' \vdash_{\mathrm{NF}} u' : \tau \otimes \sigma$. By Theorem B.9, it suffices to show that $\mathsf{let}\ x \otimes y = u'\ \mathsf{in}\ t[\gamma, \delta_2] \to_{\beta\varepsilon}^* v$ such that $\Psi';\ \Delta_1',\ \Delta_2' \models v : \kappa$ to conclude. We do so by going by induction over the judgment $\Psi';\ \Delta_1' \vdash_{\mathrm{NF}} u' : \tau \otimes \sigma$.
  – If we have $\Psi';\ \Delta_1' \vdash_{\mathrm{NE}} u' : \tau \otimes \sigma$, then we may use the outer inductive hypothesis $\Psi';\ \Delta_2', x : \tau,\ y : \sigma \models t[\gamma, \delta_2] : \kappa$ and apply Lemma B.11.
  – If we have $u' = \mathsf{let}\ x' \otimes y' = u''\ \mathsf{in}\ u'''$, applying the induction hypothesis, we have some v such that

$$\mathsf{let}\ x \otimes y = u'''\ \mathsf{in}\ t[\gamma, \delta_2] \to_{\beta\varepsilon}^* v \qquad \text{and} \qquad \Psi';\ \Delta', x' : \tau', y' : \sigma' \models v : \kappa$$

We may thus conclude using the sequence of reductions

$$\begin{aligned}\mathsf{let}\ x \otimes y = \mathsf{let}\ x' \otimes y' = u''\ \mathsf{in}\ u'''\ \mathsf{in}\ t[\gamma, \delta_2] \quad &\to_\varepsilon \quad \mathsf{let}\ x' \otimes y' = u''\ \mathsf{in}\ \mathsf{let}\ x \otimes y = u'''\ \mathsf{in}\ t[\gamma, \delta_2] \\ &\to_{\beta\varepsilon}^* \quad \mathsf{let}\ x' \otimes y' = u''\ \mathsf{in}\ v\end{aligned}$$

  and Lemma B.11
  – We proceed similarly if $u' = \mathsf{case}(u'', x'.u''', y'.u'''')$, $\mathsf{let}\ () = u''\ \mathsf{in}\ u'''$ or $\pi_i(u'')$.
  – Finally, if $u' = u'' \otimes u'''$, we apply the outer induction hypothesis with the substitution $\gamma, \delta_2, u''/x, u'''/y$ to conclude.

$\square$

Proof of Theorem B.1. Instantiate Theorem B.12 in the case of a trivial substitution ($t_i = x_i$ and $u_j = a_j$) using Corollary B.10 and conclude with Theorem B.9. $\square$

Appendix C. Proof Lemma 3.24 (on $\lambda\ell^{\oplus\&}$-terms defining tree functions)

Definition C.1. Write $\sqsubseteq_+$ for the least preorder relation over $\lambda\ell^{\oplus\&}$ types satisfying the following for every types τ and σ

$$\tau, \sigma \sqsubseteq_+ \tau \otimes \sigma \quad \tau, \sigma \sqsubseteq_+ \tau \oplus \sigma \quad \tau, \sigma \sqsubseteq_+ \tau \,\&\, \sigma \quad \sigma \sqsubseteq_+ \tau \multimap \sigma \quad \sigma \sqsubseteq_+ \tau \to \sigma$$

We say that τ is a *strictly positive subtype* of σ whenever $\tau \sqsubseteq_+ \sigma$.

Definition C.2. A context $\Psi;\ \Delta$ is called *consistent* if there is no term t such that $\Psi;\ \Delta \vdash t : 0$.

Lemma C.3. *A context $\Psi;\ \Delta$ is inconsistent if and only if there is a neutral term t such that $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : 0$.*

Furthermore, if $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau$, the last typing rule applied has one premise $\Psi';\ \Delta' \vdash_{\mathrm{NE}} u : \tau'$ and $\Psi;\ \Delta$ is consistent, then so is $\Psi';\ \Delta'$.

Proof. The first point is an easy corollary of Theorem B.1. The second point follows from a case analysis, using the following facts:

- If $\Psi,\ \Psi';\ \Delta,\ \Delta'$ is consistent, then so is $\Psi;\ \Delta$.
- If $\Psi;\ \Delta,\ \Delta'$ is consistent and $\Psi;\ \Delta' \vdash t : \tau$, then $\Psi;\ \Delta,\ x : \tau$ is consistent.

$\square$

Lemma C.4. *If $\Psi;\ \Delta$ is consistent and $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau$, then there is a variable in $\Psi;\ \Delta$ of type σ with $\tau \sqsubseteq_+ \sigma$.*

Proof. By induction on the judgement $\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau$.

- If the last rule applied was a variable lookup.

$$\overline{\Psi;\ x : \tau \vdash_{\mathrm{NE}} x : \tau} \qquad\qquad \overline{\Psi, x : \tau;\ \cdot \vdash_{\mathrm{NE}} x : \tau}$$

then the conclusion immediately follows.

- The more interesting cases are those of the elimination rules for $\multimap$, $\otimes$ and $\&$.

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau \multimap \sigma \qquad \Psi;\ \Delta' \vdash_{\mathrm{NF}} u : \tau}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} t\, u : \sigma}$$

$$\frac{\Psi;\ \Delta' \vdash_{\mathrm{NE}} t : \tau \otimes \sigma \qquad \Psi;\ \Delta,\ x : \tau,\ y : \sigma \vdash_{\mathrm{NE}} u : \kappa}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} \mathsf{let}\ x \otimes y = t\ \mathsf{in}\ u : \kappa} \qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau \,\&\, \sigma}{\Psi;\ \Delta \vdash_{\mathrm{NE}} \pi_1(t) : \tau}$$

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau \,\&\, \sigma}{\Psi;\ \Delta \vdash_{\mathrm{NE}} \pi_2(t) : \sigma} \qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \sigma \oplus \tau \qquad \Psi;\ \Delta',\ x : \sigma \vdash_{\mathrm{NE}} u : \kappa \qquad \Psi;\ \Delta',\ y : \tau \vdash_{\mathrm{NE}} v : \kappa}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} \mathsf{case}(t, x.u, y.v) : \kappa}$$

$$\frac{\Psi;\ \Delta' \vdash_{\mathrm{NE}} t : \mathbf{I} \qquad \Psi;\ \Delta \vdash_X u : \kappa}{\Psi;\ \Delta,\ \Delta' \vdash_X \mathsf{let}\ () = t\ \mathsf{in}\ u : \kappa} \qquad \frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : 0}{\Psi\ \Delta \vdash_{\mathrm{NE}} \mathsf{abort}(t) : \tau}$$

The treatment of $\&$ and $\mathbf{I}$ is rather straightforward and 0 is ruled out because $\Psi;\ \Delta$ is assumed to be consistent, so we only explain the inductive step for $\multimap$, $\otimes$ and $\oplus$.

– If the last rule applied is the elimination of a linear arrow

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \tau \multimap \sigma \qquad \Psi;\ \Delta' \vdash_{\mathrm{NF}} u : \tau}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} t\, u : \sigma}$$

then the induction hypothesis applied to the first premise means that there is a variable x in $\Psi;\ \Delta$ of type κ such that $\tau \multimap \sigma \sqsubseteq_+ \kappa$, and we may conclude since $\sigma \sqsubseteq_+ \tau \multimap \sigma$.

– If the last rule applied is the elimination of a tensor product

$$\frac{\Psi;\ \Delta' \vdash_{\mathrm{NE}} t : \tau \otimes \sigma \qquad \Psi;\ \Delta,\ x : \tau,\ y : \sigma \vdash_{\mathrm{NE}} u : \kappa}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} \mathsf{let}\ x \otimes y = t\ \mathsf{in}\ u : \kappa}$$

then the induction hypothesis applied to the first premise yields a variable z in $\Psi;\ \Delta,\ x : \tau,\ y : \sigma$ of type ζ with $\kappa \sqsubseteq_+ \zeta$. If $z \notin \{x, y\}$, then z occurs in $\Psi;\ \Delta$ and we may

conclude. Otherwise, suppose that $z = x$; applying the induction hypothesis to the second premise, we know that $\tau \otimes \sigma \sqsubseteq_+ \tau \sqsubseteq_+ \zeta'$ for a ζ' being the type of some variable in Δ' or a $\zeta' = !\zeta''$ with ζ'' being the type of some variable in Ψ. Therefore, we may conclude since $\kappa \sqsubseteq_+ \tau \sqsubseteq_+ \tau \otimes \sigma \sqsubseteq_+ \zeta'$. The case of $z = y$ is treated similarly.

• If the last rule applied is the elimination of a coproduct

$$\frac{\Psi;\ \Delta \vdash_{\mathrm{NE}} t : \sigma \oplus \tau \qquad \Psi;\ \Delta',\ x : \sigma \vdash_{\mathrm{NE}} u : \kappa \qquad \Psi;\ \Delta',\ y : \tau \vdash_{\mathrm{NE}} v : \kappa}{\Psi;\ \Delta,\ \Delta' \vdash_{\mathrm{NE}} \mathsf{case}(t, x.u, y.v) : \kappa}$$

then, by the inductive hypothesis applied to the second premise, there is ζ with $\kappa \sqsubseteq_+ \zeta$ such that
(1) either there is a variable in $\Psi;\ \Delta'$ with type ζ
(2) or $\zeta = \sigma$.
In the first case, we may directly conclude. Otherwise, the induction hypothesis applied to the first premise states that there is ζ'' with $\sigma \oplus \tau \sqsubseteq_+ \zeta''$ so that ζ'' is a type of some variable in Ψ or Δ. Hence, we have $\kappa \sqsubseteq_+ \sigma \sqsubseteq_+ \sigma \oplus \tau \sqsubseteq_+ \zeta''$ and we may conclude.

$\square$

Proposition C.5. *Fix a ranked alphabet Σ. The map $t \mapsto \underline{t}$ taking trees $t \in \mathbf{Tree}(\Sigma)$ to their Church encodings is a bijection.*

Proof of Proposition C.5. For the sake of this proof, let us assume that ranked alphabets Σ are ordered. Recall that if $t \in \mathbf{Tree}(\Sigma)$, we write $\underline{t} : \mathrm{Tree}_{\Sigma}$ for its Church encoding. When $\Sigma = \{a_1, \ldots, a_k\}$, $\underline{t}$ has shape $\lambda^! a_1. \ldots . \lambda^! a_k. t^\circ$. for some neutral term t°. Let us adopt this notation for a map $t \mapsto t^\circ$, mapping trees t to terms $\widetilde{\Sigma};\ \cdot \vdash t^\circ : \mathbb{o}$, and let us abbreviate the sequence of λ-abstractions $\lambda^! a_1. \ldots . \lambda^! a_k.$ as $\lambda^! \Sigma$ for arbitrary (ordered) ranked alphabets Σ.

We use those conventions to show that the map $t \mapsto \underline{t}$ is surjective (where it is understood that the codomain consists of terms *up to $\beta\eta$-equivalence.*

Lemma C.6. *Fix a ranked alphabet $\widetilde{\Sigma}$. For every typed normal term u, we have*
(1) *if $\widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NF}} u : \mathbb{o}$, then there is $t \in \mathbf{Tree}(\Sigma)$ such that $u = t^\circ$.*
(2) *if $\widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NE}} u : \mathbb{o} \multimap \ldots \multimap \mathbb{o}$ where the type of u has k arguments, then there exists $a \in \Sigma$, a list of trees $t_1, \ldots, t_{|\mathrm{ar}(a)|-k} \in \mathbf{Tree}(\Sigma)$ such that $u = a\ t_1^\circ\ \ldots\ t_{|\mathrm{ar}(a)|-k}^\circ$.*

Proof. We proceed by induction over the typing judgment of the normal form u. Many cases are easily seen to not arise (typically, constructor for various datatypes). Most eliminators can also be ignored because of Lemma C.4. For instance, suppose $u = \mathsf{case}(v, x.w, y.w')$. Then it would means that we had some $\widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NE}} v : \tau \oplus \sigma$ for some τ, σ, but that cannot be the case as $\tau \oplus \sigma$ is never a strictly positive subtype of some $\mathbb{o} \multimap \ldots \multimap \mathbb{o}$.

As a consequence, there are only two cases of interest: the variable case and (linear) function application.

• If u is a variable of type $\mathbb{o}$, then the first result (and, as a consequence, the second) result is immediate: u is the Church encoding of a tree with a single leaf. If it is a variable of type $\mathbb{o} \multimap \ldots \multimap \mathbb{o}$, the first claim is vacuously true and the second is also immediate.

• If u is a function application $v\ w$, then we have that $\widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NE}} v : \mathbb{o} \multimap \ldots \multimap \mathbb{o}$ in both cases. So we may apply the inductive hypothesis to obtain some $a \in \Sigma$ and trees $t_1, \ldots, t_l$ such that $v = a\ t_1^\circ\ \ldots\ t_l^\circ$. We also have that $\widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NF}} w : \mathbb{o}$, so we have some tree t_{l+1} such that $w =_{\beta\eta} t_{l+1}^\circ$. Altogether, we thus have $u = a\ t_1^\circ\ \ldots\ t_{l+1}^\circ$ as expected of the second item. If the first item is not vacuously true, we have that $l + 1 = |A|$, and thus, $a\ t_1^\circ\ \ldots\ t_{l+1}^\circ = (a(t_1, \ldots, t_{l+1}))^\circ$ as required.

$\square$

Given two *ordered* ranked alphabets $\mathbf{\Sigma}$ and $\mathbf{\Gamma}$, write $\mathbf{\Sigma} \otimes \mathbf{\Gamma}$ for the ordered ranked alphabet with letters $\Sigma + \Gamma$ determined by $\mathsf{in}_1(a) < \mathsf{in}_2(b)$ for $a \in \Sigma$, $b \in \Gamma$ and where the order is lifted from $\mathbf{\Sigma}$ and $\mathbf{\Gamma}$ otherwise.

Lemma C.7. *Fix ranked alphabets $\mathbf{\Sigma}, \mathbf{\Gamma}$. If we have $\widetilde{\mathbf{\Sigma}}; \cdot \vdash u : \mathrm{Tree}_{\mathbf{\Gamma}}$, then there exists some $t \in \mathbf{Tree}(\mathbf{\Sigma} \otimes \mathbf{\Gamma})$ such that $u =_{\beta\eta} \lambda^! \mathbf{\Gamma}.t^\circ$.*

Proof. First, we use Theorem B.1 to suppose that u is under normal form, and we proceed by induction over the size of $\mathbf{\Gamma}$. If it is empty, then the result follows from the first item of Lemma C.6. Otherwise $\mathbf{\Gamma} = \mathbf{S} \otimes \mathbf{\Gamma}'$ for some singleton alphabet $\mathbf{S}$ with letter b. Then, a quick case analysis shows that, as in Lemma C.6, most cases can be ignored due to the typing of u, and because of considerations based on Lemma C.4. There is only one interesting case which is the non-linear λ-abstraction.

$$\frac{\widetilde{\mathbf{\Sigma}}, \widetilde{\mathbf{S}}; \cdot \vdash v : \mathrm{Tree}_{\mathbf{\Gamma}}}{\widetilde{\mathbf{\Sigma}}; \cdot \vdash \lambda^! b.v : \mathrm{Tree}_{\mathbf{S} \otimes \mathbf{\Gamma}}}$$

We may apply the induction hypothesis as $\widetilde{\mathbf{\Sigma} \otimes \mathbf{S}} = \widetilde{\mathbf{\Sigma}}, \widetilde{\mathbf{S}}$ and get that $u =_{\beta\eta} \lambda^! b.v = \lambda^! b.\lambda^! \mathbf{\Gamma}.t^\circ = \lambda^! \mathbf{S} \otimes \mathbf{\Gamma}.t^\circ$. $\square$

By instantiating this latest lemma with $\mathbf{\Gamma}$ empty, we can thus deduce that the map $t \mapsto \underline{t}$ is surjective. We may also show that it is injective by exhibiting a left-inverse map, using a semantic interpretation of $\lambda\ell^{\oplus\&}$ into Set: with $\mathbf{\Sigma}$ fixed, use the cartesian-closed structure and coproducts to interpret $\lambda\ell^{\oplus\&}$ with the interpretation of $\mathbb{o}$ being $\mathbf{Tree}(\mathbf{\Sigma})$. This yields a map from terms $t : \mathrm{Tree}_{\mathbf{\Sigma}}$ to set-theoretic functions $(\mathbf{Tree}(\mathbf{\Sigma}) \to \ldots \to \mathbf{Tree}(\mathbf{\Sigma})) \to \ldots \to \mathbf{Tree}(\mathbf{\Sigma})$, where the arguments correspond to the arity of tree constructors; feed the actual constructors to this function to recover a tree in $\mathbf{Tree}(\mathbf{\Sigma})$.

It is straightforward to check that this map is indeed a left inverse of $t \mapsto \underline{t}$, by induction over t. Hence the map $t \mapsto \underline{t}$ is bijective. $\square$

Lemma C.8. *Let $\tau = \kappa_1 \to \ldots \to \kappa_k \to \kappa'$ be a type and s a distinguished variable of type τ. Let $\widetilde{\mathbf{\Sigma}}$ be a ranked alphabet such that $\widetilde{\mathbf{\Sigma}}; s : \tau$ is consistent. Then, if there is $k' < k$ such that $\widetilde{\mathbf{\Sigma}}; s : \tau \vdash_{\mathrm{NE}} t : \kappa_{k'+1} \to \ldots \to \kappa_k \to \kappa$, there are also terms $d_1, \ldots, d_{k'}$ such that*

$$t =_{\beta\eta} s \, d_1 \, \ldots \, d_{k'} \quad and \quad \widetilde{\mathbf{\Sigma}}; \cdot \vdash_{\mathrm{NF}} d_i : \kappa_i \qquad for \ i \in \{1, \ldots, k'\}$$

Proof. By induction over k'. Note that t being neutral is essential here. $\square$

Lemma C.9. *Let $\tau = \kappa_1 \to \ldots \to \kappa_k \to \kappa'$ be a type with κ' purely linear and s a distinguished variable of type τ. Let $\widetilde{\mathbf{\Sigma}}$ be a ranked alphabet such that $\widetilde{\mathbf{\Sigma}}; s : \tau$ is consistent and t be a term such that $\widetilde{\mathbf{\Sigma}}; s : \tau \vdash_{\mathrm{NE}} t : \sigma$ for some σ such that $\sigma \sqsubseteq_+ \kappa'$ or of the shape $\mathbb{o} \multimap \ldots \multimap \mathbb{o}$. Then, there are terms $o, d_1, \ldots, d_k$ such that $t =_{\beta\eta} o \, (s \, d_1 \, \ldots \, d_k)$ and*

$$\widetilde{\mathbf{\Sigma}}; \cdot \vdash o : \kappa' \multimap \sigma \qquad \qquad \widetilde{\mathbf{\Sigma}}; \cdot \vdash_{\mathrm{NF}} d_i : \kappa_i \quad for \ i \in \{1, \ldots, k\}$$

Proof. We proceed by induction over a derivation of $\widetilde{\mathbf{\Sigma}}; s : \tau \vdash_{\mathrm{NE}} t : \sigma$. Note that to apply the induction hypothesis, we need to ensure that every context under consideration is consistent. We keep this check implicit as it always follows from Lemma C.3.

- If the last rule applied is a variable lookup, then the term in question must be s itself. Furthermore, we must have $k = 0$, so we may simply take $o = \lambda x.x$ to conclude.

- If the last rule considered is the following instance of the application rule

$$\frac{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} t:\sigma' \multimap \sigma \qquad \widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NF}} u:\sigma'}{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} t\,u:\sigma}$$

  then, by Lemma C.4 (applied on the first premise), it means that we have

$$\text{either} \quad \sigma' \multimap \sigma \sqsubseteq_+ \tau \qquad \text{or} \quad \sigma \multimap \sigma' = \mathbb{0} \multimap \ldots \multimap \mathbb{0}$$

  In the first case, we can further see that $\sigma' \multimap \sigma \sqsubseteq_+ \kappa'$, so in both cases the induction hypothesis can be applied to the first premise to yield some o' and $d_1,\ldots,d_k$ such that $o'\,(s\,d_1 \ldots d_k) =_{\beta\eta} t$, and we may set $o = \lambda s.\ o'\,s\,u$ to conclude.

- If the last rule considered is the following instance of the application rule

$$\frac{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} t:\sigma' \to \sigma \qquad \widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NF}} u:\sigma'}{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} t\,u:\sigma}$$

  then, by Lemma C.4 (applied on the first premise), it means that we have

$$\text{either} \quad \sigma' \to \sigma \sqsubseteq_+ \tau \qquad \text{or} \quad \sigma \to \sigma' = \mathbb{0} \multimap \ldots \multimap \mathbb{0}$$

  The second alternative is absurd, and the first leads to $\sigma = \kappa'$ and $\sigma' = \kappa_k$. Therefore, we may apply Lemma C.8 to get terms $d_1,\ldots,d_{k-1}$ in normal form such that $t =_{\beta\eta} s\,d_1\,\ldots\,d_{k-1}$. We then set d_k to be u and o to be the identity to conclude.

- If the last rule considered is the other instance of the application rule

$$\frac{\widetilde{\Sigma};\ \Delta \vdash_{\mathrm{NE}} t:\sigma' \multimap \sigma \qquad \widetilde{\Sigma};\ \Delta',\ s:\tau \vdash_{\mathrm{NF}} u:\sigma'}{\widetilde{\Sigma};\ \Delta,\ \Delta',\ s:\tau \vdash_{\mathrm{NE}} t\,u:\sigma}$$

  by Lemma C.4 applied to the first premise, we know that $\sigma' = \mathbb{0}$. Therefore, we may apply the induction hypothesis to the second premise to obtain $d_1,\ldots,d_k$ and o' such that $u =_{\beta\eta} o'\,(s\,d_1\,\ldots\,d_k)$, in which case, $t\,u =_{\beta\eta} (\lambda x.\ t\,(o\,x))\,(s\,d_1\,\ldots\,d_k)$. We conclude by setting $o = \lambda z.\ t\,(o'\,z)$.

- If the last rule considered is the following instance of the elimination of tensor products

$$\frac{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} t:\zeta_1 \otimes \zeta_2 \qquad \widetilde{\Sigma};\ x_1:\zeta_1,\ x_2:\zeta_2 \vdash_{\mathrm{NE}} u:\sigma}{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} \mathsf{let}\ x \otimes y = t\ \mathsf{in}\ u:\sigma}$$

  then, by the induction hypothesis (which is applicable because of Lemma C.4), there are $d_1,\ldots,d_k$ and o' such that $t =_{\beta\eta} o'\,(s\,d_1\,\ldots\,d_k)$, in which case, we conclude by setting $o = \lambda z.\ \mathsf{let}\ x \otimes y = o'\,z\ \mathsf{in}\ u$.

- The last rule considered cannot be the following instance of the elimination of tensor products

$$\frac{\widetilde{\Sigma};\ \cdot \vdash_{\mathrm{NE}} t:\zeta_1 \otimes \zeta_2 \qquad \widetilde{\Sigma};\ s:\tau,\ x_1:\zeta_1,\ x_2:\zeta_2 \vdash_{\mathrm{NE}} u:\sigma}{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} \mathsf{let}\ x \otimes y = t\ \mathsf{in}\ u:\sigma}$$

  as Lemma C.4 would require that $\zeta_1 \otimes \zeta_2$ be a strictly positive subtype of some $\mathbb{0} \multimap \ldots \multimap \mathbb{0}$.

- If the last rule considered types a projection

$$\frac{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} t:\sigma_1\,\&\,\sigma_2}{\widetilde{\Sigma};\ s:\tau \vdash_{\mathrm{NE}} \pi_i(t):\sigma_i}$$

  then the induction hypothesis yields terms o' and $d_1,\ldots,d_k$ such that $t =_{\beta\eta} o'\,(s\,d_1 \ldots d_k)$. We may set $o = \lambda z.\ \pi_i(o'\,z)$ to conclude.

- If the last rule applied is an elimination of a coproduct

$$\frac{\widetilde{\Sigma};\; s:\tau \vdash_{\mathrm{NE}} t:\zeta_1 \oplus \zeta_2 \qquad \widetilde{\Sigma};\; x:\zeta_1 \vdash_{\mathrm{NE}} u:\sigma \qquad \widetilde{\Sigma};\; y:\zeta_2 \vdash_{\mathrm{NE}} v:\sigma}{\widetilde{\Sigma};\; \cdot \vdash_{\mathrm{NE}} \mathsf{case}(t,x.u,y.v):\sigma}$$

then, the induction hypothesis (applicable because of Lemma C.4) yields terms o' and $d_1,\ldots,d_k$ such that $t =_{\beta\eta} o'\ (s\ d_1 \ldots d_k)$. We may set $o = \lambda z.\ \mathsf{case}(o'\ z, x.u, y.v)$ to conclude.

- The last rule applied cannot be one of the following instances of the elimination of a coproduct because of Lemma C.4 applied to the first premise:

$$\frac{\widetilde{\Sigma};\; \cdot \vdash_{\mathrm{NE}} t:\zeta_1 \oplus \zeta_2 \qquad \widetilde{\Sigma};\; s:\tau,\, x:\zeta_1 \vdash_X u:\sigma \qquad \widetilde{\Sigma};\; y:\zeta_2 \vdash_X v:\sigma}{\widetilde{\Sigma};\; s:\tau \vdash_X \mathsf{case}(t,x.u,y.v):\sigma}$$

$$\frac{\widetilde{\Sigma};\; \cdot \vdash_{\mathrm{NE}} t:\zeta_1 \oplus \zeta_2 \qquad \widetilde{\Sigma};\; x:\zeta_1 \vdash_X u:\sigma \qquad \widetilde{\Sigma};\; s:\tau,\, y:\zeta_2 \vdash_X v:\sigma}{\widetilde{\Sigma};\; s:\tau \vdash_X \mathsf{case}(t,x.u,y.v):\sigma}$$

- If the last rule applied is an elimination of $\mathbf{I}$

$$\frac{\widetilde{\Sigma};\; s:\tau \vdash_{\mathrm{NE}} t:\mathbf{I} \qquad \widetilde{\Sigma};\; \cdot \vdash_{\mathrm{NE}} u:\sigma}{\widetilde{\Sigma};\; \cdot \vdash_{\mathrm{NE}} \mathsf{let}\ () = t\ \mathsf{in}\ u:\sigma}$$

then, the induction hypothesis (applicable because of Lemma C.4) yields terms o' and $d_1,\ldots,d_k$ such that $t =_{\beta\eta} o'\ (s\ d_1 \ldots d_k)$. We may set $o = \lambda z.\ \mathsf{let}\ () = o'\ z\ \mathsf{in}\ u$ to conclude.

- The last rule applied cannot be one of the other instances of $\mathbf{I}$ because of Lemma C.4.
- Finally, since the context under consideration are assumed to be consistent, the last rule applied cannot be an elimination of 0.

$\square$

Lemma C.10. *Let Σ and Γ be ranked alphabets. If the context $\widetilde{\Gamma};\; s : \mathrm{Tree}_{\Sigma}[\kappa]$ is inconsistent, then $\mathbf{Tree}(\Sigma) = \varnothing$.*

Proof. Let us write $[\![-]\!]$ for a semantic interpretation of $\lambda\ell^{\oplus\&}$ types as (classical) propositions following the usual type/proposition mapping (i.e., $[\![\tau \multimap \sigma]\!] = [\![\tau \to \sigma]\!] = [\![\tau]\!] \Rightarrow [\![\sigma]\!]$, $[\![\tau \otimes \sigma]\!] = [\![\tau \& \sigma]\!] = [\![\tau]\!] \wedge [\![\sigma]\!]$, $[\![0]\!] = \bot$, ...) and such that

$$[\![\mathbb{o}]\!] \quad \Leftrightarrow \quad \mathbf{Tree}(\Gamma) \neq \varnothing$$

It is easy to check that, under the usual conjunctive interpretation of contexts, if $\Psi;\; \Delta \vdash t : \tau$, then $[\![\Psi]\!] \wedge [\![\Delta]\!] \Rightarrow [\![\tau]\!]$ holds. Further, our choice for $[\![\mathbb{o}]\!]$ means that $[\![\widetilde{\Gamma}]\!]$ holds, so, if our context $\widetilde{\Gamma};\; s : \mathrm{Tree}_{\Sigma}[\kappa]$ is inconsistent, $\neg[\![\mathrm{Tree}_{\Sigma}[\kappa]]\!]$ holds. Now, assume that $\mathbf{Tree}(\Sigma)$ has an inhabitant t. There is a corresponding Church encoding $\underline{t}$ of type Tree_{Σ}, which has also type $\mathrm{Tree}_{\Sigma}[\kappa]$. Hence, $[\![\mathrm{Tree}_{\Sigma}[\kappa]]\!]$ also holds, leading to a contradiction. $\square$

Proof of Lemma 3.24. Assume that we have a closed $\lambda\ell^{\oplus\&}$-term t of type $\mathrm{Tree}_{\Sigma}[\kappa] \multimap \mathrm{Tree}_{\Gamma}$ with $\mathbf{Tree}(\Sigma) \neq \varnothing$, so that we may safely assume $\widetilde{\Gamma};\; s : \mathrm{Tree}_{\Sigma}[\kappa]$ to be consistent. By η-expansion, we have is of the shape

$$t =_{\beta\eta} \lambda w.\lambda^! b_1.\ldots.\lambda^! b_k.t\ w\ b_1 \ldots b_k$$

and $\widetilde{\Gamma};\; s : \mathrm{Tree}_{\Sigma}[\kappa] \vdash t\ w\ b_1 \ldots b_k : \mathbb{o}$. By normalization of $\lambda\ell^{\oplus\&}$, there is t' such that

$$t' =_{\beta\eta} t\ w\ b_1 \ldots b_k \qquad \text{and} \qquad \widetilde{\Gamma};\; s : \mathrm{Tree}_{\Sigma}[\kappa] \vdash_{\mathrm{NF}} t' : \mathbb{o}$$

For the latter, note that an easy case analysis shows that every (open) term of type $\mathbb{0}$ is in fact neutral, so we may conclude by applying Lemma C.9 and the fact that $=_{\beta\eta}$ is a congruence over terms. $\qquad\square$

Appendix D. Proof of Theorem 3.53 (monoidal closure)

Here we give a detailed proof of Theorem 3.53 by establishing a tensor/hom adjunction

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(C \otimes A, B\right) \cong \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(C, A \multimap B\right)$$

natural in C. While we will not devote too much space to checking naturality down to low-level details, we have attempted to break down this verification into manageable chunks. In particular, our notations are going to stress the functoriality of the various intermediate expressions in our computation. Some lemmas will be useful for this.

Lemma D.1. *We can lift every functor $F : \mathcal{C}^{\mathrm{op}} \to \mathsf{Set}$ to a functor $\langle\&\rangle F : (\mathcal{C}_\&)^{\mathrm{op}} \to \mathsf{Set}$ such that for $(C_i) \in \mathsf{Obj}(C)^I$,*

$$\langle\&\rangle F\left(\underset{i \in I}{\&}\, C_i\right) \;=\; \sum_{i \in I} F(C_i)$$

and in such a way that if $F, G : \mathcal{C}^{\mathrm{op}} \to \mathsf{Set}$ are naturally isomorphic, so are $\langle\&\rangle F$ and $\langle\&\rangle G$.

Proof. Since $(\mathcal{C}_\&)^{\mathrm{op}} = (\mathcal{C}^{\mathrm{op}})_\oplus$ by duality, the existence of $\langle\&\rangle F$ is simply the universal property of the *coproduct* completion! Concretely, consider a morphism

$$f = \left(\varphi_f : I \to J,\; \left(f_i : C'_{\varphi_f(i)} \to C_i\right)_{i \in I}\right) \in \mathsf{Hom}_{\mathcal{C}_\&}\left(\underset{j \in J}{\&}\, C'_j,\; \underset{i \in I}{\&}\, C_i\right)$$

Its functorial image is given by

$$\langle\&\rangle F(f) \quad : \quad (i, x) \in \sum_{i \in I} F(C_i) \quad \mapsto \quad (\varphi_f(i), F(f_i)(x)) \in \sum_{j \in J} F(C'_j)$$

Using this explicit expression, one can check that if $\eta_X : F(X) \xrightarrow{\sim} G(X)$ is a natural isomorphism (for X varying over $\mathcal{C}^{\mathrm{op}}$), then

$$(i, x) \in \sum_{i \in I} F(C_i) \quad \mapsto \quad (i, \eta_{C_i}(x)) \in \sum_{i \in I} G(C_i)$$

defines a natural isomorphism between $\langle\&\rangle F$ and $\langle\&\rangle G$. $\qquad\square$

Lemma D.2. *We can lift every functor $F : (\mathcal{C}_\&)^{\mathrm{op}} \to \mathsf{Set}$ to $\langle\oplus\rangle F : (\mathcal{C}_{\oplus\&})^{\mathrm{op}} \to \mathsf{Set}$ in such a way that if $F, G : (\mathcal{C}_\&)^{\mathrm{op}} \to \mathsf{Set}$ are naturally isomorphic, so are $\langle\oplus\rangle F$ and $\langle\oplus\rangle G$, and with the action on objects*

$$\langle\oplus\rangle F\left(\bigoplus_{u \in U} \underset{x \in X_u}{\&}\, C_x\right) \;=\; \prod_{u \in U} F\left(\underset{x \in X_u}{\&}\, C_x\right)$$

Proof. Write $\mathcal{C}_{\oplus\&} \cong \mathcal{D}_\oplus$ with $\mathcal{D} = \mathcal{C}_\&$. The statement is a special case of a property that works for any category $\mathcal{D}$; analogously (and dually) to the previous lemma, it corresponds to the universal property of the product completion. $\qquad\square$

Before we can state the next lemma, we first need a few definitions. Let

$$\iota_{\&}^{\oplus} : \mathcal{C}_\oplus \to \mathcal{C}_{\oplus\&} \quad \text{such that} \quad \iota_{\oplus\&} = \iota_{\&}^{\oplus} \circ \iota_\oplus$$

be the full and faithful embedding be obtained by applying the universal property of $\mathcal{C}_\oplus$ to the coproduct-preserving functor $\iota_{\oplus\&}$. Furthermore, the isomorphism $\mathcal{C}_{\oplus\&} \cong (\mathcal{C}_\&)_\oplus$ entails the existence of another full and faithful functor

$$\iota_\oplus' : \mathcal{C}_\& \to \mathcal{C}_{\oplus\&} \quad \text{such that} \quad \iota_{\oplus\&} = \iota_\oplus' \circ \iota_\&$$

corresponding to the canonical embedding $\mathcal{D} \to \mathcal{D}_\oplus$ for $\mathcal{D} = \mathcal{C}_\&$. Both those functors have straightforward explicit constructions, whose actions on objects are

$$\iota_{\&}^{\oplus}\left(\bigoplus_{i\in I} C_i\right) \;=\; \bigoplus_{i\in I}\mathop{\text{\Large\&}}_{j\in\{*\}} C_i \qquad \iota_\oplus'\left(\mathop{\text{\Large\&}}_{i\in I} C_i\right) \;=\; \bigoplus_{j\in\{*\}}\mathop{\text{\Large\&}}_{i\in I} C_i$$

(where the (co)product notation is syntactic sugar for indexed families).

Lemma D.3. *For any $D \in \mathsf{Obj}(\mathcal{C}_\oplus)$, there is a natural isomorphism (with domain $\mathcal{C}_\&$)*

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota_\oplus'(-), \iota_{\&}^{\oplus}(D)\right) \quad\cong\quad \langle\&\rangle\left[\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(-), D\right)\right]$$

Proof. Let $C = \mathop{\text{\Large\&}}_{i\in I} C_i$ and $D = \bigoplus_{k\in K} D_k$ for $C_i, D_k \in \mathsf{Obj}(\mathcal{C})$. By definition,

$$\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus'(C), \iota_{\&}^{\oplus}(D)\right) = \prod_{p\in\{*\}}\sum_{k\in K}\prod_{q\in\{*\}}\sum_{i\in I}\mathsf{Hom}_\mathcal{C}\left(C_i, D_k\right)$$

$$\cong \sum_{i\in I}\sum_{k\in K}\mathsf{Hom}_\mathcal{C}\left(C_i, D_k\right)$$

$$\cong \sum_{i\in I}\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(C_i), D\right)$$

This establishes the isomorphism for each object of $\mathcal{C}_\&$. The naturality condition can be recast as the commutation of the following diagram:

$$
\begin{array}{ccccc}
\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota_\oplus'(C), \iota_{\&}^{\oplus}(D)\right) & \xleftarrow{\ \sim\ } & \displaystyle\sum_{i,k}\mathsf{Hom}_\mathcal{C}\left(C_i, D_k\right) & \xrightarrow{\ \sim\ } & \displaystyle\sum_{i\in I}\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(C_i), D\right) \\
{\scriptstyle -\circ f}\Big\downarrow & & & & \Big\downarrow{\scriptstyle (i,g)\mapsto(\varphi_f(i), g\circ f_i)} \\
\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota_\oplus'(C'), \iota_{\&}^{\oplus}(D)\right) & \xrightarrow{\ \sim\ } & \displaystyle\sum_{j,k}\mathsf{Hom}_\mathcal{C}\left(C_j', D_k\right) & \xleftarrow{\ \sim\ } & \displaystyle\sum_{j\in J}\mathsf{Hom}_{\mathcal{C}_\oplus}\left(\iota_\oplus(C_j'), D\right)
\end{array}
$$

Fortunately, it can be checked from the definition of composition in $\mathcal{C}_\oplus$ and $\mathcal{C}_{\oplus\&}$ that both paths in this diagrams denote the same map, namely $(i, k, h) \mapsto (\varphi_f(i), k, h \circ f_i)$. $\qquad\square$

Proof of Theorem 3.53. Let $A, B, C \in \mathsf{Obj}(\mathcal{C}_{\oplus\&})$. By definition, we can write

$$A = \bigoplus_{u\in U}\mathop{\text{\Large\&}}_{x\in X_u} A_x \qquad B = \bigoplus_{v\in V}\mathop{\text{\Large\&}}_{y\in Y_v} B_y \qquad C = \bigoplus_{w\in W}\mathop{\text{\Large\&}}_{z\in Z_w} C_z$$

where $A_x, B_y, C_z \in \mathsf{Obj}(\mathcal{C})$. Using the definition of $\otimes$ in $\mathcal{C}_{\oplus\&}$ and of $\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(-,-)$,

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(C \otimes A, B) = \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\bigoplus_{(w,u)\in W\times U}\underset{(z,x)\in Z_w\times X_u}{\&} C_z \otimes A_x, \; \bigoplus_{v\in V}\underset{y\in Y_v}{\&} B_y\right)$$

$$= \prod_{w,u}\sum_{v\in V}\prod_{y\in Y_v}\sum_{z,x}\mathsf{Hom}_{\mathcal{C}}(C_z \otimes A_x, \; B_y)$$

$$\cong \prod_{w\in W}\prod_{u\in U}\sum_{v\in V}\prod_{y\in Y_v}\sum_{x\in X_u}\sum_{z\in Z_w}\mathsf{Hom}_{\mathcal{C}}(C_z \otimes A_x, \; B_y)$$

Using the operators from Lemmas D.1 and D.2, the last expression above can be turned into a functor in C, and the isomorphism is then natural in C, that is:

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(- \otimes A, B) \quad \cong \quad \langle\oplus\rangle\left[\prod_{u\in U}\sum_{v\in V}\prod_{y\in Y_v}\sum_{x\in X_u}\langle\&\rangle\left[\mathsf{Hom}_{\mathcal{C}}(- \otimes A_x, \; B_y)\right]\right]$$

(Any reader who is not convinced that rearranging the indexing of sums/products is innocuous with respect to the morphisms in $\mathcal{C}_{\oplus\&}$ can check the naturality by a brute-force computation.)

For any $A', B' \in \mathsf{Obj}(\mathcal{C})$, we have a sequence of natural isomorphisms

$$\mathsf{Hom}_{\mathcal{C}}(- \otimes A', B') \cong \mathsf{Hom}_{\mathcal{C}_{\oplus}}(\iota_{\oplus}(- \otimes A'), \iota_{\oplus}(B'))$$

$$\cong \mathsf{Hom}_{\mathcal{C}_{\oplus}}(\iota_{\oplus}(-) \otimes \iota_{\oplus}(A'), \iota_{\oplus}(B'))$$

$$\cong \mathsf{Hom}_{\mathcal{C}_{\oplus}}(\iota_{\oplus}(-), \iota_{\oplus}(A') \multimap \iota_{\oplus}(B'))$$

by using the strong monoidality and full faithfulness of $\iota_{\oplus}$ as well as the assumption on internal homsets in $\mathcal{C}_{\oplus}$. Lemma D.1 allows us to lift natural isomorphisms through $\langle\&\rangle$; combined with Lemma D.3, this gives us

$$\langle\&\rangle\left[\mathsf{Hom}_{\mathcal{C}}(- \otimes A', B')\right] \cong \langle\&\rangle\left[\mathsf{Hom}_{\mathcal{C}_{\oplus}}(\iota_{\oplus}(-), \iota_{\oplus}(A') \multimap \iota_{\oplus}(B'))\right]$$

$$\cong \mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(\iota'_{\oplus}(-), \iota_{\&}^{\oplus}(\iota_{\oplus}(A') \multimap \iota_{\oplus}(B')))$$

Let us introduce the abbreviation $A' \Rightarrow B' = \iota_{\&}^{\oplus}(\iota_{\oplus}(A') \multimap \iota_{\oplus}(B'))$. Using Proposition 3.28 and the dual of Remark 3.29 for products, we have

$$\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(- \otimes A, B) \cong \langle\oplus\rangle\left[\prod_{u\in U}\sum_{v\in V}\prod_{y\in Y_v}\sum_{x\in X_u}\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota'_{\oplus}(-), \; A_x \Rightarrow B_y\right)\right]$$

$$\cong \langle\oplus\rangle\left[\prod_{u\in U}\sum_{v\in V}\prod_{y\in Y_v}\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota'_{\oplus}(-), \; \bigoplus_{x\in X_u} A_x \Rightarrow B_y\right)\right]$$

$$\cong \cdots$$

$$\cong \langle\oplus\rangle\left[\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}\left(\iota'_{\oplus}(-), \; \overset{A\multimap B}{\overbrace{\underset{u\in U}{\&}\bigoplus_{v\in V}\underset{y\in Y_v}{\&}\bigoplus_{x\in X_u} A_x \Rightarrow B_y}}\right)\right]$$

To conclude, it suffices to show that the last functor in the above computation is naturally isomorphic to $\mathsf{Hom}_{\mathcal{C}_{\oplus\&}}(-, A \multimap B)$. The isomorphism can seen as an instance of the universal

property of the coproduct, cf. Remark 3.29 which unfortunately only states naturality in the "wrong" argument for our purposes. The remaining naturality condition can be verified routinely; we leave it to the reader. □

Appendix E. Proof of Theorem 3.17 (building functors from $\mathcal{SR}$)

Let us fix a an symmetric monoidal category $(\mathcal{C}, \otimes, \mathbf{I})$ with an internal monoid (M, μ, η). We call the functors built from the monoidal structure on $\mathcal{C}$ "tensorial functors" since the monoidal product $\otimes$ is sometimes called a tensor product. They play a central role in our proof and are more precisely defined as follows.

Definition E.1. The *tensorial expressions* over a finite indexing set I are freely inductively generated as follows:
- $\mathbf{I}$ is a tensorial expression over $\varnothing$;
- i is a tensorial expression over $\{i\}$;
- if e, e' are tensorial expressions over I and I' respectively, with $I \cap I' = \varnothing$, then $(e \otimes e')$ is a tensorial expression over $I \cup I'$.

In other words, a tensorial expression over I is a binary tree whose leaves are labeled either by $\mathbf{I}$ or by $i \in I$, such that each element i appears exactly once.

A tensorial expression e over I induces a functor $F_e : \mathcal{C}^I \to \mathcal{C}$ in an obvious way. The functors thus obtained are called *tensorial functors*. A tensorial functor $F_e : \mathcal{C}^I \to \mathcal{C}$ is *ordered* when I is endowed with the total order that corresponds to the infix order on the I-labeled leaves of the expression e.

Of course, the basic intuition is that over a given totally ordered indexing set, two ordered tensorial functors express "the same thing" up to inessential bracketing, while two unordered tensorial functors should be morally the same "up to permutation". This is expressed formally as a natural isomorphism between functors, but *Mac Lane's coherence theorem* gives us something stronger: the natural isomorphisms can be chosen *canonically*.

Theorem E.2 (Coherence for symmetric monoidal categories [ML98, §XI.1 (Theorem 1)])**.** *There exists a map that sends each triple (I, e, e'), where I is a finite set and e, e' are tensorial expressions over I, to a natural isomorphism from the tensorial functor F_e to $F_{e'}$, which we call a* canonical isomorphism*, such that:*
- *identities, associators, unitors and symmetries are canonical isomorphisms, i.e. are in the image of this map;*
- *composing the image of (I, e, e') by this map with the image of (I, e', e'') yields the image of (I, e, e'') – in other words, canonical isomorphisms are closed under composition;*
- *canonical isomorphisms are also closed under monoidal product.*

If $F, G : \mathcal{C}^I \to \mathcal{C}$ are *ordered* tensorial functors for the same total order on I, the construction of the canonical isomorphism between F and G works in any monoidal category, not necessarily symmetric. Indeed, general monoidal categories enjoy a coherence theorem of their own [ML98, §VII.2], which involves its own canonical isomorphisms; thanks to the uniqueness clauses in the coherence theorems with and without symmetry, one can check that the two notions of canonical isomorphism coincide for ordered tensorial functors. The ordered case is important for us because even though our monoidal category is symmetric, the internal monoid (M, μ, η) that we are given need not be *commutative*. What the axioms

for monoid objects *do* state, however, is a suitable form of associativity, a consequence of which is that *n*-ary products are somehow "independent of bracketing". Formally speaking:

Definition E.3. To any tensorial expression e, we associate an "e-fold monoid multiplication" morphism $\mu^{(e)} : F_e((M)_{i \in I}) \to M$ inductively:

$$\mu^{(\mathbf{I})} = \eta \qquad \mu^{(i)} = \text{id for } I = \{i\} \qquad \mu^{(e \otimes e')} = \mu \circ \left(\mu^{(e)} \otimes \mu^{(e')} \right)$$

Theorem E.4 (General associativity law [ML98, §VII.3]). *Let I be a totally ordered finite set, $F_e, F_{e'} : \mathcal{C}^I \to \mathcal{C}$ be two ordered tensorial functors and $\Xi : F_e \Rightarrow F_{e'}$ be their canonical natural isomorphism. Then $\mu^{(e)} = \mu^{(e')} \circ \Xi_{\vec{M}}$ where $\vec{M} = (M)_{i \in I}$.*

We find it convenient to work directly with tensorial functors in the rest of this appendix, leaving the tensorial expressions that define them implicit. Therefore, we write $\mu^{(F)}$ instead of $\mu^{(e)}$ when $F = F_e$; strictly speaking, two expressions could define the same functor for accidental reasons, but for our purposes, the right choice of e can always be inferred from the context. Similarly, we shall speak of *the* canonical isomorphism between two tensorial functors $\mathcal{C}^I \to \mathcal{C}$. In accordance with Section 2.1, we write

$$\bigotimes_{i=1}^{n} Y_i = (\dots (Y_1 \otimes Y_2) \otimes \dots) \otimes Y_n \qquad M^{\otimes n} = \bigotimes_{i=1}^{n} M$$

and this will be used to define ordered tensorial functors. Furthermore, for each finite set I, we fix an arbitrary choice of tensorial functor $\bigotimes_{i \in I}(-) : \mathcal{C}^I \to \mathcal{C}$, and denote by $\bigotimes_{i \in I} Y_i$ the image of $(Y_i)_{i \in I}$ by this functor.

After these general preliminaries, let us focus on the specific study of the category $\mathcal{SR}(\Gamma)$ for a fixed finite alphabet Γ.

Definition E.5. For $t \in [R \to_{\mathcal{SR}} R']$ – recall that this implies $t : R' \to (\Gamma + R)^*$, let $\partial(t) \subseteq R$ be the set of register variables that do not occur in any $t(r')$ for $r' \in R'$.

Definition E.6. Given $w \in (\Gamma + R)^*$, we write

$$\bigotimes_{r \leftarrow w} Y_r = \bigotimes_{i=1}^{|w|} \begin{cases} \mathbf{I} & \text{when } w[i] \in \text{in}_1(\Gamma) \\ Y_r & \text{when } w[i] = \text{in}_2(r) \text{ for } r \in R \end{cases} \qquad \text{and} \qquad M^{\leftarrow w} = \bigotimes_{r \leftarrow w} M$$

Lemma E.7. *For $t \in [R \to_{\mathcal{SR}} R']$, we have a canonical isomorphism*

$$\bigotimes_{r \in R} Y_r \quad \cong \quad \bigotimes_{r \in \partial(t)} Y_r \otimes \bigotimes_{r' \in R'} \bigotimes_{r \leftarrow t(r')} Y_r$$

Proof. To apply Mac Lane's coherence theorem, we just have to check that the right-hand side defines a tensorial functor with indexing set R. This amounts to the equality

$$R = \partial(t) \cup \bigcup_{r' \in R'} \{r \mid \exists i.\, t(r')[i] = \text{in}_2(r)\}$$

where, to ensure that no index in R is repeated, the union must be *disjoint* and the letters of $t(r')$ that are in $\text{in}_2(R)$ must all be distinct. Those conditions are consequences of copylessness, while the equality itself is essentially the definition of $\partial(t)$. $\square$

We are now in a position to build the functor promised in Theorem 3.17. Following the statement of this theorem, we fix a family of morphisms $(m_c) \in \text{Hom}_{\mathcal{C}}(\mathbf{I}, M)^{\Gamma}$, and assume that $(\mathcal{C}, \otimes, \mathbf{I})$ is affine. Thus, we may write $\langle \rangle_A : A \to \mathbf{I}$ for the terminal morphism from A, omitting the subscript A when it can be inferred from the context.

Definition E.8. We define a map on objects $F : R \in \mathsf{Obj}(\mathcal{SR}) \mapsto M^{\otimes R} \in \mathsf{Obj}(\mathcal{C})$.

As for morphisms, given a register transition $t \in [R \to_{\mathcal{SR}} R']$, we set $F(t)$ to be

$$F(R) = M^{\otimes R} \xrightarrow{\sim} M^{\otimes \partial(t)} \otimes \bigotimes_{r' \in R'} \bigotimes_{i=1}^{|t(r')|} X_{r',i} \xrightarrow{\langle\rangle \otimes \widetilde{F}(t)} \mathbf{I} \otimes M^{\otimes R'} \xrightarrow{\sim} M^{\otimes R'} = F'(R)$$

where the left arrow instantiates the canonical isomorphism of Lemma E.7, with

$$X_{r',i} = \begin{cases} \mathbf{I} & \text{when } t(r')[i] \in \mathsf{in}_1(\Gamma) \\ M & \text{otherwise, i.e. } t(r')[i] \in \mathsf{in}_2(R) \end{cases} \qquad \text{so that} \qquad \bigotimes_{i=1}^{|t(r')|} X_{r',i} = M^{\leftharpoonup \rho t(r')}$$

and in the middle arrow, $\langle\rangle : M^{\otimes \partial(t)} \to \mathbf{I}$ is the terminal morphism and

$$\widetilde{F}(t) = \bigotimes_{r' \in R'} \left(\widetilde{F}(t)_{r'} : \bigotimes_{i=1}^{|t(r')|} X_{r',i} \xrightarrow{\otimes_i f_{r',i}} \bigotimes_{i=1}^{|t(r')|} M = M^{\otimes |t(r')|} \xrightarrow{\mu^{(|t(r')|)}} M \right)$$

where for $i \in \{1, \ldots, |t(r')|\}$, we pick

$$f_{r',i} = \begin{cases} m_c & \text{when } t(r')[i] = \mathsf{in}_1(c) \text{ for } c \in \Gamma \\ \mathrm{id}_M & \text{otherwise, i.e. } t(r')[i] \in \mathsf{in}_2(R) \end{cases}$$

(recall that $m_c : \mathbf{I} \to M$ is the prescribed functorial image, by the F that we are defining, of the register transition $\widehat{c} \in [\varnothing \to_{\mathcal{SR}} \{\bullet\}]$).

The tedious part in proving Theorem 3.17 is checking that the above definition works.

Proposition E.9. *The operation F introduced in Definition E.8 is a functor.*

Proof. Let $t \in [R \to_{\mathcal{SR}} R']$ and $t' \in [R' \to_{\mathcal{SR}} R'']$; we want to reason on $F(t') \circ F(t)$ to show that it is equal to $F(t' \circ t)$. Beware: we write $t' \circ t$ for composition of (copyless) register transitions in the category $\mathcal{SR}$, and will employ the notation $t(r')$ for set-theoretic application $(t : R' \to (\Gamma + R)^*)$, but we do *not* have $(t' \circ t)(r') = t'(t(r'))$ – indeed, the two sides of the equality are not even well-defined for $r' \in R'$! Since $t' \circ t$ is in $[R \to_{\mathcal{SR}} R'']$, it is a set-theoretic map $R'' \to (\Gamma + R)^*$.

To prove this, we first reduce our goal to Lemma E.10, and then prove that lemma. The reduction is given by the commutative diagram of Figure 22 (with N to be defined later). Indeed, while the morphism on the top is $F(t') \circ F(t)$, the one defined by the three other sides of the outermost square is

$$M^{\otimes R} \xrightarrow{\sim} M^{\otimes \partial(t' \circ t)} \otimes \bigotimes_{r'' \in R''} M^{\leftharpoonup \rho(t' \circ t)(r'')} \xrightarrow{\langle\rangle \otimes \widetilde{F}(t' \circ t)} \mathbf{I} \otimes M^{\otimes R''} \xrightarrow{\sim} M^{\otimes R''}$$

thanks to the closure of canonical isomorphisms of tensorial functors under composition. Since these canonical isomorphisms are also unique, we can equate that with the expression for $F(t' \circ t)$ given in Definition E.8, hence $F(t') \circ F(t) = F(t' \circ t)$.

The main justifications that are missing from Figure 22 are the commutation of one square treated in Lemma E.10, and the existence of the various canonical isomorphisms involved. The ones in the right column are just unitors, so let us focus on the left column.

- The top left isomorphism is an instance of Lemma E.7.
- The next one (skipping the equality) is provided by Lemma E.10.

$$M^{\otimes R} \xrightarrow{\quad F(t) \quad} M^{\otimes R'} \xrightarrow{\quad F(t') \quad} M^{\otimes R''}$$

$$\downarrow{\sim} \qquad \text{definition of } F(t) \qquad \sim{\uparrow} \qquad \text{naturality} \qquad \sim{\uparrow}$$

$$M^{\otimes \partial(t)} \otimes \bigotimes_{r' \in R'} M^{\leftarrow\!\rho t(r')} \xrightarrow{\langle\rangle \otimes \widetilde{F}(t)} \mathbf{I} \otimes M^{\otimes R'} \xrightarrow{\mathbf{I} \otimes F(t')} \mathbf{I} \otimes M^{\otimes R''}$$

$$\| \qquad\qquad \text{bifunctoriality of } \otimes \qquad\qquad \|$$

$$M^{\otimes \partial(t)} \otimes \bigotimes_{r' \in R'} M^{\leftarrow\!\rho t(r')} \xrightarrow{\;\langle\rangle \otimes (F(t') \circ \widetilde{F}(t))\;} \mathbf{I} \otimes M^{\otimes R''}$$

$$\downarrow{\sim} \qquad \langle\rangle \otimes (\text{Lemma E.10 / Figure 23}) \qquad \sim{\uparrow}$$

$$M^{\otimes \partial(t)} \otimes \left(N \otimes \bigotimes_{r'' \in R''} M^{\leftarrow\!\rho(t \circ t')(r'')} \right) \xrightarrow{\;\langle\rangle \otimes (\langle\rangle \otimes \widetilde{F}(t' \circ t))\;} \mathbf{I} \otimes \left(\mathbf{I} \otimes M^{\otimes R''} \right)$$

$$\downarrow{\sim} \qquad \text{naturality of the associator} \qquad \sim{\uparrow}$$

$$\left(M^{\otimes \partial(t)} \otimes N \right) \otimes \bigotimes_{r'' \in R''} M^{\leftarrow\!\rho(t \circ t')(r'')} \xrightarrow{\;(\langle\rangle \otimes \langle\rangle) \otimes \widetilde{F}(t' \circ t)\;} (\mathbf{I} \otimes \mathbf{I}) \otimes M^{\otimes R''}$$

$$\downarrow{\sim} \qquad \mathbf{I} \otimes \mathbf{I} \text{ and } \mathbf{I} \text{ are terminal (affineness assumption)} \qquad \sim{\uparrow}$$

$$M^{\otimes \partial(t' \circ t)} \otimes \bigotimes_{r'' \in R''} M^{\leftarrow\!\rho(t \circ t')(r'')} \xrightarrow{\;\langle\rangle \otimes \widetilde{F}(t' \circ t)\;} \mathbf{I} \otimes M^{\otimes R''}$$

Figure 22: The commutativity of the outer square of this diagram establishes Proposition E.9. The text in the inner squares explains why they commute; the proof text for Proposition E.9 defines N and gives further justifications (in particular for the existence of the canonical isomorphisms denoted by $\xrightarrow{\sim}$).

- Then the penultimate one is just the inverse of an associator (see Section 2.4), of the form

$$\alpha_{A,B,C}^{-1} \; : \; A \otimes (B \otimes C) \; \to \; (A \otimes B) \otimes C$$

- Finally, the last one reduces to $M^{\otimes \partial(t)} \otimes N \cong M^{\otimes \partial(t' \circ t)}$.

To prove the latter, we must first clarify it by defining N consistently with Lemma E.10:

$$N = \bigotimes_{r' \in \partial(t')} M^{\leftarrow\!\rho t(r')}$$

$$
\begin{array}{ccc}
\displaystyle\bigotimes_{r'\in R'} M^{\leftarrow t(r')} & \xrightarrow{\ \widetilde{F}(t)=\bigotimes_{r'\in R'}\widetilde{F}(t)_{r'}\ } M^{\otimes R'} & \xrightarrow{\ F(t')\ } M^{\otimes R''}
\end{array}
$$

$$\sim\quad \text{naturality / Lemma E.7} \qquad \sim \qquad \text{definition of } F(t') \qquad \sim$$

$$
N\otimes\bigotimes_{r''\in R''} O_{r''} \xrightarrow{\ (\dots)\otimes\bigotimes_{r''}\psi_{r''}\ } M^{\otimes\partial(t')}\otimes\bigotimes_{r''\in R''} M^{\leftarrow t'(r'')} \xrightarrow{\ \langle\rangle\otimes\widetilde{F}(t')\ } \mathbf{I}\otimes M^{\otimes R''}
$$

$$\otimes \text{ is a bifunctor and } \mathbf{I} \text{ is a terminal object}$$

$$
N\otimes\bigotimes_{r''\in R''} O_{r''} \xrightarrow{\ \langle\rangle\otimes\bigotimes_{r''\in R''}(\widetilde{F}(t')_{r''}\circ\psi_{r''})\ } \mathbf{I}\otimes M^{\otimes R''}
$$

$$\sim\quad \text{manipulations on monoid objects, explained in main text}$$

$$
N\otimes\bigotimes_{r''\in R''} M^{\leftarrow(t\circ t')(r'')} \xrightarrow{\ \langle\rangle\otimes\bigotimes_{r''\in R''}\widetilde{F}(t'\circ t)_{r''}\ =\ \langle\rangle\otimes F(t'\circ t)\ } \mathbf{I}\otimes M^{\otimes R''}
$$

Figure 23: Lemma E.10 defines N, $O_{r''}$, and $\psi_{r''}$, and proves that this diagram commutes.

We then see that the canonical isomorphism that we are looking for is the instantiation to the constant family $(M)_{r\in\partial(t'\circ t)}$ of the following natural isomorphism in Y_r for $r\in\partial(t'\circ t)$:

$$
\bigotimes_{r\in\partial(t)} Y_r \ \otimes\ \bigotimes_{r'\in\partial(t')}\bigotimes_{r\leftarrow t(r')} Y_r \ \cong\ \bigotimes_{r\in\partial(t'\circ t)} Y_r
$$

This is a consequence of an elementary combinatorial fact:

$$
\partial(t)\cup\bigcup_{r'\in\partial(t')}\{r\mid\exists i\in\{1,\dots,|t(r')|\}.\,t(r')[i]=\mathsf{in}_2(r)\}\ =\ \partial(t'\circ t)
$$

Informally speaking, this means that a register is thrown away by $t'\circ t$ if and only if is either thrown away by t, or used by t to compute a value that is then discarded by t'. Furthermore, thanks to copylessness, the union is *disjoint* and for each r in the union over $r'\in\partial(t')$, there is a *single* (r',i) such that $t(r')[i]=\mathsf{in}_2(r)$; those properties are necessary to get the natural isomorphism. We leave a formal proof of this combinatorial identity to the reader.

This being done, all that remains to conclude our proof is to show Lemma E.10 below. □

Lemma E.10. *The diagram of Figure 23 commutes, with the following definitions:*

$$
N=\bigotimes_{r'\in\partial(t')} M^{\leftarrow t(r')} \qquad O_{r''}=\bigotimes_{r'\leftarrow t'(r'')} M^{\leftarrow t(r')} \qquad \psi_{r''}=\bigotimes_{r'\leftarrow t'(r'')}\widetilde{F}(t)_{r'}
$$

Proof. In addition to the information already given in Figure 23, there are two things to justify about this figure: the top-left naturality square, and the commutativity of the bottom

square. Let us tackle the former. Recall that Lemma E.7 gives us the canonical isomorphism

$$\bigotimes_{r' \in R'} Y_{r'} \quad \cong \quad \bigotimes_{r' \in \partial(t')} Y_{r'} \otimes \bigotimes_{r'' \in R''} \bigotimes_{r' \leftarrow t'(r'')} Y_{r'}$$

which is natural in $Y_{r'}$ for $r' \in R'$. By instantiating with $Y_{r'} = M^{\leftarrow t(r')}$, we get

$$\bigotimes_{r' \in R'} M^{\leftarrow t(r')} \quad \cong \quad \bigotimes_{r' \in \partial(t')} M^{\leftarrow t(r')} \otimes \bigotimes_{r'' \in R''} \bigotimes_{r' \leftarrow t'(r'')} M^{\leftarrow t(r')} \quad = \quad N \otimes \bigotimes_{r'' \in R''} O_{r''}$$

by definition of N and $O_{r''}$. Hence the vertical canonical isomorphism at the top left of Figure 23; as for the top middle, it is the instantiation of the same natural isomorphism with $Y_{r'} = M$, as already observed in Definition E.8. To make the top-left square a naturality square, we should then have

$$(\dots) \otimes \bigotimes_{r'' \in R''} \psi_{r''} \quad = \quad \bigotimes_{r' \in \partial(t')} \widetilde{F}(t)_{r'} \otimes \bigotimes_{r'' \in R''} \bigotimes_{r' \leftarrow t'(r'')} \widetilde{F}(t)_{r'}$$

which is indeed the case with our definition of $\psi_{r''}$.

Our next and final task is the commutativity of the bottom square of Figure 23. Thanks to the bifunctoriality of $\otimes$, it reduces to a simpler commutative diagram:

$$O_{r''} = \bigotimes_{r' \leftarrow t'(r'')} M^{\leftarrow t(r')} \xrightarrow{\widetilde{F}(t')_{r''} \circ \psi_{r''}} M \qquad \text{for } r'' \in R''$$

with the vertical map $\sim$ down to $M^{\leftarrow (t \circ t')(r'')}$ and the diagonal $\widetilde{F}(t' \circ t)_{r''}$.

Let $r'' \in R''$. To show that this indeed commutes, we start by writing out $\widetilde{F}(t')_{r''} \circ \psi_{r''}$ as

$$\bigotimes_{r' \leftarrow t'(r'')} M^{\leftarrow t(r')} \xrightarrow{\otimes_{r'} \widetilde{F}(t)_{r'}} M^{\leftarrow t'(r'')} = \bigotimes_{j=1}^{|t'(r'')|} X'_{r'',j} \xrightarrow{\otimes_j f'_{r'',j}} M^{\otimes |t'(r'')|} \xrightarrow{\mu^{(|t'(r'')|)}} M$$

where, analogously to the $X_{r',i}$ and $f_{r',i}$ involved in the definition of $\widetilde{F}(t)_{r'}$, we take

$$(X'_{r'',j}, \ (f'_{r'',j} : X'_{r'',j} \to M)) = \begin{cases} (\mathbf{I}, \ m_c) & \text{when } t'(r'')[j] = \mathsf{in}_1(c) \text{ for } c \in \Gamma \\ (M, \ \mathrm{id}_M) & \text{otherwise, i.e. } t'(r'')[j] \in \mathsf{in}_2(R') \end{cases}$$

The leftmost arrow in the above sequence is how we defined $\psi_{r''}$ in the lemma statement, while the composition of the two others equals $\widetilde{F}(t')_{r''}$ by definition.

To manipulate this, let us introduce a new notation:

$$\overset{\texttt{foldMap}}{\bigotimes_{w}} f \quad = \quad \bigotimes_{i=1}^{|w|} f(w_i) \qquad \text{for } w = w_1 \dots w_n \text{ and } f \text{ a function}$$

Note that in general, this may lead to monoidal products with repeated factors: there is no a priori guarantee that this defines a tensorial functor. The function f will often be expressed as a copairing using the following notation:

$$f : \begin{cases} \mathsf{in}_1(y) \mapsto f_1(y) \\ \mathsf{in}_2(z) \mapsto f_2(z) \end{cases} \quad \Longleftrightarrow \quad f : x \in Y + Z \mapsto \begin{cases} f_1(y) & \text{when } x = \mathsf{in}_1(y) \text{ for } y \in Y \\ f_2(z) & \text{when } x = \mathsf{in}_2(z) \text{ for } z \in Z \end{cases}$$

In the case that we are interested in right now, we have by functoriality of $\otimes$:

$$\widetilde{F}(t')_{r''} \circ \psi_{r''} \;=\; \mu^{(|t'(r'')|)} \circ \overset{\texttt{foldMap}}{\underset{t'(r'')}{\bigotimes}} \begin{cases} \mathsf{in}_1(c) \;\mapsto\; m_c \\ \mathsf{in}_2(r') \mapsto \widetilde{F}(t)_{r'} = \mu^{(|t(r')|)} \circ \overset{\texttt{foldMap}}{\underset{t(r')}{\bigotimes}} \begin{cases} \mathsf{in}_1(a) \mapsto m_a \\ \mathsf{in}_2(r) \mapsto \mathrm{id}_M \end{cases} \end{cases}$$

To simplify this, we introduce the following tensorial functor:

$$L_{r''}((Y_p)_{p \in P_{r''}}) \;=\; \overset{|t'(r'')|}{\underset{j=1}{\bigotimes}} \begin{cases} Y_{j,1} & \text{when } t'(r'')[j] \in \mathsf{in}_1(\Gamma) \\ \overset{|t(r')|}{\underset{i=1}{\bigotimes}} Y_{j,i} & \text{when } t'(r'')[j] = \mathsf{in}_2(r') \end{cases}$$

$$\text{where} \qquad P_{r''} \;=\; \overset{|t'(r'')|}{\underset{j=1}{\sum}} \begin{cases} \{1\} & \text{when } t'(r'')[j] \in \mathsf{in}_1(\Gamma) \\ \{1, \dots, |t(r')|\} & \text{when } t'(r'')[j] = \mathsf{in}_2(r') \end{cases}$$

(using the dependent sum operation, cf. Section 2.1, so that $P_{r''} \subset \mathbb{N}^2$). There is a unique total order on $P_{r''}$ that makes $L_{r''}$ into an *ordered* tensorial functor: the *lexicographical order* inherited from $\mathbb{N}^2$. Thus, we may meaningfully speak of $\mu^{(L_{r''})} : L_{r''}((M)_{p \in P_{r''}}) \to M$, the $L_{r''}$-ary monoid multiplication, whose inductive definition leads to:

$$\mu^{(L_{r''})} \;=\; \mu^{(|t'(r'')|)} \circ \overset{\texttt{foldMap}}{\underset{t'(r'')}{\bigotimes}} \begin{cases} \mathsf{in}_1(c) \;\mapsto\; \mu^{(1)} = \mathrm{id}_M \\ \mathsf{in}_2(r') \mapsto \mu^{(|t(r')|)} \end{cases}$$

Now that this is defined, we can state the following equation, that directly follows by functoriality from the previous expressions of $\widetilde{F}(t')_{r''} \circ \psi_{r''}$ and of $\mu^{(L_{r''})}$:

$$\widetilde{F}(t')_{r''} \circ \psi_{r''} \;=\; \mu^{(L_{r''})} \circ \overset{\texttt{foldMap}}{\underset{t'(r'')}{\bigotimes}} \begin{cases} \mathsf{in}_1(c) \;\mapsto\; m_c \\ \mathsf{in}_2(r') \mapsto \overset{\texttt{foldMap}}{\underset{t(r')}{\bigotimes}} \begin{cases} \mathsf{in}_1(a) \mapsto m_a \\ \mathsf{in}_2(r) \mapsto \mathrm{id}_M \end{cases} \end{cases}$$

For the next step, recall that by Definition 3.11, $(t' \circ t)(r'') = t^{\ddagger}(t'(r''))$ (which is *not* $t'(t(r''))$, as previously emphasized, since this '$\circ$' is composition in $\mathcal{SR}(\Gamma)$), where $t^{\ddagger}$ is the morphism of free monoids such that $t^{\dagger}(\mathsf{in}_1(c)) = \mathsf{in}_1(c)$ and $t^{\dagger}(\mathsf{in}_2(r')) = t(r')$. Thus, the j-th term of the dependent sum above is equal to $\{1, \dots, |t^{\dagger}(t'(r'')[j])|\}$, and from the morphism property

$$(t' \circ t)(r'') = t^{\ddagger}(t'(r'')) = t^{\ddagger}(t'(r'')[1]) \cdot \ldots \cdot t^{\ddagger}(t'(r'')[|t'(r'')|])$$

we obtain an bijection $\xi_{r''} : P_{r''} \xrightarrow{\sim} \{1, \dots, (t' \circ t)(r'')\}$ such that for all $(j, i) \in P_{r''}$,

$$(t' \circ t)(r'')[\xi(j, i)] = t^{\ddagger}(t'(r'')[j])[i] = \begin{cases} \mathsf{in}_1(c) & \text{when } t'(r'')[j] = \mathsf{in}_1(c) \\ t(r')[i] & \text{when } t'(r'')[j] = \mathsf{in}_2(r') \end{cases}$$

Thanks to this, we have

$$\widetilde{F}(t')_{r''} \circ \psi_{r''} \;=\; \mu^{(L_{r''})} \circ L_{r''}\left(\left(\begin{cases} m_c & \text{when } (t' \circ t)(r'')[\xi(p)] = \mathsf{in}_1(c) \text{ for } c \in \Gamma \\ \mathrm{id}_M & \text{otherwise, i.e. } (t' \circ t)(r'')[\xi(p)] \in \mathsf{in}_2(R) \end{cases} \right)_{p \in P_{r''}} \right)$$

It is not hard to see that the bijection ξ is *monotone* (taking the lexicographical order on $P_{r''}$ as we did earlier). Therefore, the natural isomorphism

$$\Xi_{(Y_p)_{p \in P_{r''}}} : L\left((Y_p)_{p \in P_{r''}}\right) \xrightarrow{\sim} \bigotimes_{k=1}^{|(t' \circ t)(r'')|} Y_{\xi^{-1}(k)}$$

induced by ξ is a *canonical isomorphism of ordered tensorial functors* (i.e. it only uses associators and unitors, not symmetries). Thus, we may apply the general associativity law for internal monoids (Theorem E.4): $\mu^{(L_{r''})} = \mu^{(|(t' \circ t)(r'')|)} \circ \Xi$ (we omit the subscript of Ξ for convenience). By naturality of Ξ, we then have

$$\widetilde{F}(t')_{r''} \circ \psi_{r''} \quad = \quad \mu^{(|(t' \circ t)(r'')|)} \circ \left(\bigotimes_{(t' \circ t)(r'')}^{\texttt{foldMap}} \begin{cases} \mathsf{in}_1(c) \mapsto m_c \\ \mathsf{in}_2(r) \mapsto \mathrm{id}_M \end{cases} \right) \circ \Xi$$

By definition, this is equal to $\widetilde{F}(t' \circ t) \circ \Xi$, which is what we needed to conclude the proof. $\square$

Proposition E.11. *The functor F is strong monoidal.*

Proof. From the definition, we immediately get

$$F(R + R') = M^{\otimes(R+R')} \cong M^{\otimes R} \otimes M^{\otimes R'} = F(R) \otimes F(R')$$

as an instance of a canonical isomorphism

$$\bigotimes_{x \in R+R'} Y_x \quad \cong \quad \bigotimes_{r \in R} Y_{\mathsf{in}_1(r)} \otimes \bigotimes_{r' \in R'} Y_{\mathsf{in}_2(r')}$$

which is natural in Y_x for $x \in R + R'$. One can then verify that this family of isomorphisms $F(R + R') \cong F(R) \otimes F(R')$ is natural in R and R'. $\square$

To finish proving Theorem 3.17, it suffices to carry out some short explicit computations to check that this functor F satisfies the claimed equalities. We leave this to the reader.

Appendix F. Equivalence with $\lambda\ell^{\&}$-definable tree functions

One natural question is whether of Theorems 1.1 and 1.2 for variations of $\lambda\ell^{\oplus\&}$. Indeed, we expect that for strings, the equivalence between $\lambda\ell^{\oplus\&}$-definability and regular functions still holds if we forbid $\oplus$ and $\&$ in the former (see our discussion in [NP20, Claim 6.2]). We do not attempt to prove this here, as we expect this would require a notable development exploiting the Krohn-Rhodes theorem. However, calling $\lambda\ell^{\&}$ the restriction of $\lambda\ell^{\oplus\&}$ forbidding the use of sum types, we have a straightforward proof of the following.

Theorem F.1. *Tree and string functions definable in $\lambda\ell^{\&}$ are exactly the regular functions.*

Of course, since $\lambda\ell^{\oplus\&}$ is more expressive than $\lambda\ell^{\&}$, one inclusion follows from our main theorems. The converse can be done at the level of streaming settings, by first considering the analogue of $\mathcal{L}$ and $\mathfrak{L}$ for $\lambda\ell^{\&}$ and using a trick reminiscent of *continuation passing style transformation* [Rey93] in order to simulate coproducts with a combination of products and higher-order functions.

We do not spell out the full details of this proof, but give the key lemmas, both in the case of strings and trees.

Lemma F.2. *Let $\mathfrak{C}$ be a symmetric monoidal closed string streaming setting with products such that $\top$ is the monoidal unit. There is a streaming setting morphism $\mathfrak{C}_\oplus \to \mathfrak{C}$.*

Proof. First note that the "negation" $N(C) = C \multimap \perp\!\!\!\perp$ is a contravariant functor $\mathcal{C} \to \mathcal{C}^{\mathrm{op}}$ and that the coproduct completion $(-)_\oplus$ is itself functorial over Cat. We thus define the underlying functor $NN : \mathcal{C}_\oplus \to \mathcal{C}$ of our streaming setting morphism as a composite

$$\mathcal{C}_\oplus \xrightarrow{\quad N_\oplus \quad} (\mathcal{C}^{\mathrm{op}})_\oplus \xrightarrow{\qquad\qquad} \mathcal{C}^{\mathrm{op}} \xrightarrow{\quad N \quad} \mathcal{C}$$

where the middle arrow is obtained with by the universal property of the free completion (recall that products of $\mathcal{C}$ become coproducts of $\mathcal{C}^{\mathrm{op}}$). On objects we thus have

$$N \quad : \quad \bigoplus_{u \in U} C_u \quad \longmapsto \quad \left(\bigwith_{u \in U} C_u \multimap \perp\!\!\!\perp \right) \multimap \perp\!\!\!\perp$$

The map $i : \mathbf{I} \to \iota_\oplus((\mathbf{I} \multimap \perp\!\!\!\perp) \multimap \perp\!\!\!\perp)$ is induced by the transposition of the evaluation map $\Lambda(\mathrm{ev} \circ \gamma)$ and $o : \iota_\oplus((\perp\!\!\!\perp \multimap \perp\!\!\!\perp) \multimap \perp\!\!\!\perp) \to \perp\!\!\!\perp$ is the evaluating of its argument on the constant map $\tilde{\mathrm{id}} : \mathbf{I} \to \perp\!\!\!\perp \multimap \perp\!\!\!\perp$. To check that this triple is indeed a morphism of streaming settings, it suffices to prove that the following diagram commutes in $\mathcal{C}$ for any $f \in \mathsf{Hom}_\mathcal{C}(\mathbf{I}, \perp\!\!\!\perp)$:

$$
\begin{array}{ccc}
\mathbf{I} & \xrightarrow{\quad f \quad} & \perp\!\!\!\perp \\
{\scriptstyle \Lambda(\mathrm{ev} \circ \gamma)} \downarrow & & \uparrow {\scriptstyle \mathrm{ev} \circ (\mathrm{id} \otimes \tilde{\mathrm{id}}) \circ \rho^{-1}} \\
(\mathbf{I} \multimap \perp\!\!\!\perp) \multimap \perp\!\!\!\perp & \xrightarrow{\quad N(N(f)) \quad} & (\perp\!\!\!\perp \multimap \perp\!\!\!\perp) \multimap \perp\!\!\!\perp
\end{array}
$$

Unravelling the defininition of $N(N(f))$, this follows from the elementary equational properties of symmetric monoidal categories. $\qquad\square$

Lemma F.3. *Let $\mathfrak{C}$ be a symmetric monoidal closed tree streaming setting with products such that $\top$ is the monoidal unit. There is a streaming setting morphism $\mathfrak{C}_\oplus \to \mathfrak{C}$.*

Proof idea. The underlying functor NN is exactly the same. To complete the construction in the case of trees, one needs to append a suitable lax monoidal structure to NN, that is, maps

$$\mathbf{I} \xrightarrow{\quad m^0 \quad} NN(\mathbf{I}) \qquad \text{and} \qquad NN(A) \otimes NN(B) \xrightarrow{\quad m^1_{A,B} \quad} NN(A \otimes B)$$

with $m^1_{A,B}$ natural in A and B and such that the diagrams in [Mel09, Section 5.1] commute; we only sketch the definitions of those maps. For $m^0 : \mathbf{I} \to ((\mathbf{I} \multimap \perp\!\!\!\perp) \multimap \perp\!\!\!\perp)$, we may take the usual $\Lambda(\mathrm{ev} \circ \gamma)$. As for $m^2_{-,-}$ for the objects $\bigoplus_{i \in I} A_i$ and $\bigoplus_{j \in J} B_j$, it should be a map

$$\left(\left(\bigwith_i A_i \multimap \perp\!\!\!\perp \right) \multimap \perp\!\!\!\perp \right) \otimes \left(\left(\bigwith_j B_j \multimap \perp\!\!\!\perp \right) \multimap \perp\!\!\!\perp \right) \longrightarrow \left(\left(\bigwith_{i,j} A_i \otimes B_j \multimap \perp\!\!\!\perp \right) \multimap \perp\!\!\!\perp \right)$$

There are two distinct options one might take, which intuitively correspond to the two intuitive way of proving $\neg\neg A \otimes \neg\neg B \vdash \neg\neg(A \otimes B)$ in linear logic; one can proceed for instance with the one which is biased towards the left. $\qquad\square$