Memoryless systems generate the class of all discrete systems

Erwan Beurier, Dominique Pastor, David I. Spivak

IMT Atlantique
Dépt. Signal & Communications
Technopôle de Brest-Iroise - CS 83818
29238 Brest Cedex 3
Téléphone: +33 (0)2 29 00 13 04
Télécopie: +33 (0)2 29 00 10 12
URL: **www.imt-atlantique.fr**

Erwan Beurier, IMT Atlantique, Lab-STICC, UBL, 29238 Brest, France
Dominique Pastor, IMT Atlantique, Lab-STICC, UBL, 29238 Brest, France
David I. Spivak, MIT, Cambrige, USA

Memoryless systems generate the class of all discrete systems

Collection des rapports de recherche d'IMT Atlantique

Date d'édition : 31 Mai 2018
Version du document : 1.1

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Summary

Abstract

Automata are machines, which receive inputs, accordingly update their internal state, and produce output, are a common abstraction for the basic building blocks used in engineering and science to describe and design complex systems. These arbitrarily simple machines can be wired together—so that the output of one is passed to another as its input—to form more complex machines. Indeed, both modern computers and biological systems can be described in this way, as assemblies of transistors or assemblies of simple cells. The complexity is in the network, i.e., the connection patterns between simple machines. The main result of this paper is to show that the range of simplicity for parts as compared to the complexity for wholes is in some sense complete: the most complex automaton can be obtained by wiring together direct-output memoryless components. The model we use—discrete-time automata sending each other messages from a fixed set of possibilities—is certainly more appropriate for computer systems than for biological systems. However, the result leads one to wonder what might be the simplest sort of machines, broadly construed, that can be assembled to produce the behavior found in biological systems, including the brain.

Keywords: automata, category theory, discrete system, memoryless system, monoidal category, wiring diagram.

Grants

David Spivak was supported by AFOSR grants FA9550-14-1-0031 and FA9550-17-1-0058, as well as NASA grant NNH13ZEA001N-SSAT while working on this project.

1. Introduction

Automata represent systems that receive inputs, alter their internal states, and produce outputs. The state set of the automaton is to be interpreted as the set of all potential memories, or storable experiences. In automata theory, the state set is typically finite. In this case, one can view this memory capacity as limited. On the contrary, when the memory of the automaton is not assumed to be limited (human brain), or its capacity can always be extended (RAM-machines or Turing machines as models of computers in computation theory), the automaton should have an infinite state set.

In the theory of dynamical systems, we use a generalisation of automata in which the size of the state space is not restricted to finiteness, or even to countability. Dynamical systems with the behaviour of an automaton, that is, taking inputs in discrete time, are called discrete systems. The state space of such a system acts as a sort of memory of the inputs. Each input influences the current state of the automaton, and the current state is the result of the system's own form—how it deals with inputs—together with the system's history.

One can imagine a dynamical system whose state space is that of all possible input-histories; a new input simply appends to the existing history to become a new history. On the other hand, one can imagine the "opposite" kind of system: one that completely forgets the previous inputs. These systems are referred to as "simple-reflex" in [1, p. 49], reactive, or memoryless in this paper. The transitions of these automata depend only on the input, as no experience is stored. The system decides according to the current perception of the world, rather than current perception together with past perception. In fact, these memoryless systems could act by making a single distinction in the input—a yes/no Boolean response—and nothing more; we call these Boolean reactive systems.

In this paper, we will study the links between systems that have memory and those that do not. More precisely, we prove that systems with memory can be simulated by wiring together

systems without memory. Our result provides a theoretical framework that supports artificial neural network approaches. Memory is carried by connections, and not only by individuals, within a compositional hierarchy of parts.

This article lies between two fields of mathematics: category theory and dynamical systems. In Section 2, we introduce all the background related to category theory and its use in the study of discrete systems. Readers with a background in category theory are invited to skip Section 2.1. We need no more than the basic definitions of categories, functors, natural transformations, monoidal categories, and monoidal functors. Readers already familiar with the study of boxes and discrete systems from a category-theoretic point of view (as in [2]) can skip Section 2.2. Here again, we only need the basic understanding of $\mathscr{C}$-typed-finite sets, $\mathscr{C}$-boxes, wiring diagrams and discrete systems inside a $\mathscr{C}$-box.

In Section 3, we introduce discrete systems and a specific mapping that will serve our purposes (Section 3.1). We then introduce two equivalence relations between discrete systems. Both are bigger than the usual bisimulation used in automata theory (in the sense of set inclusion). One corresponds to an external point of view; two systems are equivalent if they transform input streams into output streams in the same way (Section 3.2). The other relation corresponds to an internal point of view: two systems are equivalent if they have "the same structure" (in a sense that is defined in Section 3.3). We prove that these are just two perspectives on the same relation.

This equivalence relations plays a crucial role in the two results we show in Section 4. First, we show that any discrete system is equivalent to some wiring-together of memoryless systems (Section 4.2). Second, we show that any discrete system with a finite state set is equivalent to a combination of finitely many Boolean reactive systems (Section 4.3).

1.1. Notation

In this article, we will use the following notation.

- Let $\mathbb{N}$ denote the set of all natural numbers, $\mathbb{N} := \{0, 1, 2, \dots\}$.
- By default, the variable n will refer to a natural number: $n \in \mathbb{N}$. We will also see the integers n as their set-theoretic counterparts, that is $0 = \emptyset$, and $n = \{0, 1, \dots, n-1\}$; in that context, $i \in n$ simply means $i \in \{0, \dots, n-1\}$. Note that the set n contains exactly n elements and this is what really matters in this notation.
- When the size of a sequence $(x_0, x_1, \dots, x_{n-1})$ does not matter, it will be denoted $\overline{x}$, which makes it easier to write and read. If each x_i is an element of the same set X, then we will write $\overline{x} \in X$, instead of defining an $n = length(\overline{x})$ and writing $\overline{x} \in X^n$. If $x_i \in X_i$ for possibly different X_i, and if there exists a compact notation $\widehat{X}$ for $X_0 \times X_1 \times \cdots \times X_{n-1}$, then we will write $\overline{x} \in \widehat{X}$ too.
- **Sets** is the usual category of sets; it will be defined in section 2.1.

2. Background

In this chapter, we will present the background necessary for the understanding of this paper, namely that of category theory and dynamical systems. Both will be reduced to the absolute minimum used in this article.

2.1. Background in category theory

This section will present some basic notions about category theory. If the reader is already familiar with the notions of category, functor, natural transformation, product, monoidal category,

and lax monoidal functor, they can skip directly to section 2.2.

Definition 2.1 (Category [3])**.** A *category* $\mathscr{C}$ consists of the following data:

- A collection of *objects*, denoted $\mathrm{Ob}_{\mathscr{C}}$
- A collection of *morphisms*, denoted $\mathrm{Mor}_{\mathscr{C}}$
- A map $\mathrm{dom} : \mathrm{Mor}_{\mathscr{C}} \to \mathrm{Ob}_{\mathscr{C}}$; for each morphism f, $\mathrm{dom}(f)$ is called the *domain* of f
- A map $\mathrm{cod} : \mathrm{Mor}_{\mathscr{C}} \to \mathrm{Ob}_{\mathscr{C}}$; for each morphism f, $\mathrm{cod}(f)$ is the called *codomain* of f
- For each morphism $f \in \mathrm{Mor}_{\mathscr{C}}$, we write $f : A \to B$ if $A = \mathrm{dom}(f)$ and $B = \mathrm{cod}(f)$
- A *composition law* $\circ$ such that, for all $f : A \to B$ and $g : B \to C$, there is a chosen morphism $g \circ f : A \to C$
- For each object $A \in \mathrm{Ob}_{\mathscr{C}}$, there is a chosen morphism $1_A : A \to A$ called *identity morphism of A*

The composition law is required to be associative: $\forall A, B, C, D \in \mathrm{Ob}_{\mathscr{C}}$, $\forall f : A \to B$ and $g : B \to C$ and $h : C \to D$, $(h \circ g) \circ f = h \circ (g \circ f)$. Identity morphisms are required to act like identities: $\forall A, B \in \mathrm{Ob}_{\mathscr{C}}$, $\forall f : A \to B$, $f \circ 1_A = 1_B \circ f = f$.

By abuse of notation, we will often write $C \in \mathscr{C}$ instead of $C \in \mathrm{Ob}_{\mathscr{C}}$, and $(f : C \to D) \in \mathscr{C}$ instead of $(f : C \to D) \in \mathrm{Mor}_{\mathscr{C}}$. It should be clear from the way these are written that we refer respectively to an object and a morphism.

In the rest of the article, a category $\mathscr{C}$ will be described according to the following presentation:

Objects: An object in $\mathscr{C}$ is...

Morphisms: A morphism in $\mathscr{C}$ is...

Identities: An identity morphism is...

Composition: The composition law for morphisms is...

Usually, the description of morphisms suffices to implicitly define dom and cod, as in the following example.

Example 2.2. We define the category **Sets** as the following:

Objects: An object in **Sets** is any set

Morphisms: A morphism in **Sets** is any function $f : A \to B$

Identities: An identity morphism is an identity function $\mathrm{id}_A : A \to A$

Composition: The composition law for morphisms is the usual composition of functions

Similarly, we can define the category **FinSets** whose objects are the finite sets.

We also define mappings somewhat similar to functions, or homomorphisms, between categories.

Definition 2.3 (Functor [3])**.** Let $\mathscr{C}$ and $\mathscr{D}$ be categories.

A *functor* $F : \mathscr{C} \to \mathscr{D}$ is a mapping from $\mathscr{C}$ to $\mathscr{D}$ such that:

- $\forall C \in \mathrm{Ob}_{\mathscr{C}},\ F(C) \in \mathrm{Ob}_{\mathscr{D}}$
- $\forall f : A \to B \in \mathrm{Mor}_{\mathscr{C}},\ F(f) : F(A) \to F(B) \in \mathrm{Mor}_{\mathscr{D}}$
- $\forall A \in \mathrm{Ob}_{\mathscr{C}},\ F(1_A) = 1_{F(A)}$
- $\forall f : A \to B, g : B \to C \in \mathrm{Mor}_{\mathscr{C}},\ F(g \circ f) = F(g) \circ F(f)$

In other words, a functor $F : \mathscr{C} \to \mathscr{D}$ sends the objects (resp. morphisms) in $\mathscr{C}$ to objects (resp. morphisms) in $\mathscr{D}$, preserving domains and codomains of morphisms, as well as identities and composition.

Functors are maps between categories; there also exist maps between functors.

Definition 2.4 (Natural transformation [3])**.** Let $\mathscr{C}$ and $\mathscr{D}$ be two categories, and let $F, G : \mathscr{C} \to \mathscr{D}$ be functors. A *natural transformation* $\theta : F \to G$ consists of a collection of morphisms $(\theta_C : F(C) \to G(C))_{C \in \mathrm{Ob}_{\mathscr{C}}}$ such that, for all $C, D \in \mathscr{C}$, and for all $h : C \to D$, the following square commutes:

$$\begin{array}{ccc} C & & F(C) \xrightarrow{\theta_C} G(C) \\ \downarrow{\scriptstyle h} & \rightsquigarrow & \downarrow{\scriptstyle F(h)} \qquad \downarrow{\scriptstyle G(h)} \\ D & & F(D) \xrightarrow[\theta_D]{} G(D) \end{array}$$

For each object $C \in \mathscr{C}$, the morphism θ_C is called the C-component of θ.

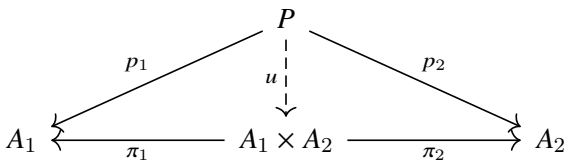
Depending on the context, and for the sake of readability, the C-component of a natural transformation θ can be written θ_C as above (C as an index) or $\theta(C)$ (C as a parameter).

Definition 2.5 (Natural isomorphism [3])**.** A natural transformation $\theta : F \to G$ is called a *natural isomorphism* when all of its components $\theta_C : F(C) \to G(C)$ are isomorphisms.

Basic constructions inside categories include the product of a pair of objects, defined as follows.

Definition 2.6 (Product [3])**.** Let $\mathscr{C}$ be a category and let A_1 and A_2 be objects of $\mathscr{C}$.

A product of A_1 and A_2, written $A_1 \times A_2$, is an object in $\mathscr{C}$, together with two morphisms $\pi_1 : A_1 \times A_2 \to A_1$ and $\pi_2 : A_1 \times A_2 \to A_2$ such that, for all objects P with two morphisms $p_1 : P \to A_1$ and $p_2 : P \to A_2$, there exists a unique morphism $u : P \to A_1 \times A_2$ such that $\pi_1 \circ u = p_1$ and $\pi_2 \circ u = p_2$, that is, such that the following diagram commutes:



We call π_1, π_2 projections, and we denote u by $u = (p_1, p_2)$.

The above can be generalized from $n = 2$ to any $n \in \mathbb{N}$; the result is called an n-ary product. The 1-ary product of A_1 is just A_1 and the projection $\pi_1 : A_1 \to A_1$ is the identity. The 0-ary product is an object $*$ such that for all objects P, there exists a unique morphism $u : P \to *$; it is called a *terminal object*

Note that products need not exist in arbitrary categories, however they do exist in many categories of interest here. For example, in **Sets**, 2-ary products are given by the usual product of sets, and 0-ary products is any set $\{*\}$ with one element.

Definition 2.7 (Category with finite products)**.** A category $\mathscr{C}$ is said to be a *category with finite products* when $\forall n \in \mathbb{N}$, $\forall A_1, \ldots, A_n$ objects of $\mathscr{C}$, the product $A_1 \times \cdots \times A_n$ exists.

An important class of categories is that of monoidal categories.

Definition 2.8 (Monoidal category [4])**.** A *monoidal category* is a 6-tuple $(\mathscr{C}, \otimes, I, a, r, l)$, consisting of a category $\mathscr{C}$, a functor $\otimes : \mathscr{C} \times \mathscr{C} \to \mathscr{C}$, an object $I \in \mathscr{C}$, and three natural isomorphisms a, r, and l of the following form

1. $a(A, B, C) : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$
2. $r(A) : A \otimes I \to A$
3. $l(A) : I \otimes A \to A$

such that, for all objects A, B, C and D of $\mathscr{C}$, the following rules hold:

1. The following diagram commutes:

$$\begin{array}{ccc} A \otimes (I \otimes B) & \xrightarrow{a(A,I,B)} & (A \otimes I) \otimes B \\ & {\scriptstyle A \otimes l(B)} \searrow \quad \swarrow {\scriptstyle r(A) \otimes B} & \\ & A \otimes B & \end{array}$$

2. The following diagram commutes too:

$$\begin{array}{ccc} & A \otimes (B \otimes (C \otimes D)) & \\ {\scriptstyle A \otimes a(B,C,D)} \swarrow & & \searrow {\scriptstyle a(A,B,C \otimes D)} \\ A \otimes ((B \otimes C) \otimes D) & & (A \otimes B) \otimes (C \otimes D) \\ {\scriptstyle a(A,B \otimes C,D)} \downarrow & & \downarrow {\scriptstyle a(A \otimes B,C,D)} \\ (A \otimes (B \otimes C)) \otimes D & \xrightarrow[a(A,B,C) \otimes D]{} & ((A \otimes B) \otimes C) \otimes D \end{array}$$

We also say that $(\otimes, I, a, r, l)$ forms a *monoidal structure* on $\mathscr{C}$.

Roughly, a monoidal category is a category with an operation $\otimes$ which can be seen as associative (items 1 and 2), and a distinguished element I that behaves like a unit for the operation (items 2, 3 and 1).

The category $(\mathbf{Sets}, \times, 1, a, r, l)$ is monoidal for the usual product of sets, where a, r, l are the obvious isomorphisms and 1 is the singleton $1 = \{*\}$.

Remark 2.9. Readers not familiar with category theory may wonder what functors were used to define the natural transformations a, r and l.

The natural transformation a is defined in item 1 by its (A, B, C)-component: $a(A, B, C) : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$. Define the two functors:

$$F : \left\{ \begin{array}{ccc} \mathscr{C} \times \mathscr{C} \times \mathscr{C} & \longrightarrow & \mathscr{C} \\ (A, B, C) & \longmapsto & A \otimes (B \otimes C) \end{array} \right. \quad \text{and} \quad G : \left\{ \begin{array}{ccc} \mathscr{C} \times \mathscr{C} \times \mathscr{C} & \longrightarrow & \mathscr{C} \\ (A, B, C) & \longmapsto & (A \otimes B) \otimes C \end{array} \right.$$

then a is defined as the natural transformation $a : F \to G$. The same holds for r and l. Define:

$$R : \left\{ \begin{array}{ccc} \mathscr{C} & \longrightarrow & \mathscr{C} \\ A & \longmapsto & A \otimes I \end{array} \right. \qquad L : \left\{ \begin{array}{ccc} \mathscr{C} & \longrightarrow & \mathscr{C} \\ A & \longmapsto & I \otimes A \end{array} \right. \qquad \mathrm{id}_{\mathscr{C}} : \left\{ \begin{array}{ccc} \mathscr{C} & \longrightarrow & \mathscr{C} \\ A & \longmapsto & A \end{array} \right.$$

then r is the natural transformation $r : R \to \mathrm{id}_{\mathscr{C}}$ and l is the natural transformation $l : L \to \mathrm{id}_{\mathscr{C}}$.

Definition 2.10 (Symmetric monoidal category [4])**.** A *symmetric monoidal category* is a 7-tuple $(\mathscr{C}, \otimes, I, a, r, l, s)$ such that $(\mathscr{C}, \otimes, I, a, r, l)$ is a monoidal category, and s is a natural isomorphism $s(A, B) : A \otimes B \to B \otimes A$ such that, for all A, B and C of $\mathscr{C}$, the following diagrams commute:

$$\begin{array}{ccc} A \otimes B & \xrightarrow{s(A,B)} & B \otimes A \\ & {\scriptstyle 1_{A\otimes B}} \searrow \quad \swarrow {\scriptstyle s(B,A)} & \\ & A \otimes B & \end{array} \qquad (1) \qquad \begin{array}{ccc} A \otimes I & \xrightarrow{s(A,I)} & I \otimes A \\ & {\scriptstyle r(A)} \searrow \quad \swarrow {\scriptstyle l(A)} & \\ & A & \end{array} \qquad (2)$$

$$\begin{array}{ccc} A \otimes (B \otimes C) & \xrightarrow{a(A,B,C)} & (A \otimes B) \otimes C \\ \downarrow {\scriptstyle A\otimes s(B,C)} & & \downarrow {\scriptstyle s(A\otimes B,C)} \\ A \otimes (C \otimes B) & & C \otimes (A \otimes B) \\ \downarrow {\scriptstyle a(A,C,B)} & & \downarrow {\scriptstyle a(C,A,B)} \\ (A \otimes C) \otimes B & \xrightarrow[s(A,C)\otimes B]{} & (C \otimes A) \otimes B \end{array} \qquad (3)$$

A symmetric monoidal category has an associative and commutative law, with a unit object. Again, **Sets** is a symmetric monoidal category.

Definition 2.11 (Lax monoidal functor [5])**.** Let $(\mathscr{C}, \boxplus, I, a_{\mathscr{C}}, r_{\mathscr{C}}, l_{\mathscr{C}})$ and $(\mathscr{D}, \otimes, J, a_{\mathscr{D}}, r_{\mathscr{D}}, l_{\mathscr{D}})$ be monoidal categories.

A monoidal functor between $\mathscr{C}$ and $\mathscr{D}$ is a 3-tuple (F, σ, σ') where:

- $F : \mathscr{C} \to \mathscr{D}$ is a functor
- σ is a natural transformation $\sigma = \left(\sigma_{A,B} : F(A) \otimes F(B) \to F(A \boxplus B)\right)_{A,B\in\mathscr{C}}$ between two $\mathscr{C} \times \mathscr{C} \to \mathscr{D}$ functors
- σ' is a morphism $\sigma' : J \to F(I)$

such that, for all $A, B, C \in \mathscr{C}$, the following three diagrams commute:

$$\begin{array}{ccc} (F(A) \otimes F(B)) \otimes F(C) & \xrightarrow{a_{\mathscr{D}}(F(A),F(B),F(C))} & F(A) \otimes (F(B) \otimes F(C)) \\ \downarrow {\scriptstyle \sigma_{A,B}\otimes 1_{F(C)}} & & \downarrow {\scriptstyle 1_{F(A)}\otimes\sigma_{B,C}} \\ F(A \boxplus B) \otimes F(C) & & F(A) \otimes F(B \boxplus C) \\ \downarrow {\scriptstyle \sigma_{A\boxplus B,C}} & & \downarrow {\scriptstyle \sigma_{A,B\boxplus C}} \\ F((A \boxplus B) \boxplus C) & \xrightarrow[F(a_{\mathscr{C}}(A,B,C))]{} & F(A \boxplus (B \boxplus C)) \end{array} \qquad (1)$$

$$\begin{array}{ccc} F(A)\otimes F(I) & \xleftarrow{1_{F(A)}\otimes\sigma'} & F(A)\otimes J \\ \downarrow{\scriptstyle \sigma_{A,I}} & & \downarrow{\scriptstyle r_{\mathscr{D}}(F(A))} \\ F(A\boxplus I) & \xrightarrow[F(r_{\mathscr{C}}(A))]{} & F(A) \end{array} \quad (2) \qquad \begin{array}{ccc} F(I)\otimes F(A) & \xleftarrow{\sigma'\otimes 1_{F(A)}} & J\otimes F(A) \\ \downarrow{\scriptstyle \sigma_{I,A}} & & \downarrow{\scriptstyle l_{\mathscr{D}}(F(A))} \\ F(I\boxplus A) & \xrightarrow[F(l_{\mathscr{C}}(A))]{} & F(A) \end{array} \quad (3)$$

The pair (σ, σ') is called the *coherence maps of* F. We sometimes refer to σ as the *first coherence map of* F.

Remark 2.12. Just like we call $\mathbb{R}$ a field without clarifying its two laws and its two units, we often write $(\mathscr{C}, \otimes, I)$, omitting the natural isomorphisms a, r and l, because they are only a matter of "bookkeeping". We may even write $\mathscr{C}$ for a monoidal category when the context makes it clear what the unit and monoidal product are.

2.2. Boxes and wiring diagrams

We will now apply the categorical framework to build discrete systems. Our approach is different from the one in [6]. The dynamical systems presented here are defined as a generalisation of automata whose input and output spaces are predetermined. We will define a category of lists, a category of boxes, and diverse operations on them.

In this section, $\mathscr{C}$ will be any category with finite products (typically **Sets**). Most of the following notions were already defined in [2]; we only recall them without proving their properties. We also give examples in order to help for comprehension.

2.2.1. The category of typed finite sets

Before defining proper boxes, we need to define the notion of input and output ports. These will eventually be the sides of our boxes.

Definition 2.13 (Category of $\mathscr{C}$-typed finite sets [2])**.** The category $\mathbf{TFS}_{\mathscr{C}}$ *of* $\mathscr{C}$*-typed finite sets* is defined as follows:

Objects: An object is any pair (P, τ) such that P is a finite set and $\tau : P \to \mathrm{Ob}_{\mathscr{C}}$ is a function

Morphisms: A morphism from (P, τ) to (P', τ') is a function $\gamma : P \to P'$ such that $\tau = \tau' \circ \gamma$

Identities: The identity morphism on (P, τ) is the identity function of the set P

Composition: The composition of morphisms is the usual composition of functions

An object in $\mathbf{TFS}_{\mathscr{C}}$ is called a $\mathscr{C}$*-typed finite set*; a morphism in $\mathbf{TFS}_{\mathscr{C}}$ is called a $\mathscr{C}$*-typed function*.

We can rewrite a $\mathscr{C}$-typed finite set (P, τ) as the finite sequence $\langle \tau(p_0), \ldots, \tau(p_{n-1})\rangle$, where $P = \{p_0, \ldots, p_{n-1}\}$. A $\mathscr{C}$-typed finite set is simply a list of objects in $\mathscr{C}$, indexed by a finite set P. If $\mathscr{C} = \mathbf{Sets}$, a **Sets**-typed finite set is a list of sets.

A $\mathscr{C}$-typed function $\gamma : (P, \tau) \to (P', \tau')$ can be then seen as a means to obtain the former list $\langle \tau(p_0), \ldots, \tau(p_{n-1})\rangle$ from the latter list $\langle \tau'(p_0), \ldots, \tau'(p_{n-1})\rangle$, by reordering, duplicating or even ignoring its elements. As $\tau = \tau' \circ \gamma$, the list $\langle \tau(p_0), \ldots, \tau(p_{n-1})\rangle$ can be rewritten $\langle \tau'(\gamma(p_0)), \ldots, \tau'(\gamma(p_{n-1}))\rangle$. Beware of the inversion: γ goes from (P, τ) to (P', τ') and we see it as a transformation of the list (P', τ') into the list (P, τ).

Example 2.14. Let A, B and C be objects of $\mathscr{C}$ and consider the following two $\mathscr{C}$-typed finite sets:

- $(2, \tau_2)$ such that $\tau_2(0) = A$ and $\tau_2(1) = B$; thus $(2, \tau_2)$ is the list $\langle A, B\rangle$
- $(3, \tau_3)$ such that $\tau_3(0) = B$, $\tau_3(1) = C$ and $\tau_3(2) = A$; thus $(3, \tau_3)$ is the list $\langle B, C, A\rangle$
- $(4, \tau_4)$ such that $\tau_4(0) = \tau_4(1) = \tau_4(2) = A$ and $\tau_4(3) = B$; thus $(4, \tau_4)$ is the list $\langle A, A, A, B\rangle$

(Remember our set-theoretic notation: $2 = \{0, 1\}$, $3 = \{0, 1, 2\}$ and $4 = \{0, 1, 2, 3\}$.)

The list $\langle A, B\rangle$ can be obtained from the list $\langle B, C, A\rangle$ by taking its third and first elements in this order. A $\mathscr{C}$-typed function from $(2, \tau_2)$ to $(3, \tau_3)$ could be $\gamma : 2 \to 3$ such that $\gamma(0) = 2$ and $\gamma(1) = 0$.

Similarly, the morphisms $\gamma' : 2 \to 4$ that convert the list $\langle A, A, A, B\rangle$ to $\langle A, B\rangle$ are such that $\gamma'(0) = 0$ and $\gamma'(1) = 3$, or $\gamma'(0) = 1$ and $\gamma'(1) = 3$, or $\gamma'(0) = 2$ and $\gamma'(1) = 3$.

We let the reader find the morphisms $(4, \tau_4) \to (3, \tau_3)$ and the morphism $(4, \tau_4) \to (2, \tau_2)$, that is, the morphisms that transform the list $\langle B, C, A\rangle$ into the list $\langle A, A, A, B\rangle$ and the (unique) morphism that transforms the list $\langle A, B\rangle$ into the list $\langle A, A, A, B\rangle$.

What about the morphisms from $(3, \tau_3)$ to $(2, \tau_2)$? The list $\langle A, B\rangle$ does not contain the object C. There is simply no morphism $(3, \tau_3) \to (2, \tau_2)$. The same argument applies to morphisms from $(3, \tau_3)$ to $(4, \tau_4)$.

Definition 2.15 (Sum of typed finite sets [2])**.** Let $(P_0, \tau_0), (P_1, \tau_1) \in \mathbf{TFS}_{\mathscr{C}}$ be two $\mathscr{C}$-typed finite sets.

We define their *sum* by $(P_0, \tau_0) + (P_1, \tau_1) = (P_0 + P_1, \tau_0 + \tau_1)$ as the usual disjoint union of sets $P_0 + P_1$ and $\tau_0 + \tau_1$ as τ_i on P_i for $i \in 2$.

Definition 2.16 (Sum of typed functions [2])**.** Let $\gamma_i : (P_i, \tau_i) \to (P_i', \tau_i')$ $(i \in 2)$ be two $\mathscr{C}$-typed functions.

We define their *sum* as the $\mathscr{C}$-typed function $\gamma_0 + \gamma_1 : (P_0, \tau_0) + (P_1, \tau_1) \to (P_0', \tau_0') + (P_1', \tau_1')$ such that $\forall x \in P_0 + P_1, (\gamma_0 + \gamma_1)(x) = \gamma_i(x)$ if $x \in P_i$ $(i \in 2)$.

We can view the sum $(P, \tau) + (P', \tau')$ as the concatenation of the lists $\langle \tau(p_0), \ldots, \tau(p_n)\rangle$ and $\langle \tau'(p_0'), \ldots, \tau'(p_{n'}')\rangle$, that is, the list $\langle \tau(p_0), \ldots, \tau(p_n), \tau'(p_0'), \ldots, \tau'(p_{n'}')\rangle$, and the sum of $\mathscr{C}$-typed functions as a action on each part of the concatenated list.

Proposition 2.17. *The category* $\mathbf{TFS}_{\mathscr{C}}$ *has the following properties:*

- *The sum of* $\mathscr{C}$*-typed finite sets is a coproduct.*
- *There is only one* $\mathscr{C}$*-typed finite set* (P, τ) *where* $P = \emptyset$*. We denote it by* 0*.*
- $\mathbf{TFS}_{\mathscr{C}}$ *has a symmetric monoidal structure for the sum* $+$*, with* 0 *as the unit.*

Proof. See [2]. □

2.2.2. Dependent products

In this subsection, we define the dependent product functor. If a $\mathscr{C}$-typed finite set can be viewed as a list of objects of $\mathscr{C}$, then the dependent product of this list is simply the product of its elements.

Definition 2.18 (Dependent product [2])**.** We define the *dependent product* as the functor $\widehat{\ } : \mathbf{TFS}_{\mathscr{C}}^{\mathrm{op}} \to \mathscr{C}$ such that:

Action on objects: $\widehat{(P,\tau)} = \prod_{p\in P} \tau(p)$

Action on morphisms: If $\gamma : (P,\tau) \to (P',\tau')$, then $\widehat{\gamma} : \widehat{(P',\tau')} \to \widehat{(P,\tau)}$ is defined as the function $\widehat{\gamma} : \prod_{p'\in P'} \tau'(p') \to \prod_{p\in P} \tau(p)$ such that $\forall \left(a_{p'}\right)_{p'\in P'} \in \widehat{(P',\tau')}$, $\widehat{\gamma}\left(\left(a_{p'}\right)_{p'\in P'}\right) = \left(a_{\gamma(p)}\right)_{p\in P}$.

The interpretation of the dependent product is actually quite straightforward: the dependent product of a $\mathscr{C}$-typed finite set, viewed as a list, is the product of the elements of the list in the same order as they appear in the list.

We remind that $\mathscr{C}$ has finite products; as a consequence, the dependent product always exists.

Example 2.19. Consider the same A, B and C objects of $\mathscr{C}$ and $\mathscr{C}$-typed finite sets as in Example 2.14:

- $(2,\tau_2) = \langle A, B\rangle$
- $(3,\tau_3) = \langle B, C, A\rangle$
- $(4,\tau_4) = \langle A, A, A, B\rangle$

The dependent products of these $\mathscr{C}$-typed finite sets are:

- $\widehat{(2,\tau_2)} = A \times B$
- $\widehat{(3,\tau_3)} = B \times C \times A$
- $\widehat{(4,\tau_4)} = A \times A \times A \times B$

In order to see what the dependent product does to morphisms, take a morphism $\gamma : 2 \to 4$ that converts the list $\langle A, A, A, B\rangle$ to $\langle A, B\rangle$, for example the morphism defined by $\gamma(0) = 0$ and $\gamma(1) = 3$. Its dependent product $\widehat{\gamma}$ will be the function $\widehat{(4,\tau_4)} \to \widehat{(2,\tau_2)} = A \times A \times A \times B \to A \times B$ such that $\widehat{\gamma}(x_0, x_1, x_2, x_3) = \left(x_{\gamma(0)}, x_{\gamma(1)}\right) = (x_0, x_3)$.

We let the reader find the other dependent products as an exercise.

The dependent product is thus a functor that packages the usual operations of diagonal $A \to A \times A$, projection $A \times B \to A$, and swapping $A \times B \to B \times A$.

Proposition 2.20. *There is a natural isomorphism* $\widehat{(P_0,\tau_0) + (P_1,\tau_1)} \cong \widehat{(P_0,\tau_0)} \times \widehat{(P_1,\tau_1)}$*; in other words, the dependent product functor sends coproducts in* $\mathbf{TFS}_{\mathscr{C}}$ *to products in* $\mathscr{C}$*.*

Proof. See [2]. □

This property is also quite intuitive: if one views the coproduct in $\mathbf{TFS}_{\mathscr{C}}$ as the concatenation of lists, and the dependent product as the product of the elements of the list, then the dependent product of the concatenation of two lists is the product of the dependent products of each lists.

2.2.3. The category of boxes and wiring diagrams

The category $\mathbf{TFS}_{\mathscr{C}}$ is not the main purpose of this article; however its properties will be useful for the rest of this article.

In the following, by abuse of notation, we will write $X \in \mathbf{TFS}_{\mathscr{C}}$ for (X,τ), and $\widehat{X}$ for $\widehat{(X,\tau)}$.

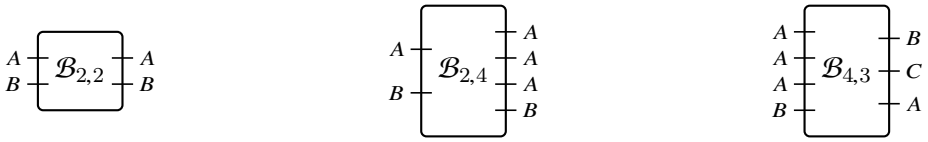
Definition 2.21 ($\mathscr{C}$-box [2])**.** We call *$\mathscr{C}$-box* any pair $X = \left(X^{\text{in}}, X^{\text{out}}\right) \in \mathbf{TFS}_{\mathscr{C}} \times \mathbf{TFS}_{\mathscr{C}}$.

A $\mathscr{C}$-box is a pair of $\mathscr{C}$-typed finite sets $(X^{\text{in}}, X^{\text{out}})$, where X^{in} represent the list of inputs ports, and X^{out} represent the list of outputs ports.

Example 2.22. From the typed finite sets in Example 2.14, we can build the following $\mathscr{C}$-boxes:

- $\mathcal{B}_{2,2} = ((2, \tau_2), (2, \tau_2)) = (\langle A, B\rangle, \langle A, B\rangle)$
- $\mathcal{B}_{2,4} = ((2, \tau_2), (4, \tau_4)) = (\langle A, B\rangle, \langle A, A, A, B\rangle)$
- $\mathcal{B}_{4,3} = ((4, \tau_4), (3, \tau_3)) = (\langle A, A, A, B\rangle, \langle B, C, A\rangle)$

These $\mathscr{C}$-boxes are represented here:



In the rest of the paper, the ports will no longer be labelled, for the sake of readability.

Definition 2.23 (Wiring diagram [2])**.** Let $X = (X^{\text{in}}, X^{\text{out}})$ and $Y = (Y^{\text{in}}, Y^{\text{out}})$ be $\mathscr{C}$-boxes.

A *wiring diagam* $\varphi : X \to Y$ is a pair of $\mathscr{C}$-typed functions $(\varphi^{\text{in}}, \varphi^{\text{out}})$ such that:

- $\varphi^{\text{in}} : X^{\text{in}} \to Y^{\text{in}} + X^{\text{out}}$
- $\varphi^{\text{out}} : Y^{\text{out}} \to X^{\text{out}}$

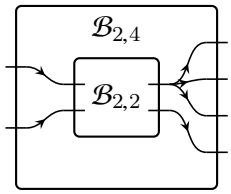
The $\mathscr{C}$-typed function φ^{in} tells what feeds the input ports of the box X: each input port of X is either connected to an input port of Y or to an output port of X (in case of feedback); the $\mathscr{C}$-typed function φ^{out} tells what feeds the output ports of Y: each output port of Y is connected to some output port of X.

Example 2.24. Given $\mathcal{B}_{2,2}$ and $\mathcal{B}_{2,4}$ defined in Example 2.22, the wiring diagrams $\varphi : \mathcal{B}_{2,2} \to \mathcal{B}_{2,4}$ will have the following form:

- $\varphi^{\text{in}} : (2, \tau_2) \to (2, \tau_2) + (2, \tau_2) = \langle A, B\rangle \to \langle A, B, A, B\rangle$
- $\varphi^{\text{out}} : (4, \tau_4) \to (2, \tau_2) = \langle A, A, A, B\rangle \to \langle A, B\rangle$

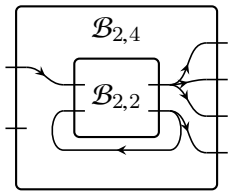
We can build specific wiring diagrams such as:

$\varphi_0^{\text{in}}(0) = 0$
$\varphi_0^{\text{in}}(1) = 1$



$\varphi_0^{\text{out}}(0) = 0$
$\varphi_0^{\text{out}}(1) = 0$
$\varphi_0^{\text{out}}(2) = 0$
$\varphi_0^{\text{out}}(3) = 1$

$\varphi_1^{\text{in}}(0) = 0$
$\varphi_1^{\text{in}}(1) = 3$



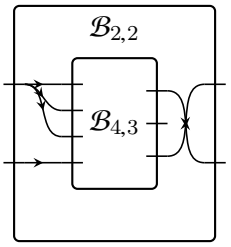
$\varphi_1^{\text{out}}(0) = 0$
$\varphi_1^{\text{out}}(1) = 0$
$\varphi_1^{\text{out}}(2) = 0$
$\varphi_1^{\text{out}}(3) = 1$

Let us consider a wiring diagram $\psi : \mathcal{B}_{4,3} \to \mathcal{B}_{2,2}$ (defined in Example 2.22). It will look like:

- $\psi^{\text{in}} : (4, \tau_4) \to (2, \tau_2) + (3, \tau_3) = \langle A, A, A, B \rangle \to \langle A, B, B, C, A \rangle$
- $\psi^{\text{out}} : (2, \tau_2) \to (3, \tau_3) = \langle A, B \rangle \to \langle B, C, A \rangle$

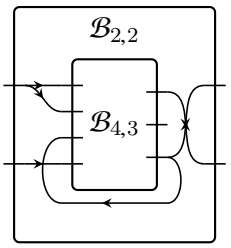
We can build specific wiring diagrams:

$\psi_0^{\text{in}}(0) = 0$
$\psi_0^{\text{in}}(1) = 0$
$\psi_0^{\text{in}}(2) = 0$
$\psi_0^{\text{in}}(3) = 1$



$\psi_0^{\text{out}}(0) = 2$
$\psi_0^{\text{out}}(1) = 0$

$\psi_1^{\text{in}}(0) = 0$
$\psi_1^{\text{in}}(1) = 0$
$\psi_1^{\text{in}}(2) = 4$
$\psi_1^{\text{in}}(3) = 1$



$\psi_1^{\text{out}}(0) = 2$
$\psi_1^{\text{out}}(1) = 0$

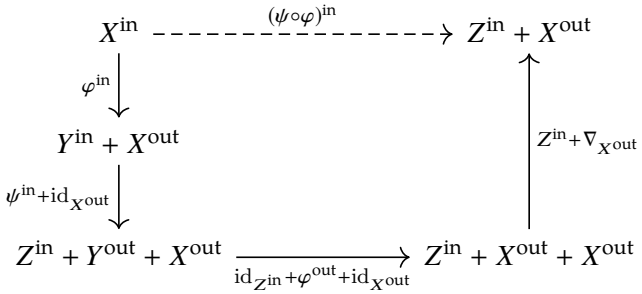
What about the reverse wiring diagram $\rho : \mathcal{B}_{2,2} \to \mathcal{B}_{4,3}$? It will look like:

- $\rho^{\text{in}} : (2, \tau_2) \to (4, \tau_4) + (2, \tau_2) = \langle A, B \rangle \to \langle A, A, A, B, A, B \rangle$
- $\rho^{\text{out}} : (3, \tau_3) \to (2, \tau_2) = \langle B, C, A \rangle \to \langle A, B \rangle$

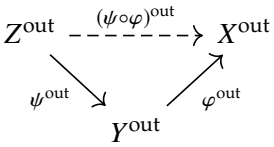
There is no problem with the first $\mathscr{C}$-typed function, but we already know that there is no $\mathscr{C}$-typed function $(3, \tau_3) \to (2, \tau_2)$ (cf. Example 2.14).

We can now compose the wiring diagrams:

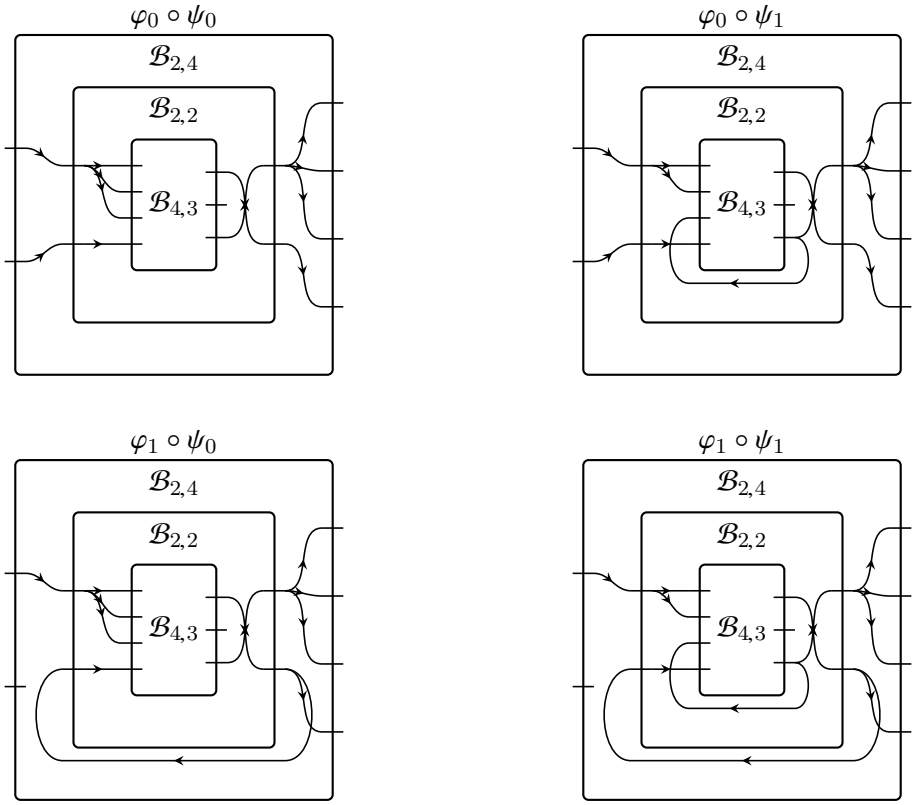
Definition 2.25 (Composition of wiring diagrams [2])**.** Let $\varphi : X \to Y$ and $\psi : Y \to Z$ be two wiring diagrams. We define their *composition*, denoted $\psi \circ \phi$, as the pair $\left((\psi \circ \phi)^{\text{in}}, (\psi \circ \phi)^{\text{out}}\right)$, where $(\psi \circ \phi)^{\text{in}}$ is defined such that the following diagram commutes:



and $(\psi \circ \phi)^{\text{out}}$ is defined such that the following diagram commutes:



Example 2.26. We can compose $\psi_i : \mathcal{B}_{4,3} \to \mathcal{B}_{2,2}$ with $\varphi_j : \mathcal{B}_{2,2} \to \mathcal{B}_{2,4}$ $(i, j \in 2)$ (defined in Example 2.24).



Definition 2.27 (Category of $\mathscr{C}$-boxes and wiring diagrams [2])**.** The *category $\mathscr{W}_{\mathscr{C}}$ of $\mathscr{C}$-boxes and wiring diagrams* is defined as follows:

Objects: An object in $\mathscr{W}_{\mathscr{C}}$ is a $\mathscr{C}$-box

Morphisms: A morphism between two $\mathscr{C}$-boxes X and Y is a wiring diagram $\phi : X \to Y$

Identities: An identity morphism on X is the identity wiring diagram

Composition: The composition of wiring diagrams is the composition defined in definition 2.25

2.2.4. Monoidal structure of the category of boxes

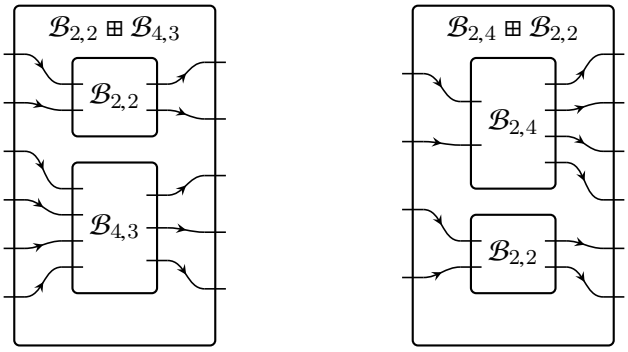
The category $\mathscr{W}_{\mathscr{C}}$ has a monoidal structure for the parallel composition of boxes, that corresponds to the intuitive idea of parallelising boxes.

Definition 2.28 (Parallel composition of boxes [2])**.** Let $X = \left(X^{\text{in}}, X^{\text{out}}\right)$ and $Y = \left(Y^{\text{in}}, Y^{\text{out}}\right)$ be two $\mathscr{C}$-boxes.

The *parallel composition*, or *sum*, of X and Y, denoted $X \boxplus Y$, is the box $X \boxplus Y = \left(X^{\text{in}} + Y^{\text{in}}, X^{\text{out}} + Y^{\text{out}}\right)$, where $+$ is the sum of $\mathscr{C}$-typed finite sets (cf. Definition 2.15).

The parallel composition of two $\mathscr{C}$-boxes summarises to the concatenation of both input ports, and both output ports.

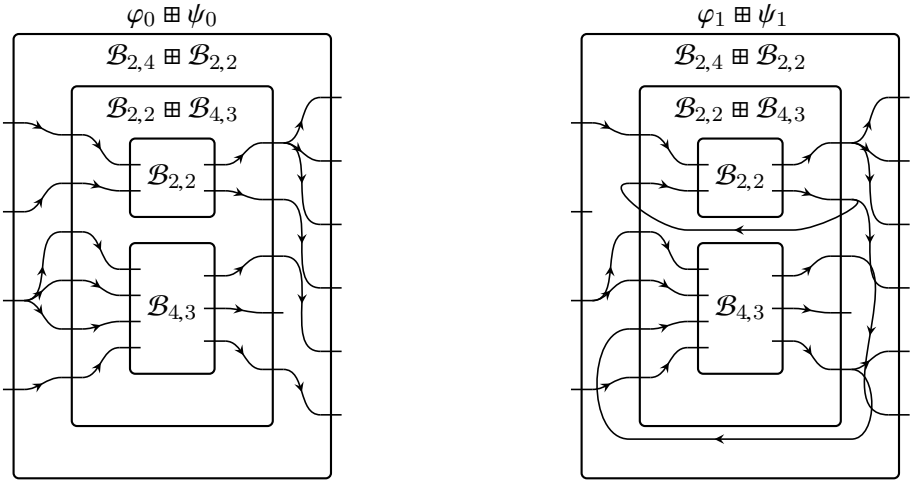
Example 2.29. Any two $\mathscr{C}$-boxes can be put in parallel. For example:



Definition 2.30 (Parallel composition of wiring diagrams [2])**.** Let $\varphi : X \to Y = (\varphi^{\text{in}}, \varphi^{\text{out}})$ and $\psi : Y \to Z = (\psi^{\text{in}}, \psi^{\text{out}})$ be two wiring diagrams.

The *parallel composition*, or *sum*, of φ and ψ, denoted $\varphi \boxplus \psi$, is the wiring diagram $\varphi \boxplus \psi = (\varphi^{\text{in}} + \psi^{\text{in}}, \varphi^{\text{out}} + \psi^{\text{out}})$, where $+$ is the sum of $\mathscr{C}$-typed functions (cf. Definition 2.16).

Example 2.31. Using the notations of Example 2.24, we can build $\varphi_i \boxplus \psi_i : \mathcal{B}_{2,2} \boxplus \mathcal{B}_{4,3} \to \mathcal{B}_{2,4} \boxplus \mathcal{B}_{2,2}$ $(i \in 2)$:



Proposition 2.32. *The category $\mathscr{W}_{\mathscr{C}}$ has the following properties:*

- *The* closed box $\square$*, defined as* $\square = (0, 0)$*, where* 0 *is* $\mathscr{C}$*-typed finite set* $(\emptyset, \emptyset \to \mathscr{C})$ *defined in* 2.17*, is the unit for the sum of boxes* $\boxplus$*.*
- $\mathscr{W}_{\mathscr{C}}$ *has a symmetric monoidal structure for the sum of boxes* $\boxplus$*, with* $\square$ *as the unit.*

Proof. See [2]. □

2.2.5. Dependent product of boxes

The aim of this section is to extend the notion of dependent product (Definition 2.18) to $\mathscr{C}$-boxes and wiring diagrams.

Definition 2.33 (Dependent product of a $\mathscr{C}$-box [2])**.** The *dependent product* $\widehat{X}$ of the $\mathscr{C}$-box $X = (X^{\text{in}}, X^{\text{out}})$ is the pair $\widehat{X} = \left(\widehat{X^{\text{in}}}, \widehat{X^{\text{out}}}\right)$.

Remark 2.34. The dependent product of $X_0 \boxplus X_1$ is $\widehat{X_0 \boxplus X_1} = \left(\widehat{X_0^{\text{in}}} \times \widehat{X_1^{\text{in}}}, \widehat{X_0^{\text{out}}} \times \widehat{X_1^{\text{out}}}\right)$.

Definition 2.35 (Dependent product of wiring diagrams [2])**.** The *dependent product* $\widehat{X}$ of the wiring diagram $\varphi : X \to Y$ is the pair $\widehat{\varphi} = \left(\widehat{\varphi^{\text{in}}}, \widehat{\varphi^{\text{out}}}\right)$.

Remark 2.36. The dependent product $\varphi_0 \boxplus \varphi_1$ is $\widehat{\varphi_0 \boxplus \varphi_1} = \left(\widehat{\varphi_0^{\text{in}}} \times \widehat{\varphi_1^{\text{in}}}, \widehat{\varphi_0^{\text{out}}} \times \widehat{\varphi_1^{\text{out}}}\right)$.

Proposition 2.37. *Let $\varphi : X \to Y$ and $\psi : Y \to Z$. The dependent product of $\psi \circ \varphi$ is the pair $\widehat{\psi \circ \varphi} = \left(\widehat{(\psi \circ \varphi)^{in}}, \widehat{(\psi \circ \varphi)^{\text{out}}}\right)$ where:*

- $\widehat{(\psi \circ \varphi)^{in}}(x, z) = \widehat{\varphi^{in}}\left(\widehat{\psi^{in}}\left(z, \widehat{\varphi^{\text{out}}}(x)\right), x\right)$
- $\widehat{(\psi \circ \varphi)^{\text{out}}}(x) = \widehat{\psi^{\text{out}}}\left(\widehat{\varphi^{\text{out}}}(x)\right)$

Proof. See [2]. □

Remark 2.38. The dependent product of $\mathscr{C}$-boxes and wiring diagrams could be described in terms of monoidal functors; however the codomain of this functor is not $\mathscr{C} \times \mathscr{C}$ as expected, but a category that has the same objects (pairs of objects (A, B) of $\mathscr{C}$) but whose morphisms are pairs of morphisms $\left(\widehat{f^{\text{in}}}, \widehat{f^{\text{out}}}\right) : (A_0, B_0) \to (A_1, B_1)$ such that $\widehat{f^{\text{in}}}$ is the morphism $\widehat{f^{\text{in}}} : A_1 \times B_0 \to A_0$ in $\mathscr{C}$ and $\widehat{f^{\text{out}}}$ is the morphism $\widehat{f^{\text{out}}} : B_0 \to B_1$ in $\mathscr{C}$. The composition law is the one given in Proposition 2.37.

Until now, we have only defined a category of $\mathscr{C}$-boxes, with interesting properties. These $\mathscr{C}$-boxes are exactly as their name suggests: empty boxes. The extension of the dependent product to $\mathscr{C}$-boxes is a necessary step in order to define the "inhabitants" of $\mathscr{C}$-boxes.

3. Discrete systems and their equivalences

In this chapter and in the rest of this paper, we will consider the special case where $\mathscr{C} = \mathbf{Sets}$. Thus, in general, we will simply call "boxes" what we introduced as "**Sets**-boxes". We denote the symmetric monoidal category of boxes as $\mathscr{W}_{\mathbf{Sets}}$.

3.1. Definition and basic properties

The notions introduced in this section come from [2]. The properties stated here are proven in the same article.

Definition 3.1 (Discrete systems [2])**.** Let $X = \left(X^{\text{in}}, X^{\text{out}}\right) \in \mathscr{W}_{\mathbf{Sets}}$ be a box.

A *discrete system for the box X*, or *discrete system* for short, is a 4-tuple $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right)$ where:

- $S_F \in \mathbf{Sets}$ is the *state set* of F
- $f^{\text{rdt}} : S_F \to \widehat{X^{\text{out}}}$ is its *readout function*
- $f^{\text{upd}} : \widehat{X^{\text{in}}} \times S_F \to S_F$ is its *update function*
- $s_0 \in S_F$ is its *initial state*

We denote by $\mathrm{DS}(X)$ the set of all discrete systems for the box X.

Remark 3.2. In Proposition 2.32, we defined the closed box $\square = (0, 0)$, where 0 denotes $(\emptyset, \tau : \emptyset \to \mathbf{Sets})$. Its dependent product is $\widehat{\square} = \left(\prod_{p\in\emptyset} \tau(p), \prod_{p\in\emptyset} \tau(p)\right) \cong (1', 1')$, where $1'$ is any typed finite set of the form $(1, \tau : 1 \to \mathbf{Sets})$. As a consequence, we have:

$$\mathrm{DS}(\square) \cong \left\{\left(S_F, f^{\mathrm{rdt}}, f^{\mathrm{upd}}, s_{F,0}\right) \mid S \in \mathbf{Sets},\ f^{\mathrm{rdt}} : S \to 1,\ f^{\mathrm{upd}} : 1 \times S_F \to S_F,\ s_{F,0} \in S_F\right\}$$

In other words, an inhabitant of a closed box is a dynamical system with no inputs and no outputs, just a set S and a function $S \to S$.

Remark 3.3. From a set-theoretic point of view, $\mathrm{DS}(X)$ is too big to be a set. A potential solution is to define the $\mathrm{DS}(X)$ within a set big enough for our purposes; for example, the set V_{ω_1} from the von Neumann hierarchy of sets, which contains the usual sets, vector spaces, measurable spaces, Hausdorff spaces, fields, etc. used in mathematics ($V_{\omega\times 2}$ suffices [7, Lemma 2.9]).

In the following, we will continue to write $\mathrm{DS}(X)$ (and similarly for mappings) with the state set in $S_F \in \mathbf{Sets}$ for the sake of understandability, but in case set-theoretic problems emerge, we should not write $S_F \in \mathbf{Sets}$ but $S_F \in V_{\omega_1}$.

Note that discrete systems can be viewed as a generalisation of automata. They have no final states, the transition function is always a function, i.e. all discrete systems are deterministic, the input alphabet can be infinite, and the transition function is always defined on every input and every state. Discrete systems are not automata that recognize a language, but rather, automata that take any input stream and return an output stream based on the states it transitioned to; that is, discrete systems are a generalisation of transducers as defined in [8]. Alternatively, discrete systems exactly correspond to the *sequential automata* in [9].

Example 3.4. For this example, we generalise the notation seen in Definition 2.13 to the set $\mathbf{TFS}_{\mathbb{R}}$ of $\mathbb{R}$-typed finite sets, seen as lists of real numbers, that is, finite sequences of real numbers:

$$\mathbf{TFS}_{\mathbb{R}} = \{(n, \tau) \mid n \in \mathbb{N}, \tau : n \to \mathbb{R}\}$$

Here, this is a set, not a category; besides, we use $n \in \mathbb{N}$ instead of $P \in \mathbf{FinSets}$ so that we define a set. As a discrete category (having only identity morphisms), it is equivalent to $\mathbf{TFS}_{\mathscr{C}\mathbb{R}}$ obtained by also considering $\mathbb{R}$ as a discrete category.

We also generalise the sum of finite sequences, seen as the concatenation.

Consider the box $X_0 = (\langle\mathbb{R}\rangle, \langle\mathbb{R}\rangle) \in \mathscr{W}_{\mathbf{Sets}}$. We define the following discrete system $F_0 \in \mathrm{DS}(X_0)$:

- $S_{F_0} = \mathbf{TFS}_{\mathbb{R}}$
- $f_0^{\mathrm{upd}} : \left\{\begin{array}{ccc} \mathbb{R} \times \mathbf{TFS}_{\mathbb{R}} & \to & \mathbf{TFS}_{\mathbb{R}} \\ (a, \langle a_0, \ldots, a_{n-1}\rangle) & \mapsto & \langle a_0, \ldots, a_{n-1}, a\rangle \end{array}\right.$
- $f_0^{\mathrm{rdt}} : \left\{\begin{array}{ccc} \mathbf{TFS}_{\mathbb{R}} & \to & \mathbb{R} \\ \langle a_0, \ldots, a_{n-1}\rangle & \mapsto & \max(\langle a_0, \ldots, a_{n-1}\rangle) \end{array}\right.$
- $s_{F_0,0} = 0$ (where 0 is the empty list as defined in Proposition 2.17)

In this example, the state set $S_{F_0} = \mathbf{TFS}_{\mathbb{R}}$ is uncountably infinite and clearly works as a memory. This discrete system F_0 takes a real number as an input, appends it to its memory, and computes the maximum value of the stored list.

A more complicated discrete system could return several results; for example, in the box $X_1 = (\langle\mathbb{R}\rangle, \langle\mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R}\rangle) \in \mathscr{W}_{\mathbf{Sets}}$, we can define $F_1 \in \mathrm{DS}(X_1)$ such that:

- $S_{F_1} = S_{F_0} = \mathbf{TFS}_{\mathbb{R}}$, $f_1^{\text{upd}} = f_0^{\text{upd}}$ and $s_{F_1,0} = 0$ (same state set, update function and initial state as F_0)
- $f_1^{\text{rdt}} : \left\{ \begin{array}{ccc} \mathbf{TFS}_{\mathbb{R}} & \to & \mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\mathbb{R} \\ \langle a_0,\dots,a_{n-1}\rangle & \mapsto & (\max(\langle a_0,\dots,a_{n-1}\rangle), \min(\langle a_0,\dots,a_{n-1}\rangle), \\ & & \text{mean}(\langle a_0,\dots,a_{n-1}\rangle), \text{var}(\langle a_0,\dots,a_{n-1}\rangle)) \end{array} \right.$

Or we could add a switch so that we can decide what output we want; in the box $X_2 = (\langle \mathbb{R}, 4\rangle, \langle \mathbb{R}\rangle)$, we define F_2 as:

- $S_{F_2} = \mathbf{TFS}_{\mathbb{R}} \times 4$ (here we see 4 as its set-theoretic counterpart: $4 = \{0,1,2,3\}$)
- $f_2^{\text{upd}} : \left\{ \begin{array}{ccc} \mathbb{R}\times 4\times \mathbf{TFS}_{\mathbb{R}}\times 4 & \to & \mathbf{TFS}_{\mathbb{R}}\times 4 \\ (a, b, \langle a_0,\dots,a_{n-1}\rangle, c) & \mapsto & (\langle a_0,\dots,a_{n-1}, a\rangle, b) \end{array} \right.$
- $f_2^{\text{rdt}} : \left\{ \begin{array}{ccc} \mathbf{TFS}_{\mathbb{R}}\times 4 & \to & \mathbb{R} \\ (\langle a_0,\dots,a_{n-1}\rangle, b) & \mapsto & \left\{ \begin{array}{ll} \max(\langle a_0,\dots,a_{n-1}\rangle) & \text{if } b = 0 \\ \min(\langle a_0,\dots,a_{n-1}\rangle) & \text{if } b = 1 \\ \text{mean}(\langle a_0,\dots,a_{n-1}\rangle) & \text{if } b = 2 \\ \text{var}(\langle a_0,\dots,a_{n-1}\rangle) & \text{otherwise} \end{array} \right. \end{array} \right.$
- $s_{F_2,0} = (0,0)$

We could even add a reset button on the discrete system; in the box $X_3 = (\langle \mathbb{R}, 4, 2\rangle, \langle \mathbb{R}\rangle)$, we define F_3 as:

- $S_{F_3} = S_{F_2} = \mathbf{TFS}_{\mathbb{R}} \times 4$, $f_3^{\text{rdt}} = f_2^{\text{rdt}}$ and $s_{F_3,0} = s_{F_2,0} = (0,0)$ (same state set, same readout function and same initial state as F_2)
- $f_3^{\text{upd}} : \left\{ \begin{array}{ccc} \mathbb{R}\times 4\times 2\times \mathbf{TFS}_{\mathbb{R}}\times 4 & \to & \mathbf{TFS}_{\mathbb{R}}\times 4 \\ (a, b, r, \langle a_0,\dots,a_{n-1}\rangle, c) & \mapsto & \left\{ \begin{array}{ll} s_{F_3,0} & \text{if } r = 0 \\ (\langle a_0,\dots,a_{n-1}, a\rangle, b) & \text{otherwise} \end{array} \right. \end{array} \right.$

We previously viewed general boxes (objects in $\mathscr{W}_{\mathbf{Sets}}$) as empty frames. Discrete systems are the objects that "live" inside. One can draw a parallel with programming: a $\mathscr{C}$-box is the signature of the function, that is, its accepted types of inputs and outputs, and the discrete system is the actual code of the function.

In the rest of the article, we will often represent a discrete system $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right)$ as the following two-arrow graph:

$$F : \qquad \widehat{X^{\text{in}}} \times S_F \xrightarrow{\quad f^{\text{upd}} \quad} S_F \xrightarrow{\quad f^{\text{rdt}} \quad} \widehat{X^{\text{out}}}$$

The first function describes how a state and an input are transformed into a new state; the second describes how the state is output, or "read out". In general, the initial state $s_{F,0} \in S_F$ will not be represented in these diagrams, though it is implicitly there.

Discrete systems are part of the more general class of *dynamical systems*. We can define other types of dynamical systems depending on the category $\mathscr{C}$ that we are interested in. If $\mathscr{C}$ is the category **Euc** of Euclidean spaces, then we will refer to *continuous systems*. For more examples, see [2].

Definition 3.5 (DS-application of a wiring diagram [2])**.** Let $\varphi : X \to Y$ be a wiring diagram. Let $F = \left(S_F, f^{\mathrm{rdt}}, f^{\mathrm{upd}}, s_{F,0}\right) \in \mathrm{DS}(X)$.

The *DS-application of φ to F*, denoted $\mathrm{DS}(\varphi)(F)$, is the discrete system $\mathrm{DS}(\varphi)(F) = \left(S_G, g^{\mathrm{rdt}}, g^{\mathrm{upd}}, s_{G,0}\right) \in \mathrm{DS}(Y)$ such that:

- $S_G = S_F$
- $g^{\mathrm{rdt}} : s \mapsto \widehat{\varphi^{\mathrm{out}}}\left(f^{\mathrm{rdt}}(s)\right)$
- $g^{\mathrm{upd}} : (y, s) \mapsto f^{\mathrm{upd}}\left(\widehat{\varphi^{\mathrm{in}}}\left(y, f^{\mathrm{rdt}}(s)\right), s\right)$
- $s_{G,0} = s_{F,0}$

We can view $\mathrm{DS}(\varphi)(F)$ as the discrete system we obtain from F by implementing the wiring diagram φ.

Definition 3.6 (Parallel composition of discrete systems [2])**.** Let X_0, X_1 be boxes and let $F_i = \left(S_{F_i}, f_i^{\mathrm{rdt}}, f_i^{\mathrm{upd}}, s_{F_i,0}\right) \in \mathrm{DS}(X_i)$ $(i \in 2)$ be discrete systems.

The *parallel composition of F_0 and F_1*, denoted $F_0 \boxtimes F_1$, is the discrete system $\left(S_G, g^{\mathrm{rdt}}, g^{\mathrm{upd}}, s_{G,0}\right) \in \mathrm{DS}(X_0 \boxplus X_1)$ such that:

- $S_G = S_{F_0} \times S_{F_1}$
- $s_{G,0} = \left(s_{F_0,0}, s_{F_1,0}\right)$
- $g^{\mathrm{rdt}} = f_0^{\mathrm{rdt}} \times f_1^{\mathrm{rdt}} : S_{F_0} \times S_{F_1} \to \widehat{X_0^{\mathrm{out}}} \times \widehat{X_1^{\mathrm{out}}}$
- $g^{\mathrm{upd}} : \widehat{X_0^{\mathrm{in}}} \times \widehat{X_1^{\mathrm{in}}} \times S_{F_0} \times S_{F_1} \to S_{F_0} \times S_{F_1}$ makes the following diagram commute:

$$\begin{array}{ccc} \widehat{X_0^{\mathrm{in}}} \times \widehat{X_1^{\mathrm{in}}} \times S_{F_0} \times S_{F_1} & \xrightarrow{g^{\mathrm{upd}}} & S_{F_0} \times S_{F_1} \\ \updownarrow \cong & & \updownarrow = \\ \widehat{X_0^{\mathrm{in}}} \times S_{F_0} \times \widehat{X_1^{\mathrm{in}}} \times S_{F_1} & \xrightarrow{f_0^{\mathrm{upd}} \times f_1^{\mathrm{upd}}} & S_{F_0} \times S_{F_1} \end{array}$$

We also define the parallel composition of $\mathrm{DS}(X_0)$ and $\mathrm{DS}(X_1)$, denoted $\mathrm{DS}(X_0) \boxtimes \mathrm{DS}(X_1)$, by:

$$\mathrm{DS}(X_0) \boxtimes \mathrm{DS}(X_1) = \{F_0 \boxtimes F_1 \mid F_0 \in \mathrm{DS}(X_0), F_1 \in \mathrm{DS}(X_1)\}.$$

Proposition 3.7. *Parallel composition $(F_0, F_1) \mapsto F_0 \boxtimes F_1$ provides a natural map $DS(X_0) \times DS(X_1) \to DS(X_0 \boxplus X_1)$.*

Proof. See [2]. □

Theorem 3.8. *Definitions 3.1, 3.5, and 3.6 define a lax monoidal functor $DS : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$.*

Proof. See [2] □

3.2. An external equivalence relation on dynamical systems

Via the monoidal functor DS, a box contains a specified sort of discrete system (depending on the ports of the box). For an exterior spectator, the content of the box does not matter; what matters is the way it transforms input streams to output streams. Thus, even if two boxes contain different discrete systems, for example one with an infinite state set, and the other with a finite state set, as long as they both give the same output in response to the same input, then they are viewed as "equivalent" from an external point of view.

The following definitions formalise this idea.

Definition 3.9 (Input and output streams)**.** Let $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}(X)$.

An *input stream (for X)* is a finite sequence $\overline{x^{\text{in}}} = \left(x_i^{\text{in}}\right)_{i\in n} \in \left(\widehat{X^{\text{in}}}\right)^n$, where $n \in \mathbb{N}$.

The *output stream produced by F when given* $\overline{x^{in}}$, denoted $F\left(\overline{x^{\text{in}}}\right)$, is the stream $\overline{x^{\text{out}}}$ defined by the following recursive system:

$$\begin{cases} s_0 &= s_{F,0} \\ s_{i+1} &= f^{\text{upd}}\left(x_i^{\text{in}}, s_i\right) \\ x_i^{\text{out}} &= f^{\text{rdt}}\left(s_{i+1}\right) \end{cases}$$

We refer to the state s that F reaches after having processed the input stream $\overline{x^{\text{in}}}$ as *resulting state of F*, and denote it $F_{\text{res}}(\overline{x^{\text{in}}})$. Formally, if $\overline{x^{\text{in}}} = \left(x_i^{\text{in}}\right)_{i\in n}$, then according to the previous recursive system, the resulting state of F is $F_{\text{res}}(\overline{x^{\text{in}}}) = s_n$.

Remark 3.10. According to the notation proposed in section 1.1, $\overline{x^{\text{in}}} = \left(x_i^{\text{in}}\right)_{i\in n} \in \left(\widehat{X^{\text{in}}}\right)^n$ will be written $\overline{x^{\text{in}}} \in \widehat{X^{\text{in}}}$.

Remark 3.11. Definition 3.9 is a continuation of the definitions of *run maps* and *behaviours* in [9], which are functions that assign respectively the resulting state and the last output of the automaton given an input stream. The results we obtain with our notations are similar to those in [9].

Definition 3.12 (Equivalence as stream transducers)**.** Let $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right)$ and $G = \left(S_G, g^{\text{rdt}}, g^{\text{upd}}, s_{G,0}\right)$ be two discrete systems.

We say that *F and G are equivalent as stream transducers*, and we write $F \equiv G$, when, $\forall \overline{x^{\text{in}}} \in \widehat{X^{\text{in}}},\ F\left(\overline{x^{\text{in}}}\right) = G\left(\overline{x^{\text{in}}}\right)$.

It is easy to see that:

Proposition 3.13. *The relation* $\equiv$ *is an equivalence relation on the set* $DS(X)$*, for any box* X*.*

3.3. An internal equivalence relation on dynamical systems

The relation $\equiv$ defined above does not give any information on the links between two discrete systems that are equivalent as stream transducers. In this subsection, we define another equivalence relation that provides an internal point of view. We then prove that the two equivalence relations are the same.

In the following, $X = \left(X^{\text{in}}, X^{\text{out}}\right)$ is any box.

Definition 3.14 (Simulation relation)**.** Suppose given $F = \left(S_F, f^{\mathrm{rdt}}, f^{\mathrm{upd}}, s_{F,0}\right)$ and $G = \left(S_G, g^{\mathrm{rdt}}, g^{\mathrm{upd}}, s_{G,0}\right)$ in $\mathrm{DS}(X)$.

We say that F *simulates* G, and we write $F \vdash G$, if there exists $\alpha : S_F \to S_G$ such that $s_{G,0} = \alpha\left(s_{F,0}\right)$ and such that $\alpha \circ f^{\mathrm{upd}} = g^{\mathrm{upd}} \circ \left(\mathrm{id}_{\widehat{X^{\mathrm{in}}}}, \alpha\right)$, and $f^{\mathrm{rdt}} = g^{\mathrm{rdt}} \circ \alpha$; that is, preserving the initial state and making the following two diagrams commute:

$$
\begin{array}{cccccc}
F: & \widehat{X^{\mathrm{in}}} \times S_F & \xrightarrow{f^{\mathrm{upd}}} & S_F & \xrightarrow{f^{\mathrm{rdt}}} & \widehat{X^{\mathrm{out}}} \\
 & \downarrow{\scriptstyle \left(\mathrm{id}_{\widehat{X^{\mathrm{in}}}}, \alpha\right)} & & \downarrow{\scriptstyle \alpha} & & \updownarrow{\scriptstyle =} \\
G: & \widehat{X^{\mathrm{in}}} \times S_G & \xrightarrow[g^{\mathrm{upd}}]{} & S_G & \xrightarrow[g^{\mathrm{rdt}}]{} & \widehat{X^{\mathrm{out}}}
\end{array}
\tag{4}
$$

We refer to α as a *simulation function*: it witnesses the simulation $F \vdash G$.

A priori, the simulation relation does not relate the output of the two discrete systems F and G (though this does follow; see Lemma 3.19); it only declares a correspondence between both their state sets and update and readout functions. Both discrete systems can work in parallel; their state sets need not be the same, nor even of the same cardinality, but they somehow *coordinate* via the map α. The function α draws the parallel between the internal machinery of F and that of G.

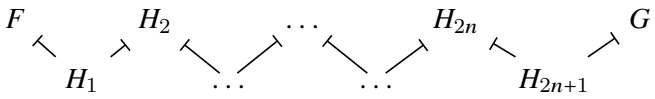
For the rest of the article, we will be more interested in the simulation relation $F \vdash G$ than any particular simulation function witnessing it: any one will do.

Remark 3.15. Definition 3.14 refers to the existence of morphisms between two automata as described in the automata theory litterature [9]. The existence of such morphisms suffices for our purposes. We are a bit more restrictive here, as the outputs need to be the same in both automata, while in the usual definition of morphisms, automata can have different output alphabets, as long as there is a function to convert one output into the other.

The simulation relation is not necessarily an equivalence relation and is not enough for our purpose, but we can use it to generate the equivalence relation we actually need.

Definition 3.16 (Internal equivalence relation on $\mathrm{DS}(X)$)**.** Let $F, G \in \mathrm{DS}(X)$.

We say that F *and* G *are simulation-equivalent*, and we write $F \sim G$, if there exists a finite sequence $(H_i)_{i \in N} \in \mathrm{DS}(X)$ such that:



It is not hard to check that:

Theorem 3.17. *The equivalence relation $\sim$ is the equivalence relation generated by $\vdash$, that is, $\sim$ is the smallest equivalence relation R such that $\vdash \subseteq R$.*

Finally, we need to show that the equivalence relation $\sim$ actually groups discrete systems that have the same behaviour as a stream transducer, in the sense of Definition 3.12; that is, the external and the internal equivalence relation are the same.

Lemma 3.18. *Let $F, G \in DS(X)$. If $F \equiv G$ then $\exists H \in DS(X)$ such that $H \vdash F$ and $H \vdash G$.*

Proof. Let $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right)$ and $G = \left(S_G, g^{\text{rdt}}, g^{\text{upd}}, s_{G,0}\right)$.

Take $H = \left(S_H, h^{\text{rdt}}, h^{\text{upd}}, s_{H,0}\right)$ such that:

- $S_H = \left\{(s, s') \in S_F \times S_G \mid \exists \overline{x^{\text{in}}} \text{ such that } F_{\text{res}}(\overline{x^{\text{in}}}) = s \text{ and } G_{\text{res}}(\overline{x^{\text{in}}}) = s' \right\}$
- $h^{\text{upd}}\,(x, (s, s')) = \left(f^{\text{upd}}\,(x, s), g^{\text{upd}}\,(x, s')\right)$
- $h^{\text{rdt}}\,(s, s') = f^{\text{rdt}}(s)$
- $s_{H,0} = (s_{F,0}, s_{G,0})$

Take as simulation functions the respective projections π_{S_F} and π_{S_G}.

It is easy to see that the required diagrams in Definition 3.14 do commute. For all $(s, s') \in S_H$, we have $h^{\text{rdt}} \circ \pi_{S_F}\,(s, s') = f^{\text{rdt}}(s)$ by defintion of h^{rdt}. Also, for all $(s, s') \in S_H$, $h^{\text{rdt}} \circ \pi_{S_G}\,(s, s') = f^{\text{rdt}}(s) = g^{\text{rdt}}(s')$ because there exists some stream $\overline{x^{\text{in}}}$ such that F results in s and G results in s'; besides, as $F \equiv G$, we have $F\left(\overline{x^{\text{in}}}\right) = G\left(\overline{x^{\text{in}}}\right)$, which implies $f^{\text{rdt}}(s) = g^{\text{rdt}}(s')$. Consequently, the diagrams commute and H simulates both F and G.

□

Lemma 3.19. *Let $F, G \in DS\,(X)$. If $F \vdash G$ then $F \equiv G$.*

Proof. Follows by induction on the length of an input stream $\left(x_i^{\text{in}}\right)_{i \in n} \in \widehat{X^{\text{in}}}$. □

Theorem 3.20. *$\forall F, G \in DS\,(X), F \equiv G \;\Leftrightarrow\; F \sim G$; or equivalently: $\equiv = \sim$.*

Proof. To prove $\equiv = \sim$, we need $\equiv \subseteq \sim$ and $\sim \subseteq \equiv$.

Suppose first that $F, G \in \mathrm{DS}\,(X)$ are dynamical systems such that $F \equiv G$. According to Lemma 3.18, there exists a $H \in \mathrm{DS}\,(X)$ such that $H \vdash F$ and $H \vdash G$, and hence $F \sim G$ by Definition 3.16. This establishes $\equiv \subseteq \sim$.

We now show that $\sim \subseteq \equiv$. According to Proposition 3.13, $\equiv$ is an equivalence relation, and according to Lemma 3.19, $\equiv$ contains $\vdash$. Theorem 3.17 states that $\sim$ is the smallest equivalence relation that contains $\vdash$; necessarily, we have $\sim \subseteq \equiv$. □

The goal of this article is to show that the behaviour of a general discrete system can be emulated by some specific wiring of some other discrete system, chosen with constraints (for example, on its internal structure). As far as we know, this result cannot be obtained with a pure equality. However, we have a description of what it means to be equivalent, both from an internal and from an external point of view, with the assurance that, seen as a blackbox, the "inhabited" box remains unchanged.

As we are not using real equalities, we need to define relations between sets that correspond to the usual inclusion and equality.

Definition 3.21 (Inclusion/equality up to equivalence)**.** Let $A, B \subseteq \mathrm{DS}\,(X)$. We consider the equivalence relation $\sim$ from Definition 3.16 (or, equivalently, in Definition 3.12).

We say that *A is a subset of B up to equivalence*, and we write $A \sqsubseteq B$, when $\forall a \in A, \exists b \in B, a \sim b$.

We say that *A is equal to B up to equivalence*, or *A is equivalent to B*, and we write $A \approx B$, when $A \sqsubseteq B$ and $A \sqsupseteq B$.

If $F, G : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ are functors, then we write $F \sqsubseteq G$ when, for all box X, we have $F(X) \sqsubseteq G(X)$. We write $F \approx G$, when $F \sqsubseteq G$ and $F \sqsupseteq G$.

If $M, N : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ are mappings (not necessarily functors), then we write $M \subseteq N$ when, for all boxes X and Y, we have $M(X) \subseteq N(X)$ and $\mathrm{Mor}_{\mathbf{Sets}}(M(X), M(Y)) \subseteq \mathrm{Mor}_{\mathbf{Sets}}(N(X), N(Y))$.

4. Main results

Before we introduce the actual results of the paper, we need a few more notions.

4.1. Algebras and closures

Definition 4.1 (Algebra)**.** Given a monoidal category $\mathscr{C}$, a functor $F : \mathscr{C} \to \mathbf{Sets}$ is called an *algebra over* $\mathscr{C}$ when it is a lax monoidal functor.

In our case, $\mathscr{C} = \mathscr{W}_{\mathbf{Sets}}$, and DS is an algebra by Theorem 3.8.

Definition 4.2 (Subalgebra)**.** Let $A : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ be an algebra over $\mathscr{W}_{\mathbf{Sets}}$. Let $\sigma_{X,Y} : A(X) \times A(Y) \to A(X \boxplus Y)$ denote its first coherence map (we recall that $\boxplus$ is the parallel composition of boxes (cf. Definition 2.28)).

A functor $B : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ is called a *subalgebra of* A when:

- $\forall X \in \mathscr{W}_{\mathbf{Sets}},\ B(X) \subseteq A(X)$
- $\forall X, Y \in \mathscr{W}_{\mathbf{Sets}}, \forall F \in B(X), \forall G \in B(Y), \sigma_{X,Y}(F, G) \in B(X \boxplus Y)$
- $\forall \varphi : X \to Y \in \mathscr{W}_{\mathbf{Sets}}, \forall F \in B(X), A(\varphi)(F) \in B(Y)$

Here, A and B are functors that transform boxes into sets. In our setting, the conditions can be interpreted as follows:

- (First item) Discrete systems generated by B are included in those generated by A;
- (Second item) The parallel composition of two discrete systems F and G generated by B is also generated by B.
- (Third item) B is stable through wiring diagrams: wiring a discrete system generated by B gives another discrete system generated by B.

Note that a subalgebra is itself an algebra.

Definition 4.3 (Closure)**.** Let $A : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ be an algebra over $\mathscr{W}_{\mathbf{Sets}}$.

Let $B : \mathrm{Ob}_{\mathscr{W}_{\mathbf{Sets}}} \to \mathrm{Ob}_{\mathbf{Sets}}$ be any map such that $\forall X \in \mathscr{W}_{\mathbf{Sets}},\ B(X) \subseteq A(X)$. The closure of B, denoted $\mathrm{Clos}(B)$, is the intersection of all subalgebras of A that contain $B(X)$ for all $X \in \mathscr{W}_{\mathbf{Sets}}$. (Any intersection of subalgebras is a subalgebra.)

The closure of a map B can be understood as the minimal lax monoidal functor (or algebra) containing B.

4.2. Memoryless systems

Our first main result concerns the subclass of discrete systems that we call *memoryless*. We show that wiring together memoryless systems can lead to systems that have memory.

Definition 4.4 (Memoryless discrete systems)**.** Let $X = \left(X^{\text{in}}, X^{\text{out}}\right)$ be a box.

A *memoryless discrete system for the box* X, or *memoryless discrete system* for short, is a discrete system $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}(X)$ such that f^{upd} immediately discards the previous state and uses only the current input; more precisely, such that f^{upd} factors as

$$f^{\text{upd}} = \widehat{X^{\text{in}}} \times S_F \xrightarrow{\pi_{\widehat{X^{\text{in}}}}} \widehat{X^{\text{in}}} \xrightarrow{f_u} S_F$$

for some $f^u : \widehat{X^{\text{in}}} \to S_F$.

We denote by $\text{DS}^{\text{ML}}(X)$ the set of all memoryless discrete systems for the box X : $\text{DS}^{\text{ML}}(X) = \left\{\left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}(X) \mid \exists f^u : \widehat{X^{\text{in}}} \to S_F,\ f^{\text{upd}} = f^u \circ \pi_{\widehat{X^{\text{in}}}}\right\}$.

We call these discrete systems *memoryless* because we see the states as a kind of memory (as in Example 3.4). The discrete systems defined above transition from one state to another without checking their current state, i.e. without checking their memory.

The following definition is a natural restriction of memoryless discrete systems; as these systems are memoryless, the only goal of their states is to produce the output via their readout function. The simplest case is when the readout function is the identity.

Definition 4.5 (Direct-output discrete systems)**.** Let $X = \left(X^{\text{in}}, X^{\text{out}}\right)$ be a box.

A *direct-output memoryless discrete system for the box* X, or *direct-output discrete system* for short, is a discrete system $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}(X)$ such that:

- $S_F = \widehat{X^{\text{out}}}$
- $f^{\text{rdt}} = \text{id}_{\widehat{X^{\text{out}}}}$
- $f^{\text{upd}} = f^u \circ \pi_{\widehat{X^{\text{in}}}}$ for some $f^u : \widehat{X^{\text{in}}} \to \widehat{X^{\text{out}}}$

We denote by $\text{DS}^{\text{ML}}_{\text{out}}(X)$ the set of all direct-output discrete systems for the box X:

$$\begin{aligned}\text{DS}^{\text{ML}}_{\text{out}}(X) &= \left\{\left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}^{\text{ML}}(X) \mid S = \widehat{X^{\text{out}}}, f^{\text{rdt}} = \text{id}_{\widehat{X^{\text{out}}}}\right\} \\ &= \left\{\left(\widehat{X^{\text{out}}}, \text{id}_{\widehat{X^{\text{out}}}}, f^{\text{upd}}, S_0\right) \in \text{DS}(X) \mid \exists f^u : \widehat{X^{\text{in}}} \to \widehat{X^{\text{out}}},\ f^{\text{upd}} = f^u \circ \pi_{\widehat{X^{\text{in}}}}\right\}\end{aligned}$$

Remark 4.6. The maps $\text{DS}^{\text{ML}} : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ and $\text{DS}^{\text{ML}}_{\text{out}} : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ are not functors, because they are not closed under wiring. Indeed, the whole point is that the result of wiring together memoryless systems is not necessarily memoryless.

We can now prove one of the main results of this paper, which is that every discrete system can be obtained by memoryless systems and feedback loops. The feedback loop is responsible for holding the state that was originally in the discrete system.

Here is the formal statement.

Theorem 4.7. *Clos* $\left(DS^{ML}_{out}\right) \approx DS$.

Proof. We have $\text{DS}^{\text{ML}}_{\text{out}} \subseteq \text{DS}^{\text{ML}} \subseteq \text{DS}$, so $\text{Clos}\left(\text{DS}^{\text{ML}}_{\text{out}}\right) \subseteq \text{DS}$, thus $\text{Clos}\left(\text{DS}^{\text{ML}}_{\text{out}}\right) \sqsubseteq \text{DS}$. We need the opposite inclusion (up to equivalence) $\text{Clos}\left(\text{DS}^{\text{ML}}_{\text{out}}\right) \sqsupseteq \text{DS}$.

Let $Y = \left(Y^{\text{in}}, Y^{\text{out}}\right) \in \mathscr{W}_{\mathbf{Sets}}$, and let $G = \left(S_G, g^{\text{rdt}}, g^{\text{upd}}, s_{G,0}\right) \in \text{DS}\left(Y\right)$. We will find $X \in \mathscr{W}_{\mathbf{Sets}}$, $F \in \text{DS}^{\text{ML}}_{\text{out}}\left(X\right)$ and $\varphi : X \to Y$ such that $\text{DS}\left(\varphi\right)\left(F\right) \sim G$.

Let $\delta_{S_G} = \langle S_G \rangle \in \mathbf{TFS}_{\mathbf{Sets}}$ be the list with one element, S_G, and consider the box $\left(\delta_{S_G}, \delta_{S_G}\right)$ with only that port on the left and the right. We define X as the parallel composition of this box $\left(\delta_{S_G}, \delta_{S_G}\right)$ and Y, that is:

$$\begin{aligned} X &= \left(X^{\text{in}}, X^{\text{out}}\right) \\ &= \left(\delta_{S_G}, \delta_{S_G}\right) \boxplus Y \\ &= \left(\delta_{S_G} + Y^{\text{in}}, \delta_{S_G} + Y^{\text{out}}\right) \end{aligned}$$

Note that $\widehat{X^{\text{in}}} = S_G \times \widehat{Y^{\text{in}}}$ and $\widehat{X^{\text{out}}} = S_G \times \widehat{Y^{\text{out}}}$. Thus, if $\overline{x^{\text{in}}} \in \widehat{X^{\text{in}}}$, then $\overline{x^{\text{in}}} = \left(s, \overline{y^{\text{in}}}\right)$. Similarly, if $\overline{x^{\text{out}}} \in \widehat{X^{\text{out}}}$, then $\overline{x^{\text{out}}} = \left(s, \overline{y^{\text{out}}}\right)$.

We choose $\varphi : X \to Y$ as the pair $\left(\varphi^{\text{in}}, \varphi^{\text{out}}\right)$ of coproduct inclusions:

- $\varphi^{\text{in}} : \left\{ \begin{array}{ccc} \delta_{S_G} + Y^{\text{in}} & \longrightarrow & Y^{\text{in}} + \delta_{S_G} + Y^{\text{out}} \\ x & \longmapsto & x \end{array} \right.$
- $\varphi^{\text{out}} : \left\{ \begin{array}{ccc} Y^{\text{out}} & \longrightarrow & \delta_{S_G} + Y^{\text{out}} \\ x & \longmapsto & x \end{array} \right.$

It follows from 2.35 that their dependent products $\widehat{\varphi^{\text{in}}} : \widehat{Y^{\text{in}}} \times S_G \times \widehat{Y^{\text{out}}} \to S_G \times \widehat{Y^{\text{in}}}$ and $\widehat{\varphi^{\text{out}}} : S_G \times \widehat{Y^{\text{out}}} \to \widehat{Y^{\text{out}}}$ are projections.

Recall that the goal is to find $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}^{\text{ML}}_{\text{out}}\left(X\right)$ such that $\text{DS}\left(\varphi\right)\left(F\right) \sim G$. So define F as follows:

- $S_F = \widehat{X^{\text{out}}} = S_G \times \widehat{Y^{\text{out}}}$
- $f^{\text{rdt}} = \text{id}_{\widehat{X^{\text{out}}}}$
- $f^{\text{upd}}\left(\overline{x^{\text{in}}}, \overline{x^{\text{out}}}\right) = f^{\text{upd}}\left(s, \overline{y^{\text{in}}}, s', \overline{y^{\text{out}}}\right) = \left(g^{\text{upd}}\left(\overline{y^{\text{in}}}, s\right), g^{\text{rdt}}\left(g^{\text{upd}}\left(\overline{y^{\text{in}}}, s\right)\right)\right)$
- $s_{F,0} = \left(s_{G,0}, g^{\text{rdt}}\left(s_{G,0}\right)\right)$

It is easy to see that F is in $\text{DS}^{\text{ML}}_{\text{out}}\left(X\right)$ because f^{rdt} and f^{upd} have the correct form. So let $\left(S_H, h^{\text{rdt}}, h^{\text{upd}}, s_{H,0}\right) = \text{DS}\left(\varphi\right)\left(F\right)$; we need to show it is equivalent to G. We compute each part of $\text{DS}\left(\varphi\right)\left(F\right)$ according to Definition 3.5.

Its state set is as follows:

$$S_H = S_F = \widehat{X^{\text{out}}} = S_G \times \widehat{Y^{\text{out}}}$$

Its readout function is defined on an arbitrary x^{out} as follows:

$$\begin{aligned} h^{\text{rdt}}\left(\overline{x^{\text{out}}}\right) &= h^{\text{rdt}}\left(s, \overline{y^{\text{out}}}\right) \\ &= \widehat{\varphi^{\text{out}}}\left(f^{\text{rdt}}\left(s, \overline{y^{\text{out}}}\right)\right) \\ &= \pi_{\widehat{Y^{\text{out}}}}\left(\text{id}_{\widehat{X^{\text{out}}}}\left(s, \overline{y^{\text{out}}}\right)\right) \\ &= \overline{y^{\text{out}}} \end{aligned}$$

Its update function is defined on an arbitrary $\left(\overline{y^{\text{in}}}, s, \overline{y^{\text{out}}}\right)$ as follows:

$$\begin{aligned} h^{\text{upd}}\left(\overline{y^{\text{in}}}, s, \overline{y^{\text{out}}}\right) &= f^{\text{upd}}\left(\widehat{\varphi^{\text{in}}}\left(\overline{y^{\text{in}}}, f^{\text{rdt}}\left(s, \overline{y^{\text{out}}}\right)\right), \left(s, \overline{y^{\text{out}}}\right)\right) \\ &= f^{\text{upd}}\left(s, \overline{y^{\text{in}}}, s, \overline{y^{\text{out}}}\right) \\ &= \left(g^{\text{upd}}\left(\overline{y^{\text{in}}}, s\right), g^{\text{rdt}}\left(g^{\text{upd}}\left(\overline{y^{\text{in}}}, s\right)\right)\right) \end{aligned}$$

Finally, its start state is as follows:

$$s_{F,0} = \left(s_{G,0}, g^{\text{rdt}}\left(s_{G,0}\right)\right)$$

Consequently, the following diagram commutes:

$$\begin{array}{lccccc} G: & \widehat{Y^{\text{in}}} \times S_G & \xrightarrow{g^{\text{upd}}} & S_G & \xrightarrow{g^{\text{rdt}}} & \widehat{Y^{\text{out}}} \\ & \downarrow{\scriptstyle \text{id}_{\widehat{X^{\text{in}}}} \times \alpha} & & \downarrow{\scriptstyle \alpha} & & \uparrow{\scriptstyle =} \\ \text{DS}(\varphi)(F): & \widehat{Y^{\text{in}}} \times S_G \times \widehat{Y^{\text{out}}} & \xrightarrow[h^{\text{upd}}]{} & S_G \times \widehat{Y^{\text{out}}} & \xrightarrow[h^{\text{rdt}}]{} & \widehat{Y^{\text{out}}} \end{array}$$

where $\alpha = \left(\text{id}_{S_G}, g^{\text{rdt}}\right)$. This yields $G \vdash \text{DS}(\varphi)(F)$ and hence $\text{DS}(\varphi)(F) \sim G$, which concludes the proof. □

Corollary 4.8. *Clos* $\left(DS^{ML}\right) \approx DS$.

Corollary 4.9. *For all* $G \in DS(Y)$*, if* G *has finite state set, then there exists* $H \in$ *Clos* $\left(DS^{ML}\right)(Y)$ *with finite state set such that* $H \sim G$.

Proof. In the proof of Theorem 4.7, take $H = \text{DS}(\varphi)(F)$, but instead of $S_F = S_G \times \widehat{Y^{\text{out}}}$, take $S_F = S_G \times g^{\text{rdt}}(S_G) \subseteq S_G \times \widehat{Y^{\text{out}}}$. If S_G is finite, so is S_F.

In that case, H is no more in Clos $\left(\text{DS}^{\text{ML}}_{\text{out}}\right)(Y)$ but in Clos $\left(\text{DS}^{\text{ML}}\right)(Y)$. □

Corollary 4.10. *(Assuming the axiom of choice) For all* $G \in DS(Y)$*, if* G *has an infinite state set, then there exists* $H \in$ *Clos* $\left(DS^{ML}\right)(Y)$ *with a state set of the same cardinality as* G*, such that* $H \sim G$.

Proof. In the proof of Theorem 4.7, take $H = \text{DS}(\varphi)(F)$, but instead of $S_F = S_G \times \widehat{Y^{\text{out}}}$, take $S_F = S_G \times g^{\text{rdt}}(S_G) \subseteq S_G \times \widehat{Y^{\text{out}}}$. The axiom of choice gives card $(S_F) =$ card $(S_G) \times$ card $\left(g^{\text{rdt}}(S_G)\right) =$ card (S_G).

In that case, H is no more in Clos $\left(\text{DS}^{\text{ML}}_{\text{out}}\right)(Y)$ but in Clos $\left(\text{DS}^{\text{ML}}\right)(Y)$. □

Theorem 4.7 states that systems without memory can be wired together to form systems with memory. In fact, the result is more subtle. It states that for any discrete system, we can find (or build) a memoryless discrete system with the certain wiring such that both systems are equivalent as stream transducers. The internal equivalence relation described in Theorem 3.20 is instrumental to prove Theorem 4.7, while the result is stated with regard to the external equivalence relation.

4.3. Finite-state systems

The second result is a refinement of Theorem 4.7, and is somewhat similar to it. We show that wiring together two-state discrete systems can generate a finite-state system with memory.

We can view the result as the generalisation of transistors being wired together in order to build a computer, or a system of neurons wired together to form a brain with finite memory.

Definition 4.11 (Finite-state systems)**.** Let $X = \left(X^{\text{in}}, X^{\text{out}}\right)$ be a box.

A *finite-state discrete system for the box* X, or *finite-state system* for short, is a discrete system $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}(X)$ such that S_F is a finite set.

We denote by $\text{DS}_{\text{Fin}}(X)$ the set of all finite-state discrete systems for the box X : $\text{DS}_{\text{Fin}}(X) = \left\{\left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}(X) \mid \text{card}(S_F) \in \mathbb{N}\right\}$. For a wiring diagram ϕ, we set $\text{DS}_{\text{Fin}}(\varphi) = \text{DS}(\varphi)$.

It is easy to see that:

Proposition 4.12. *The map* $DS_{Fin} : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ *is a subalgebra of DS.*

Proof. Follows from Definition 4.2. □

Definition 4.13 (Boolean systems)**.** Let $X = \left(X^{\text{in}}, X^{\text{out}}\right)$ be a box.

A *boolean memoryless discrete system for the box* X, or *boolean system* for short, is a discrete system $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}_{\text{Fin}}(X)$ such that F is memoryless and $S_F = 2^n = \{0,1\}^n$.

We denote by $\text{DS}^{\text{ML}}_{\text{Bool}}(X)$ the set of all boolean memoryless discrete systems for the box X : $\text{DS}^{\text{ML}}_{\text{Bool}}(X) = \left\{\left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right) \in \text{DS}^{\text{ML}}(X) \mid S_F = \{0,1\}^n\right\}$.

Remark 4.14. The map $\text{DS}^{\text{ML}}_{\text{Bool}} : \mathscr{W}_{\mathbf{Sets}} \to \mathbf{Sets}$ is not a functor, for the same reason as in remark 4.6.

Lemma 4.15. $DS^{ML}_{Bool} \approx DS^{ML} \cap DS_{Fin}$.

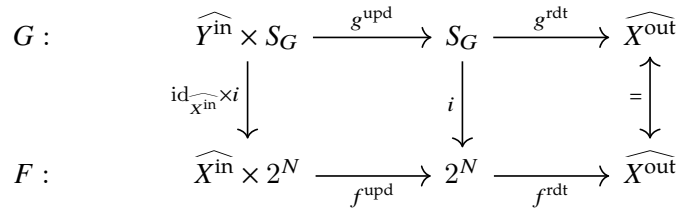
Proof. By construction, $\text{DS}^{\text{ML}}_{\text{Bool}} \subseteq \text{DS}^{\text{ML}} \cap \text{DS}_{\text{Fin}}$, so $\text{DS}^{\text{ML}}_{\text{Bool}} \sqsubseteq \text{DS}^{\text{ML}} \cap \text{DS}_{\text{Fin}}$. We need to show the other inclusion, so let $G = \left(S_G, g^{\text{rdt}}, g^{\text{upd}}, s_{G,0}\right) \in \text{DS}^{\text{ML}}(X) \cap \text{DS}_{\text{Fin}}(X)$, and it suffices to show that there is $F \in \text{DS}^{\text{ML}}_{\text{Bool}}(X)$ with $G \sim F$.

We have $g^{\text{upd}} = g^u \circ \pi_{\widehat{X^{\text{in}}}}$ and S_G finite. Let $N = \lceil \log_2(\text{card}(S_G)) \rceil$. There exists an injection $i : S_G \to 2^N$ and a surjection $p : 2^N \to S_G$ such that $p \circ i = \text{id}_{S_G}$. This is just a binary encoding of S_G.

Define $F = \left(S_F, f^{\text{rdt}}, f^{\text{upd}}, s_{F,0}\right)$ such that:

- $S_F = 2^N$
- $f^{\text{rdt}} = g^{\text{rdt}} \circ p$
- $f^{\text{upd}} = g^{\text{upd}} \circ p$
- $s_{F,0} = i\left(s_{G,0}\right)$

Then the following diagram commutes:



We have $F \in \text{DS}^{\text{ML}}_{\text{Bool}}(X)$ and $G \vdash F$ (with i as simulation function), so $F \sim G$, hence the result. □

Lemma 4.16. *$DS_{Fin} \approx Clos\left(DS^{ML}\right) \cap DS_{Fin}$.*

Proof. Observe that $\text{DS}_{\text{Fin}} = \text{DS} \cap \text{DS}_{\text{Fin}}$. By Corollary 4.8, we have $\text{DS} \approx \text{Clos}\left(\text{DS}^{\text{ML}}\right)$. In particular, $\text{Clos}\left(\text{DS}^{\text{ML}}\right) \subseteq \text{DS}$, so $\text{Clos}\left(\text{DS}^{\text{ML}}\right) \cap \text{DS}_{\text{Fin}} \subseteq \text{DS} \cap \text{DS}_{\text{Fin}}$. This gives one inclusion, $\text{Clos}\left(\text{DS}^{\text{ML}}\right) \cap \text{DS}_{\text{Fin}} \sqsubseteq \text{DS} \cap \text{DS}_{\text{Fin}}$.

As for the reverse inclusion (up to equivalence), let X be a box and let $F \in (\text{DS} \cap \text{DS}_{\text{Fin}})(X)$. By Corollary 4.8, there exists $G \in \text{Clos}\left(\text{DS}^{\text{ML}}\right)$ such that $G \sim F$. By Corollary 4.9, we can choose G so that $G \in \text{DS}_{\text{Fin}}(X)$, hence the result. □

Theorem 4.17. *$Clos\left(DS^{ML}_{Bool}\right) \approx DS_{Fin}$.*

Proof. Clearly, $\text{Clos}\left(\text{DS}^{\text{ML}}_{\text{Bool}}\right) \sqsubseteq \text{DS}_{\text{Fin}}$. By Lemma 4.15, in order to prove $\text{DS}_{\text{Fin}} \sqsubseteq \text{Clos}\left(\text{DS}^{\text{ML}}_{\text{Bool}}\right)$, it suffices to prove $\text{DS}_{\text{Fin}} \sqsubseteq \text{Clos}\left(\text{DS}^{\text{ML}} \cap \text{DS}_{\text{Fin}}\right)$. Furthermore, since $\text{DS}_{\text{Fin}} \approx \text{Clos}\left(\text{DS}^{\text{ML}}\right) \cap \text{DS}_{\text{Fin}}$ (Lemma 4.16), we reduce to proving that: $\text{Clos}\left(\text{DS}^{\text{ML}}\right) \cap \text{DS}_{\text{Fin}} \sqsubseteq \text{Clos}\left(\text{DS}^{\text{ML}} \cap \text{DS}_{\text{Fin}}\right)$.

Let $Y \in \mathscr{W}_{\mathbf{Sets}}$ be a box, and let $G \in \left(\text{Clos}\left(\text{DS}^{\text{ML}}\right) \cap \text{DS}_{\text{Fin}}\right)(Y)$. We have $G \in \text{Clos}\left(\text{DS}^{\text{ML}}\right)(Y)$, so according to Corollary 4.8, there exist a box $X \in \mathscr{W}_{\mathbf{Sets}}$, a wiring diagram $\varphi : X \to Y$, and a $F \in \text{DS}^{\text{ML}}(X)$ such that $\text{DS}(\varphi)(F) \sim G$. Furthermore, according to Corollary 4.9, we can choose F with finite state set. Finally, $F \in \left(\text{DS}^{\text{ML}} \cap \text{DS}_{\text{Fin}}\right)(X)$, and $\text{DS}(\varphi)(F) \sim G$, so $\text{Clos}\left(\text{DS}^{\text{ML}}\right) \cap \text{DS}_{\text{Fin}} \sqsubseteq \text{Clos}\left(\text{DS}^{\text{ML}} \cap \text{DS}_{\text{Fin}}\right)$, hence the result.

□

5. Conclusion

Boxes are empty frames that condition the inputs and outputs of their content, a generalisation of automata called discrete systems. Such systems come with a state set that represents their memory of previous inputs. In a sense, discrete systems can learn. However, we can define a subclass of discrete systems that do not store any experience of their past. We see these as reactive, in the sense that they still react to any input, but their past experience does not influence that reaction. Unlike typical discrete systems, they do not keep a memory of the previous inputs.

In this paper, we use a category-theoretic framework to give a constructive proof that any discrete system with memory can be simulated by some correctly-wired memoryless system. This result can be understood as a phenomenon of emergence in a complex system.

This construction opens a number of new questions. A possible question might consist in finding the "best" memoryless system, where "best" could depend on the definition of some valuation function, e.g. the most parsimonious in terms of state set. A similar question could be asked with respect to wiring diagrams, whose number of feedback loops could be bounded by a cost function.

Possible extensions of this work could concern dynamical systems other than DS. For instance, can we establish the same kind of results when considering measurable or continuous dynamical systems?

References

[1] S. J. Russel and P. Norvig, *Artificial Intelligence: a modern approach*, 3rd ed., ser. Prentice Hall series in artifical intelligence. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010.

[2] D. I. Spivak, "The steady states of coupled dynamical systems compose according to matrix arithmetic," Available online: https://arxiv.org/abs/1512.00802, 2016.

[3] S. Awodey, *Category Theory*, 2nd ed., ser. Oxford Logic Guides. Oxford University Press, Oxford, 2010, vol. 52.

[4] M. Barr and C. Wells, *Category Theory for Computing Science*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998.

[5] F. Borceux, *Handbook of Categorical Algebra*, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994, vol. 2.

[6] M. Behrisch, S. Kerkhoff, R. Pöschel, F. M. Schneider, and S. Siegmund, "Dynamical systems in categories," *Applied Categorical Structures*, vol. 25, no. 1, pp. 29–57, Feb 2017. [Online]. Available: https://doi.org/10.1007/s10485-015-9409-8

[7] K. Kunen, *Set theory - An introduction to independence proofs*, 7th ed., ser. Studies in logic and the foundations of mathematics. North-Holland Publishing Company, 1999, vol. 102.

[8] M. Sipser, *Introduction to the theory of computation*, 3rd ed. Cengage Learning, 2012.

[9] J. Adamek and V. Trnkova, *Automata and Algebras in Categories*, 1st ed. Norwell, MA, USA: Kluwer Academic Publishers, 1990.

OUR WORLDWIDE PARTNERS UNIVERSITIES - DOUBLE DEGREE AGREEMENTS

3 CAMPUS, 1 SITE

IMT Atlantique Bretagne–Pays de la Loire – **http://www.imt-atlantique.fr/**

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler
CS 20722
44307 Nantes Cedex 3
France
T +33 (0)2 51 85 81 00
F +33 (0)2 99 12 70 08

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
T +33 (0)2 99 12 70 00
F +33 (0)2 51 85 81 99

Site de Toulouse
10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
T +33 (0)5 61 33 83 65

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Imprimé à IMT Atlantique
Dépôt légal : Mai 2018
ISSN : 2556-5060