

# Applicative Functors with Strings

Shunsuke Sogame

April 10, 2016

## 1 Introduction

We will show how applicative functors are depicted in *string diagrams*. Don't trust my poor mathematics. Any correction is welcome at [github.com/okomok/strcat](https://github.com/okomok/strcat).

## 2 String Diagrams

We introduce *string diagrams*, which are useful for category theory. Don't be afraid. A string diagram in this document is just a kind of expression trees.

### 2.1 Vertical Composition

First we define how to join strings.

**Definition 2.1** A type  $a$  is depicted as a string:

$$\begin{array}{c} | \\ a \end{array}$$

Type names are often omitted.

**Definition 2.2** A function is depicted as a node:

$$\begin{array}{c} | \\ b \\ \boxed{f} \\ a \end{array} := f :: a \rightarrow b$$

**Definition 2.3** An identity function is indistinguishable from a type:

$$\begin{array}{c} | \\ a \\ \boxed{\text{id}} \\ a \end{array}$$

**Definition 2.4 (Vertical Composition)** The function composition joins strings:

$$\begin{array}{c} | \\ c \\ \boxed{g} \\ | \\ b \\ \boxed{f} \\ | \\ a \end{array} := \begin{array}{c} | \\ c \\ \boxed{g \cdot f} \\ | \\ a \end{array}$$

One can check that any diagram built upon these definitions has no ambiguity due to the famous laws:

$$\begin{aligned} h \cdot (g \cdot f) &= (h \cdot g) \cdot f \\ \text{id} \cdot f &= f \\ g \cdot \text{id} &= g \end{aligned}$$

**Definition 2.5 (Value)** Strings for the unit type  $()$  can be omitted so that a value  $x :: a$  is represented as

$$\begin{array}{c} | \\ a \\ \boxed{x} \end{array}$$

For example, a function application  $f x$  is depicted as

$$\begin{array}{c} | \\ b \\ \boxed{f} \\ | \\ a \\ \boxed{x} \end{array}$$

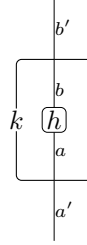
Due to the following definition, equations containing diagrams can often be simplified, known as *pointfree* style.

**Definition 2.6 (Function Equality)**

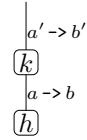
$$\begin{array}{c} | \\ b \\ \boxed{f} \\ | \\ a \\ \boxed{x} \end{array} =_{\forall x} \begin{array}{c} | \\ b \\ \boxed{g} \\ | \\ a \\ \boxed{x} \end{array} \iff \begin{array}{c} | \\ b \\ \boxed{f} \\ | \\ a \end{array} = \begin{array}{c} | \\ b \\ \boxed{g} \\ | \\ a \end{array}$$

## 2.2 Functors

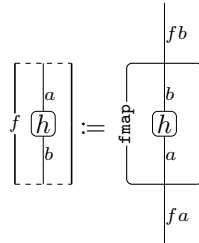
**Definition 2.7 (Functional Box)** Given a function  $k :: (a \rightarrow b) \rightarrow (a' \rightarrow b')$ , an application  $k\ h$  can be depicted as a *box*:



rather than

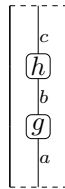


**Definition 2.8 (Functorial Tube)** Given a **Functor**  $f$ , an application of `fmap` can be depicted as a *tube* defined by



Tube names are often omitted.

The functor laws state that “tube then join” equals to “join then tube” so that any diagram like



has no ambiguity.

## 2.3 Horizontal Composition

We will make string diagrams two-dimensional, equipped with the *horizontal composition*.

**Definition 2.9** Parallel strings are pairs.

$$\left| \begin{array}{c} a_1 \\ a_2 \end{array} \right| := \left| (a_1, a_2) \right|$$

Owing to the trivial bijections

- $(a_1, (a_2, a_3)) \cong ((a_1, a_2), a_3)$
- $(a, ()) \cong a \cong ((), a)$

you can join any deeply nested pairs as far as their types are compatible, so that they are depicted as

$$\left| \begin{array}{c} a_1 \\ a_2 \\ a_3 \dots \\ a_n \end{array} \right|$$

without parentheses.

**Remark 2.10** Of course these bijections must be explicitly inserted to your haskell code.

**Definition 2.11 (Horizontal Composition)** Parallel nodes are defined by

$$\left| \begin{array}{cc} b_1 & b_2 \\ \boxed{f_1} & \boxed{f_2} \\ a_1 & a_2 \end{array} \right| := \backslash (a_1, a_2) \rightarrow (f_1 a_1, f_2 a_2)$$

With these definitions, it is easy to check that:

**Proposition 2.12 (Sliding)**

$$\left| \begin{array}{cc} b_1 & b_2 \\ \boxed{f_1} & \boxed{f_2} \\ a_1 & a_2 \end{array} \right| = \left| \begin{array}{cc} b_1 & b_2 \\ \boxed{f_1} & \boxed{f_2} \\ a_1 & a_2 \end{array} \right| = \left| \begin{array}{cc} b_1 & b_2 \\ \boxed{f_1} & \boxed{f_2} \\ a_1 & a_2 \end{array} \right|$$

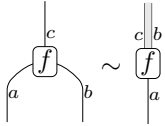
## 2.4 Currying

**Definition 2.13 (Band)** A special string for function types, a *band* is defined by

$$\left| \begin{array}{c} b \\ a \end{array} \right| := \left| a \rightarrow b \right|$$

Notice that the order of types is flipped. So we often write  $b \leftarrow a$  as  $a \rightarrow b$ .

**Definition 2.14 (Currying)** With bands, currying is represented by



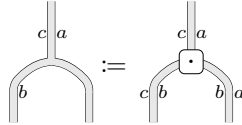
$$f \mapsto \lambda a \rightarrow \lambda b \rightarrow f(a, b)$$

$$\lambda(a, b) \rightarrow f a b \leftarrow f$$

We don't distinguish these two diagrams, because “move the right-side leg up and down” works correct in any form of diagrams.

The following definitions make bands cute.

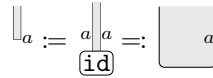
**Definition 2.15 (Function Composition)**



or you can use a *fat* form

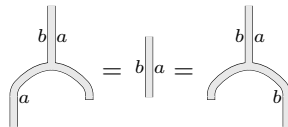


**Definition 2.16 (Identity Function)**

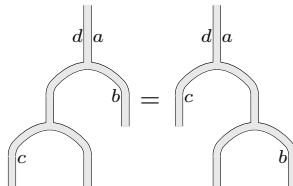


The following propositions are immediate.

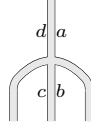
**Proposition 2.17 (Unitality)**



**Proposition 2.18 (Associativity)**



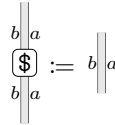
to which we assign



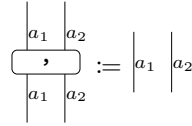
A band that has more forks is similarly defined. The equations for fat forms are left as an exercise.

For later use, we note the two famous operators.

**Definition 2.19 (Apply Operator)**

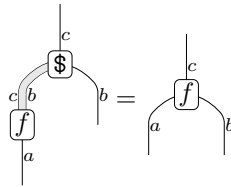


**Definition 2.20 (Comma Operator)**



One can check immediately:

**Proposition 2.21**

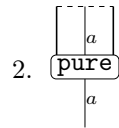


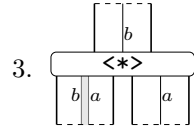
## 3 Applicative Functors

### 3.1 The Definition

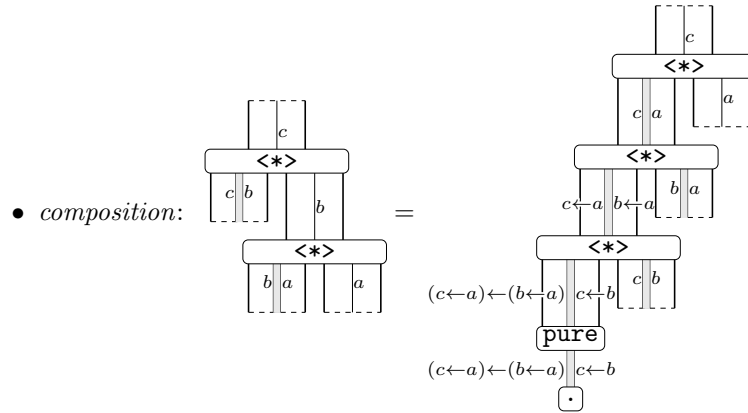
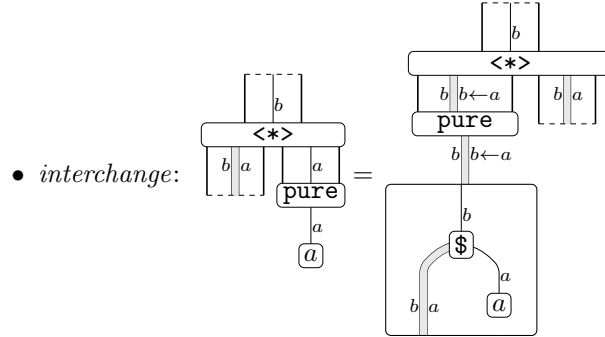
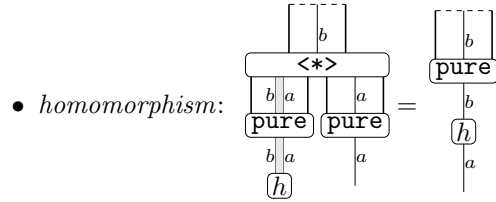
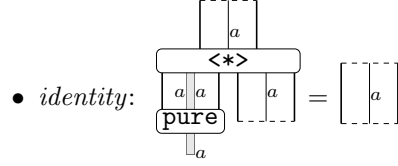
Using diagrams, an Applicative  $f$  consists of

1. Functor  $f$





satisfying the following laws, which we can never understand,



●

**Remark 3.1** The last law is redundant in case the *free theorem*[18] assumed

## 3.2 Lax Functors

To depict applicative functors cuter, we represent an applicative functor as a fork-able tube, which is called a *lax functor* in category theory.

### Definition 3.2

11

Under the applicative functor laws, one can check the following propositions that justify these pictures.

### Proposition 3.3 (Naturality)

1

### Proposition 3.4 (Unitality)

11

### Proposition 3.5 (Associativity)

1



to which we assign

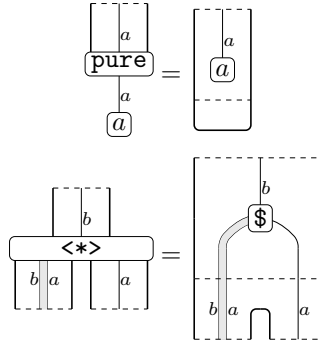


A tube that has more forks can be similarly defined without ambiguity.

**Remark 3.6** In our diagrams, cutoff lines are preferred to parentheses when we want to explicitly show bounds between components.

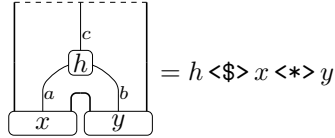
Finally one can find our goal:

**Proposition 3.7**



Thanks to these diagrams, you can immediately prove:

**Proposition 3.8 (Lift)**



## References

- [1] Jiri Adamek, Horst Herrlich, and George E Strecker. *Abstract and Concrete Categories: The Joy of Cats (Dover Books on Mathematics)*. Dover Publications, 8 2009.
- [2] John Armstrong. The "strictification" theorem. <https://unapologetic.wordpress.com/2007/07/01/the-strictification-theorem/>, 2007.
- [3] Steve Awodey. *Category Theory (Oxford Logic Guides)*. Oxford University Press, 2 edition, 8 2010.
- [4] John Baez. Classical vs quantum computation (week 5). [https://golem.ph.utexas.edu/category/2006/11/classical\\_vs\\_quantum\\_computati\\_5.html](https://golem.ph.utexas.edu/category/2006/11/classical_vs_quantum_computati_5.html), 2006.
- [5] Francis Borceux. *Handbook of Categorical Algebra: Volume 1, Basic Category Theory (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, 1 edition, 4 2008.
- [6] John Bourke and Micah Blake McCurdy. Frobenius morphisms of bicategories, 2009.
- [7] Ralf Hinze. Kan extensions for program optimisation or: Art and dan explain an old trick. In *Mathematics of Program Construction*, pages 324–362. Springer, 2012.
- [8] Max Kelly. *Basic Concepts of Enriched Category Theory (London Mathematical Society Lecture Note Series)*. Cambridge University Press, 4 1982.
- [9] Aleks Kissinger. Pictures of processes: automated graph rewriting for monoidal categories and applications to quantum computing. *arXiv preprint arXiv:1203.0202*, 2012.
- [10] Saunders Mac Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, 2nd ed. 1978. softcover reprint of the original 2nd ed. 1978 edition, 11 2010.
- [11] Dan Marsden. Category theory using string diagrams. *CoRR*, abs/1401.7220, 2014.
- [12] Conor McBride and Ross Paterson. Control.applicative. <https://hackage.haskell.org/package/base/docs/Control-Applicative.html>, 2005.
- [13] Conor McBride and Ross Paterson. Functional pearl: Applicative programming with effects. *Journal of functional programming*, 18(1):1–13, 2008.
- [14] Micah Blake McCurdy. Strings and stripes, graphical calculus for monoidal functors and monads, 2010.

- [15] Paul-André Mellies. Functorial boxes in string diagrams. In *Computer science logic*, pages 1–30. Springer, 2006.
- [16] Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.
- [17] Daniele Turi. Category theory lecture notes. *Laboratory for Foundations of Computer Science, University of Edinburgh*, 2001.
- [18] Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359. ACM, 1989.