Block Diagrams for Categorical Cybernetics

Michael Zargham
BlockScience
Tempe, AZ
zargham@block.science

Jamsheed Shorish
BlockScience
Tempe, AZ
jamsheed@block.science

***Abstract*—Block Diagrams play a central role in the design and analysis of complex engineered systems, in large part due to their capacity to represent causal flow of material, energy and information through systems whose implementations span domains. This property is especially valuable in the context of cybernetic systems where the implementations of any particular function may be behavioral. When designing systems to co-regulate human, machine and environmental processes it often does not suffice to work with dynamical systems whose state spaces are restricted to vector spaces. This paper leverages applied category theory to derive a formalism sufficient to represent Generalized Dynamical Systems (cf. e.g. [1]) using Block Diagrams. The work is motivated by applications of cybernetics to build institutions with an increasing amount of technical automation serving to regulate human behavior.**

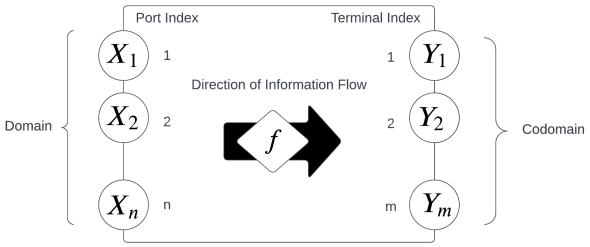
I. Introduction

The design and analysis of complex systems have long been areas of interest in various disciplines, particularly in control theory and systems engineering. In control theory, *block diagrams* are a fundamental tool for representing and analyzing systems [2]. Block diagrams provide formal logical relations to represent the flow of information (including matter, energy and digital signals) which support 'zooming, tearing and linking' operations [3]. Classical control theoretic block diagrams are, however, often limited in their ability to model complex systems, due to their focus on real-valued vector fields. This limits their application to Generalized Dynamical Systems (GDS), which extend the dynamical systems formalism to topological spaces and structured representations such as data schemas [1], [4]. In addition, the consistent application of block diagrams to computer and physical implementations requires their formal representations to be consistent with both type theory and (typed) lambda calculus [5], [6]. While lambda calculus natively supports single-input single-output (SISO) systems, and supports typed multiple-input, multiple-output (MIMO) systems via *partial functions*, it is nevertheless of use to define a formal basis for MIMO block diagrams. These are structurally similar to control theoretic block diagrams, but are extended (cf. Figure 1) to allow representations of operations from a domain (a collection of spaces) to a codomain (also a collection of spaces).

This work is motivated by the desire to apply model-based systems engineering to design and test assemblages of humans and machines. Such cybernetic assemblages should both satisfy constraints (be safe relative to agreed-upon criteria or requirements) and pursue goals (be performant with respect to agreed-upon metrics). The concept of 'categorical cybernetics' is related to the application of *category theory* to these assemblages [7]. In this way category theory can help build relations between existing concepts [8]; concepts of interest include analog circuits [9], databases [10] and games [11], among others.

Where applied category theory uses string diagrams, systems engineering uses other modeling techniques [12]. Although we focus on functional block diagrams, we distinguish this from block definition diagrams (BDD) which are supported by a systems engineering focused extension of the unified modeling language (UML), called the systems modeling language (SysML) [13]. Block diagrams are used for functional modeling [14], and are a popular computer aided design tool in MathWorks' System Composer [15]. Block diagrams are used to represent increasingly complex systems with (both discrete and continuous) state space representations, evolving in the time domain [16], [17].

Our approach aims to knit together applications of cybernetic principles to higher order systems arising in several distinct research threads (examples include, but are not limited to, cybernetics 2.0 [18], institutional design [19], and decentralized autonomous organizations (DAOs) [20]). We emphasize the importance of safety in the context of designing complex systems, drawing from the work of Leveson [21]. By introducing the *the category of blocks* we aim to provide a more robust and versatile framework for representing and analyzing complex systems, paving the way for novel design and analysis techniques that can tackle the challenges of modern engineering.



$$f : X_1 \times \cdots \times X_n \to Y_1 \times \cdots \times Y_m$$

Figure 1: The Anatomy of a Block

This section has provided a review of relevant literature and motivation for revisiting block diagrams using applied category theory. Section II is a practical review of block diagrams, emphasizing their structure and use for modeling systems. Section III presents a category of types, while Section IV develops the category of nested schemas—both are then leveraged in Section V to characterize the goal of this work, the category of blocks. Finally, in Section VI the constructed mathematical objects are discussed relative to the dynamical systems motivation offered in this section.

II. Block Diagrams Overview

Block diagrams represent a wiring of input-output relationships. A block has a set of inputs, called a *domain*, and a set of outputs, called a *codomain*. The elements of the domain are called *ports*, and the elements of the codomain are called *terminals*—in the language of dynamical systems, the elements of the domain and codomain are *spaces*. One may wire a terminal to a port only if the terminal and the port are the same space.[1] Control systems rely on blockdiagrams as tools for abstracting the details of subsystems, and focus on using *signals* made available by *sensors* to estimate state and compute control actions which drive the true state towards the target state. As a motivating example, a block diagram of a control system is presented in Figure 2.

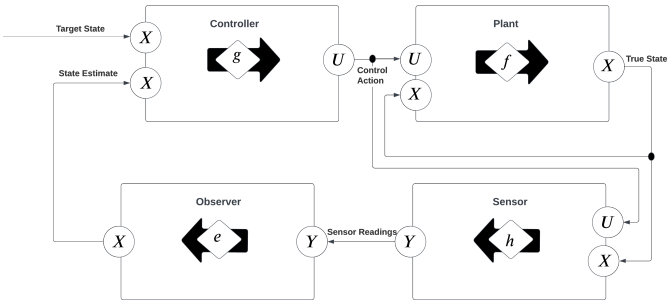


Figure 2: Block Diagram Representing a Control System

A category of nested schemas, built upon a category of types, is constructed to provide structure (and composition rules) for the "shapes" of spaces which type the wires in block diagrams. Our goal is to move beyond the relatively informal presentation of block diagrams from Section II to a formal definition which ensures that block diagrams map to executable programs for use in Model Based Systems Engineering (MBSE) [22].

[1]Our Block diagrams depart from string diagrams precisely in the directedness of these relationships—string Diagrams do not distinguish ports from terminals. The authors contend that both block diagrams and string diagrams are useful tools.

III. A Category of Types

The category of types[2] is a mathematical structure that provides a framework for studying sets and functions between sets. In this section we review the main properties of the category for a reader who is familiar with the concepts but not the details. Throughout we denote the category of types by $\mathcal{T}$.

A. Objects

The objects in the category of types are sets $t \in T := ob(\mathcal{T})$, and each set $t \in T$ is considered to be a type. Intuitively, types may represent both the shape of associated domains/codomains (e.g. the space of real numbers, $\mathbb{R}$, having the shape "real" or "float") *and* the operations that may be performed upon them. For example, a space whose shape is described by an attribute "float" and operations "add" and "scale" might represent a one-dimensional vector space, depending upon how (collectively) these attributes and operations are encoded as the 'type' of the space. Another example is a space with an attribute 'integer' and an operation conveying a *constraint*, such as a Boolean operation "isPrime?", which generates a subspace of the space of integers comprised of prime numbers, and the complement of that subspace. The combination of attribute(s) and operation(s) defines a subspace's type.

Allowing types to cover not just attributes but also operations provides a means of associating spaces (as shape types) with their representation as *schemas*, a fundamental requirement for implementing block diagrams. Schemas are discussed in more detail in Section IV, while the representation of schemas as types is treated in Section V.

B. Morphisms

The morphisms in the category of types are functions between types. Each function maps elements of one type (the domain) to elements of another type (the codomain).[3] Given two types t_1 and t_2, a morphism f from t_1 to t_2 is a function $f : t_1 \to t_2$ defined by the statement:

$$\textit{For all } x \in t_1, \textit{ there exists a } y \in t_2 \textit{ such that } f(x) = y;$$
$$\textit{there does not exist a } y' \in t_2,\ y \neq y' \textit{ such that } f(x) = y'.$$

C. Composition

The category of types has a well-defined composition operation that allows us to compose two morphisms to form a new morphism. The composition of two morphism f and g, denoted $g \circ f$, is a morphism that maps elements from the domain of f to the codomain of g via the

[2]To our knowledge there is no definitive reference formally defining 'the' category of types—in what follows we suggest a version sufficient for our need to assert block composition, based loosely upon the Category of Sets (cf. e.g. [23]).

[3]In the category of types all functions are single-input single-output (SISO), meaning the domain and codomain of the morphisms are not indexed lists (as was the case for the Block in Figure 1).

intermediate set that is the codomain of f and the domain of g. Mathematically, it is defined as:

$$(g \circ f)(x) := g(f(x)) \text{ for all } x \in \text{dom}(f), \tag{1}$$

where $\text{dom}(f)$ is a type corresponding to the domain of the morphism f. Composition also satisfies the *associativity* property, i.e., $(h \circ (g \circ f)) = (h \circ g) \circ f$ for all morphisms f, g, and h.

D. Identity Morphism

For each object (set) in the category of types, there is an identity morphism (function) that maps every element of a set to itself. This function serves as the identity function under composition, and is defined as:

$$id_t(x) := x \text{ for all } x \in t. \tag{2}$$

It satisfies the identity property, i.e., $id_{t_2} \circ f = f$ and $f \circ id_{t_1} = f$ for all morphisms $f : t_1 \to t_2$.

E. Functoriality

The category of types is a functorial category, meaning that it can be mapped to other categories in a way that preserves the structure of the category. This allows us to study the category of types in the context of other mathematical structures. Given two categories $\mathcal{C}$ and $\mathcal{D}$, a functor $F : \mathcal{C} \to \mathcal{D}$ maps objects and morphisms in $\mathcal{C}$ to objects and morphisms in $\mathcal{D}$, respectively. The functor must preserve the composition of morphisms and the identity morphism, meaning that if $f : t_1 \to t_2$ and $g : t_2 \to t_3$ are morphisms in $\mathcal{C}$, then $F(f)$ and $F(g)$ are morphisms in $\mathcal{D}$, and $F(g \circ f) = F(g) \circ F(f)$. Similarly, $F(id_{t_1}) = id_{F(t_1)}$ for any object t_1 in $\mathcal{C}$.

Functorality is of interest because functors are used to construct the category of blocks (cf. Section V), ensuring that it preserves the structure of the category of types.

IV. The Category of Nested Schemas

As a precursor to building spaces, we need a representation of the shape of those spaces. To do so we define the category of nested schemas. The category of nested schemas provides a framework for studying nested dictionaries whose values are types (cf. Figure 3). In this section, we will review the main properties of the category of nested schemas. We will denote the category of nested schemas by $\mathcal{D}$.

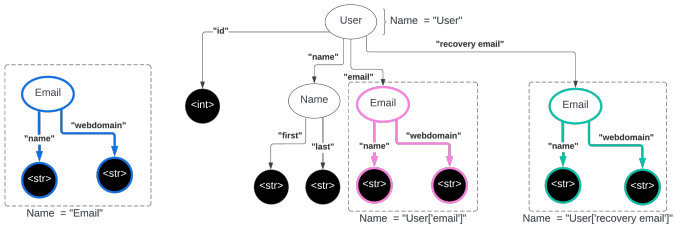


Figure 3: Nested Schemas are trees whose edges are string keys and whose subtrees are named according to their address in the tree.

A. Objects

The objects in the category of nested schemas are named nested dictionaries, where the keys are strings and each value is a type. Denote the set of objects $ob(\mathcal{D})$ by D.

A nested schema $d \in D$ may be represented as a directed tree: the *root node* is the schema's "name" which is a string, and its child nodes' "names" are their locations in the tree. *Leaf nodes* are objects in the category of types. Nodes which are not leaf nodes must be objects in the category of nested schemas, whose names are their keys. For any $d \in D$, all leaf nodes which are types are *addressed* by a sequence of strings starting with the "name" of d, proceeding along the edges in the tree (which are dictionary keys) until arriving at a type.

B. Morphisms

The morphisms in the category of nested schemas are functions that relate nested schemas to one another, and are described in what follows. Of note within this category are the *selection morphism*, indicating that one schema is a subschema of another, and the *congruence morphism*, indicating that two schemas are the same shape, but not necessarily the same schema.

1) Identity Morphism: For any nested schema $d \in D$, there is an identity morphism from d to itself. This is a trivial case of a selection morphism (see below), for which the address a_0 is an empty sequence. The identity morphism is the relationship between d and itself defined by the unique selection of entire tree. Two nested schemas $d_1, d_2 \in D$ are said to be *equal* if they are related by the identity morphism, denoted by $d_1 = d_2$.

2) Congruence Morphism: For any nested schemas $d_1, d_2 \in D$, there is a congruence morphism from d_1 to d_2 only if all nonempty key sequences a in d_1 and d_2 correspond to selection morphisms from d_1 to $d_1[a]$ and from d_2 to $d_2[a]$, respectively, and the resulting types of the leaf nodes match. This relationship is denoted $d_1 \cong d_2$. Congruence requires equality in all but name.[4]

3) Selection Morphism: For any nested schema $d \in D$, there is a selection morphism from d to each of its subschemas $d[a] \in D$, where a is the key sequence addressing $d[a]$ in d. Formally, we define a selection morphism as $a : d_1 \to d_2$ if $d_2 \cong d_1[a]$, meaning that $d_2 \in D$ has the same shape as the object addressed by key sequence a in d_1, and whose schema is the subtree of d_1 with $d_1[a]$ as its root node.

4) Membership: In the category of nested schemas, for any nested schema $d \in D$ and its subschema $d[a] \in D$, there is a membership relationship from $d[a]$ to d. This relationship is the inverse map of a selection morphism. Formally, we define a membership map as $a^{-1} : d_2 \to d_1$ if $d_2 = d_1[a]$, meaning that $d_2 \in D$ is precisely the object named by the address a in d_1 and is a subschema of d_1.

[4]It is also worth noting that equality between nested schemas implies congruence. That is, if $d_1 = d_2$, then $d_1 \cong d_2$.

Membership thus maps the subschema d_2 to its parent schema d_1.

5) Similarity: For any nested schemas $d_1, d_2 \in D$, there is a similarity relationship from d_1 to d_2 characterized by a pair of key sequences a_1 and a_2 such that $d_1[a_1] \cong d_2[a_2]$. A similarity relationship is not actually a morphism but rather a description of three morphisms: 2 selection morphisms and an equivalence morphism.

C. Composition

In the category of nested schemas, we can compose any two morphisms A_{12}, which maps from d_1 to d_2, and A_{23}, which maps from d_2 to d_3, via $A_{23} \circ A_{12}$. The properties of composition are explored further in what follows.

1) Unitality: The first important composition property is that $A_{12} \circ Id_{d_1} = A_{12}$ and $Id_{d_2} \circ A_{12} = A_{12}$. These follow directly from the definition of the identity morphism. The congruence morphisms also have identity-like properties, but can be distinguished from the identity morphism because $d_1 \cong d_2$ even when $d_1 \neq d_2$. Congruence tells us two schemas are the same shape but have different names (thus they are distinct objects in $\mathcal{D}$. The reason we don't want congruence to be equality is that these schema may have contextual interpretations which can only be distinguished by their names.

A_{12}	A_{23}	$A_{12} \circ A_{23}$
Congruence	Congruence	Congruence
Selection	Congruence	Selection
Congruence	Selection	Selection
Selection	Selection	Selection

Table I: This table summarizes the effects of combining morphisms

2) Transitivity of Congruence Morphisms: In the category of nested schemas, if two nested schemas d_1 and d_2 are related by the congruence morphism, denoted $d_1 \cong d_2$, and d_2 and d_3 are related by the congruence morphism, denoted $d_2 \cong d_3$, then d_1 and d_3 are also related by the congruence morphism, denoted $d_1 \cong d_3$. This property is known as the transitivity of congruence morphisms.

More formally, if for all key sequences a in d_1 and d_2, the selection morphisms from d_1 to $d_1[a]$ and from d_2 to $d_2[a]$ correspond, and the resulting types of the leaf nodes match, and similarly for d_2 and d_3, then for all key sequences a in d_1 and d_3, the selection morphisms from d_1 to $d_1[a]$ and from d_3 to $d_3[a]$ correspond, and the resulting types of the leaf nodes match.

The transitivity property is a crucial aspect of the category of nested schemas, as it allows us to build relationships between nested schemas that go beyond individual pairwise comparisons. Nested schemas may be compared on a higher level, helping to determine the extent to which schemas are similar or congruent to one another (even when they have different subtrees or leaf nodes). Lastly, transitivity is stronger than associativity.

3) Associativity of Morphisms: Selection morphisms are a special case of congruence morphisms where d_2 is a subschema of d_1, although for the purpose of demonstrating associativity of congruence morphisms, it does not matter if d_2 is a subschema of d_1. Let the congruence morphisms E_{ij} be the relations $d_i \cong d_j$ for any two congruent schemas d_i and d_j, irrespective of whether d_j is also a subschema of d_i. Associativity of congruence can be shown as:

$$(E_{12} \circ E_{23}) \circ E_{34} = E_{13} \circ E_{34} \tag{3}$$
$$= E_{14} \tag{4}$$
$$= E_{12} \circ E_{24} \tag{5}$$
$$= E_{12} \circ (E_{23} \circ E_{34}) \tag{6}$$

For good measure, the associativity of selection morphisms may also be considered. Let a_1 be a sequence of keys in d_1 which addresses a subschema $d_2 \cong d_1[a_1]$, let a_2 be a sequence of keys in d_2 which addresses a subschema $d_3 \cong d_2[a_2]$ and let a_3 be a sequence of keys in d_3 which addresses a subschema $d_4 \cong d_3[a_3]$. For ease of notation let ab denote the morphism created by concatenating the key sequences a and b. Then:

$$(a_1 \circ a_2) \circ a_3 = a_1 a_2 \circ a_3 \tag{7}$$
$$= a_1 a_2 a_3 \tag{8}$$
$$= a_1 \circ a_1 a_2 \tag{9}$$
$$= a_1 \circ (a_2 \circ a_3) \tag{10}$$

The intuition behind a chain of selection morphisms is that with each morphism one charts a path down the schema. It doesn't matter how one chooses to chop up the sequence of keys, there is a sequence a in d_1 arriving at a subschema with $d_4 \cong d_1[a]$. In fact, *any* sequence of selection morphisms characterized by key sequences a_i, which concatenate to form a, will compose up to the same morphism regardless of groupings. Furthermore, mixing in additional congruence morphisms does not change the result of the composition, because congruence preserves all subschemas and thus all sequences of keys traversing those subschemas.

D. Operations

As leaf nodes are objects in the category of types (cf. Section III-A), they may carry properties described by operations on the associated space. This is fundamental to identifying *subspaces* of spaces that fulfill conditions, such as constraints or other restrictions. Nested schemas which in the absence of operation types would appear congruent, become formally different once different operation types are included as part of their schema 'signature'.

A simple example is the 'isPrime?' constraint operation described in Section III-A, when added to a space with the 'integer' shape—the result is a subspace of the space that also has the 'integer' shape, but without the 'isPrime?' constraint operation. It is important to note that this means a space and a subspace defined by a constraint operation

cannot be placed into congruence, as there is no selection morphism that will match leaf node to leaf node when one space contains an operation in its nested schema that the other space lacks.

A more nuanced example is two copies of a space with the '{float, float, float}' three-dimensional real number shape, each possessing a different norm operation type (e.g. 'supNorm' or 'EuclidNorm' for the supremum norm and the Euclidean norm, respectively). Although each space represents $\mathbb{R}^3$, the operation for measuring distance, treated as part of the description of the space via the nested schema, is different—hence they are not congruent.[5]

From an implementation point of view, the usage of e.g. 'isPrime?' above as a condition (or constraint) operator is suggestive of the inclusion of general mappings between domains and codomains as functions between types, i.e. *blocks*. This motivates the ability to view, for a particular implementation, operations within a nested schema as *callables*. If, furthermore, such callables represent blocks in their own right, a nested schema is capable of encapsulating multiple block operations on a space as part of the space's shape. This facilitates the endowment of spaces with

1) **functionality**, e.g. 'add', 'scale' or other functional operations,
2) **constraints**, e.g. 'isPrime?' or other restrictions,
3) **meta-information**, e.g. norms or other metrics.

Including operations on spaces as part of their nested schema ensures that a mapping (i.e. a functor) can be defined mapping schemas from the category of nested schemas to types from the category of types, which is required to properly define the category of blocks in the following Section.

V. The Category of Blocks

A. Objects

The objects in the category of blocks $\mathcal{S}$ are objects in the category of types, i.e. they are sets $s \in T = ob(\mathcal{T})$. Denoting these objects collectively by $S = ob(\mathcal{S})$ we thus have $S \subset T$. We associate these objects formally to *spaces*, i.e. a particular set $s_X \in S$ is *identified* with a space X (cf. Section III-A). Note that S is a proper subset of T because there exist sets of types that do not represent Spaces.

B. Morphisms

The morphisms in the category of blocks are functions between types, restricted to those types in S: given such a function f with domain $\mathrm{dom}(f)$ and codomain $\mathrm{co}(f)$, it must be that $\mathrm{dom}(f)$, $\mathrm{co}(f) \subseteq S$. We associate these morphisms formally to blocks. Given a type s_X representing a space X, and a type s_Y representing a space Y, $s_X, s_Y \in S$, a block mapping X to Y is represented by a function $B_{XY} : s_X \to s_Y$.

[5]This also implies that there is no 1:1 mapping between schemas and spaces, adding structure to what will be defined as a functor between the category of nested schemas and the category of types in Section V.

C. Composition and Associativity

Blocks in the category of blocks, being morphisms between types, inherit composability from the category of types (cf. Section III-C). Intuitively, blocks may be 'chained' together such that the codomain of one block is the same as the domain of the succeeding block. This inheritance also implies that block composition is associative.

D. Identity Morphism

Since every type in the category of types possesses its own identity function, every space X with representation s_X has an identity function id_{s_X} such that $id_{s_X}(x) = x$, $\forall x \in s_X$.

E. Functoriality

A key property of the category of types is that it may be placed into relation with the category of nested schemas via a functor $\tau : \mathcal{D} \to \mathcal{S}$. Given a schema $d \in D$, the functor maps the schema to a type $\tau(d) \in S$, which we interpret as *the schema d representing the space defined by the type $\tau(d)$*.

The functor τ provides a means by which a block operation $\bar{B}_{XY} := \tau^{-1} \circ B_{XY} \circ \tau$ is defined between nested schemas $d_X, d_Y \in D$ with images (via τ) in S,

$$\begin{array}{ccc} s_X & \xrightarrow{B_{XY}} & s_Y \\ \uparrow{\scriptstyle \tau} & & \downarrow{\scriptstyle \tau^{-1}} \\ d_X & \xrightarrow{\bar{B}_{XY}} & d_Y \end{array}$$

provided a unique inverse of τ exists. Note that if instead more than one nested schema represents s_Y, then there must be a (usually implementation-dependent) selection criterion that chooses d_Y from the alternatives provided by $\tau^{-1}(s_Y)$.

Functoriality also allows us to state that since the congruence morphism of the category of nested schemas *partitions* D, there exists via the above relation a congruence morphism in the category of blocks. We interpret this morphism as saying that two congruent nested schemas (d_X and d_Y, say) are mapped, via τ, to two types ($\tau(d_X)$ and $\tau(d_Y)$) that *represent congruent spaces*, i.e. there is an induced congruence relation '$\cong$' such that $\tau(d_X) \cong \tau(d_Y)$. We may then assert that $X \cong Y$, i.e. that space X and space Y are congruent up to schema congruence.

This approach provides a linkage between block diagrams and their eventual implementation as e.g. computer code, as a framework can be developed to leverage nested schemas as "the" representation framework for spaces, and defining blocks between nested schemas as faithful representations of the morphisms within the category of blocks.

VI. Application to Dynamical Systems

A *dynamical system* typically attaches a time dimension to points in a state space, generating a *trajectory* that summarizes the change of a system over time (commencing from an initial state). Dynamical systems also often possess *boundary conditions* that are restrictions on how the system

may behave over time.[6] A block, as defined above, may represent a dynamical system if the following condition holds true:

Condition VI.1. *If the codomain of a block has a non-empty intersection with its domain, then there exists a dynamical system characterized by feeding the block's terminals back into its ports.*

The operational interpretation of the above is:

1) The intersection of codomain and domain is accomplished by the congruence of the nested schemas (spaces) resulting from the application of the selection morphism to each schema (codomain and domain representations). The resulting subschema so defined is the *state space*.
2) Subschemas that are not selected in the *codomain*, possibly containing operations (such as the formation of metrics), reflect *measurements* of the system.
3) Subschemas that are not selected in the *domain*, possibly containing operations (such as constraints), reflect the *boundary conditions* of the system.

From a block's 'wiring' perspective, the state space is represented by that subset of terminals that **feed back** to a subset of ports. Measurements are represented by the remaining terminals that feed forward, while boundary conditions are represented by those ports receiving as-yet under-specified signals from the environment.

Finally, we may describe the difference between a block which cannot be influenced by its outside environment, and that which can:

Definition VI.2. An **autonomous** dynamical system is (represented by) a block that fulfils Condition VI.1, and for which *all* ports may be wired from terminals.

Definition VI.3. A **non-autonomous** dynamical system is (represented by) a block that fulfills Condition VI.1 and at least one of the following:

1) There exists at least one domain subschema not representing the state space that is permitted to change over time (i.e. changing the boundary conditions).
2) The image of the block (treated as a mapping) is permitted to change over time (e.g. changing the parameter value of a parameterized block).

In conclusion, block diagrams can be used to represent GDS, by considering their formalization using applied category theory. This marriage of visual and formal language provides the basis for the application of mathematically advanced engineering methods to cybernetic systems whose state spaces are not well represented as vector spaces.

[6]In what follows we abstract away from *initial conditions*, which we interpret here as simply the starting value of a dynamical system, or (equivalently) the initial point of application of a block, or the initial value input to a port of a block's 'wiring'.

References

[1] M. Zargham and J. Shorish, "Generalized dynamical systems part I: Foundations," 2022.
[2] K. J. Åström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.
[3] J. C. Willems, "The behavioral approach to open and interconnected systems," *IEEE Control Systems Magazine*, vol. 27, no. 6, pp. 46–99, 2007.
[4] E. Roxin, "On generalized dynamical systems defined by contingent equations," *Journal of Differential Equations*, vol. 1, no. 2, pp. 188–205, 1965.
[5] A. Church, *A formulation of the simple theory of types*. Cambridge University Press, 1940, vol. 5.
[6] B. C. Pierce, *Types and Programming Languages*. MIT Press, 2002.
[7] M. Capucci, B. Gavranović, J. Hedges, and E. F. Rischel, "Towards foundations of categorical cybernetics," *arXiv preprint arXiv:2105.06332*, 2021.
[8] B. Fong and D. I. Spivak, "Seven sketches in compositionality: An invitation to applied category theory," *arXiv preprint arXiv:1803.05316*, 2018.
[9] J. C. Baez and B. Fong, "A compositional framework for passive linear networks," *arXiv preprint arXiv:1504.05625*, 2015.
[10] D. I. Spivak and R. E. Kent, *Ologs: A categorical framework for knowledge representation*. Public Library of Science, 2012, vol. 7, no. 1.
[11] N. Ghani, J. Hedges, V. Winschel, and P. Zahn, "Compositional game theory," in *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science*, 2018, pp. 472–481.
[12] J. A. Estefan *et al.*, "Survey of model-based systems engineering (mbse) methodologies," *Incose MBSE Focus Group*, vol. 25, no. 8, pp. 1–12, 2007.
[13] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*, 3rd ed. Morgan Kaufmann, 2014.
[14] M. A. Kurfman, R. B. Stone, M. Van Wie, K. L. Wood, and K. N. Otto, "Theoretical underpinnings of functional modeling: preliminary experimental studies," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 35142. American Society of Mechanical Engineers, 2000, pp. 203–216.
[15] MathWorks. (2023) System composer documentation. Accessed: 2023-03-16. [Online]. Available: https://www.mathworks.com/help/systemcomposer/
[16] P. J. Mosterman, J. Zander, G. Hamon, and B. Denckla, "Towards computational hybrid system semantics for time-based block diagrams," *IFAC Proceedings Volumes*, vol. 42, no. 17, pp. 376–385, 2009.
[17] B. Denckla and P. J. Mosterman, "Formalizing causal block diagrams for modeling a class of hybrid dynamic systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 4193–4198.
[18] D. A. Novikov, "Cybernetics 2.0: Modern challenges and perspectives," in *Trends in Advanced Intelligent Control, Optimization and Automation: Proceedings of KKA 2017—The 19th Polish Control Conference, Kraków, Poland, June 18–21, 2017*. Springer, 2017, pp. 693–702.
[19] S. Frey, J. Hedges, J. Tan, and P. Zahn, "Composing games into complex institutions," *arXiv preprint arXiv:2108.05318*, 2021.
[20] M. Zargham and K. Nabben, "Aligning 'decentralized autonomous organization'to precedents in cybernetics," *Available at SSRN*, 2022.
[21] N. G. Leveson, *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.
[22] B. P. Douglass, "Chapter 1-what is model-based systems engineering," *Agile Systems Engineering*, pp. 1–39, 2016.
[23] Wikiversity. (2023) Introduction to category theory/sets and functions. Accessed: 2023-03-15. [Online]. Available: https://en.wikiversity.org/wiki/Introduction_to_Category_Theory/Sets_and_Functions

This version: April 5, 2023