(Project Narrative)

Structure and dynamics of working language

David I. Spivak

Abstract

Frederick A. Leve — Dynamical Systems and Control Theory
Richard D. Riecken — Science of Information, Computation, Learning, and Fusion
Tristan Nguyen — Information Assurance and Cybersecurity

(*Abstract may be publicly released*)

Language *works* in the sense of basic physics: it directs energy transfer, leading to the displacement of material objects in space. Abstractly speaking, language includes the DNA that directs biological life, natural languages like English that direct our social interactions, and the high-level programming languages that direct our technology.

In all cases, language is selected for its expressive power, communicative success, and ease of processing. We claim that category theory is one of society's most impressive instances of these features to date. It offers schemas for handling arbitrarily complex grammatical constructions in a staggering variety of application areas. As it continues to be operationalized in compositional programming libraries such as AlgebraicJulia for scientific and industrial use, category theory will play an increasing role in coordinating humanity's efforts toward solving its biggest problems.

But category theory can also be used to study the dynamics of this process itself. Robust control of material resources demands a symbol system and compositional grammar that fits with the structural and combinatorial possibilities of those material resources. The dynamics—a gears-level description of how compositional language actually directs physical activity—seems to be a compilation process, akin to how dependent type theory is compiled into intermediate languages like C, then to Assembly, then to machine code, which directs coordinated changes to the voltage-levels on a huge array of transistors. But how can we mathematically account for this process in general, not just as it pertains to computers, but abstractly so that it also makes sense for other domains, such as systems biology?

With this grant we aim to clarify the multi-scale and substrate-independent process of constructing and operationalizing languages. We will use category theory, e.g. an approach to categorical systems theory using polynomial functors, to describe the structure and dynamics of working language. We will also treat categorical constructions—to the extent that they are operationalized in the real world as working systems with material consequences—as an interesting test case of how efficient language actually accomplishes its aims.

Contents

Part I

Statement of Objectives

Statement of Objectives

The purpose of this grant is to improve our formal understanding of how language works in the real world. Soliciting support from three AFOSR programs—Dynamical Systems and Control, Science of Information, Computation, Learning, and Fusion, and Information Assurance and Cybersecurity—we aim to understand both the structure and the scale-invariant dynamics by which language operates, matures, and consistently provides a robust and high-assurance method for directing material resources.

Language is selected for its expressive power, communicative success, and ease of processing [SS18]. This is true not only of natural languages, but also of the low-level nucleotide language in DNA that codes for protein production and of the high-level programming languages that run our computers. And it is true of mathematics itself, a language which has been wildly successful in its ability to direct activity in the material world: mathematical language forms the core of the science, engineering, and technology which have transformed the face of planet Earth.

Within mathematics, category theory perhaps holds the title of being the most concise-in-expression yet broad-in-application. For example, polynomial functors simultaneously provide a remarkably compressed and workable language both for low level dynamics, e.g. as used in computational circuits [Jac17; Spi20], and for high level *type-theoretic* programming constructs that control those circuits [Awo14; AN18].

Thus a refined purpose of this grant is to use category theory, e.g. polynomial functors, to describe the dynamics of operational language at all scales. In particular, we will aim to carry out the following objectives:

1. Continue developing the theory of polynomial functors in various categories to model combinatorial assembly of interacting dynamical systems [Spi20; Smi21; NS22] that can perform functions such as learning, prediction, and control.
2. Develop the theory and applications of dynamic operads [SS22b]—which currently include deep learning and prediction markets—to model other multi-scale organizational systems.
3. Use selection categories to articulate the structure of programming language compilation, e.g. compiling a REPL into Assembly or into machine code.
4. Model the behavior and/or fitness of dynamic populations for achieving context-specific goals.
5. Use polynomial functors to model programming constructs such as dependent type theories [Awo14; AN18] and user interfaces.
6. Mathematically model the dynamics of language as a compositional symbol system that robustly directs the flow of material resources.
7. Provide a polynomial language for synthetic programming, as found in modern tactics libraries, to solve programming tasks by assigning subtasks to agents arranged according to a dependency poset [SS22a].
8. Categorically describe the behavior contracts which allow continuous dynamical systems such as transistors to compute digitally, i.e. which provide the compositionality of attractor lattices required for them to coordinate [Hor+14; Lib20].

Part II

Research effort

1 Introduction

The purpose of this grant is to improve our formal understanding of how language works in the real world. Soliciting support from three AFOSR programs—Dynamical Systems and Control, Science of Information, Computation, Learning, and Fusion, and Information Assurance and Cybersecurity—we aim to understand both the structure and the scale-invariant dynamics by which language operates, matures, and consistently provides a robust and high-assurance method for directing material resources.

To explain the purpose and the question we are asking as our guiding theme for this grant, let's begin more fundamentally.

1.1 Matter and Pattern

Matter and *pattern* are two of the most profound distinctions in the English language. Etymologically, the word *matter* comes from mother (mater) and the word *pattern* comes from father (pater).[1] Like two parents, matter and pattern represent a fundamental dichotomy: matter is the pure material, embedded in the world and unconcerned with our ideas about it; pattern is pure structure, abstracted out of the world and unconcerned with what substantiates it.

The transistor is an excellent example of where matter and pattern meet.[2]

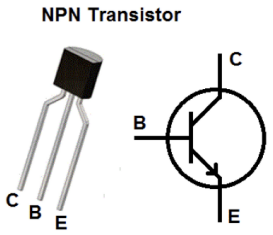


Figure 1: *A picture of an NPN transistor and its circuit diagram. Is a transistor matter or pattern?*

On the left of Fig. 1 we see the transistor as matter, a thing in the material world. On the right we see the transistor as pattern, a logical idea. There does not seem to be a perfect English word for this sort of fulcrum or janus-point which—like a transistor—is simultaneously both matter and pattern, but we will use the term *hylomorph*, a term which combines matter and form.[3]

[1] While we will sometimes refer to the mother-father roots of matter-pattern, any relation with received intuitions about gender roles is not of interest here. What's of interest is the fundamental nature of the matter-pattern dichotomy and the way that these two "parents" work in concert to produce some of the most interesting and consequential aspects of our world.

[2] This example, and my take on the philosophical question of where matter and pattern meet, arose out discussions with Scott Garrabrant, Sophie Libkind, Eliana Lorch, and Anna Salamon. The introductory section of this grant takes liberally from a blog post I wrote in November 2022.

[3] *Hylomorphism* is a term going back to Aristotle, so we should note that our matter-pattern dichotomy is related to but different than his matter-form dichotomy [Ain20]. In particular, our notion of *pattern* emphasizes structure and mathematical describability.

Mathematics is very much on the "pattern" side of the dichotomy, and it is reasonable to almost equate math and pattern: mathematics is something like the articulation of pattern. So we are equally interested in how math and matter relate.

It is now widely believed that all of math can be recorded and developed within a proof assistant like Agda, Coq, Idris, or Lean. These programs run on computers, computers are made of logic gates, logic gates are made of NAND gates, and NAND gates are formed by attaching two transistors as shown in Fig. 2.

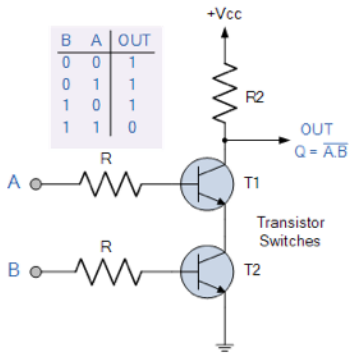


Figure 2: *A circuit diagram forming a logical NAND gate from two transistors.*

One can imagine the connection between math and matter, via proof assistants that run on transistors, as taking place in a kind of hour-glass shape.

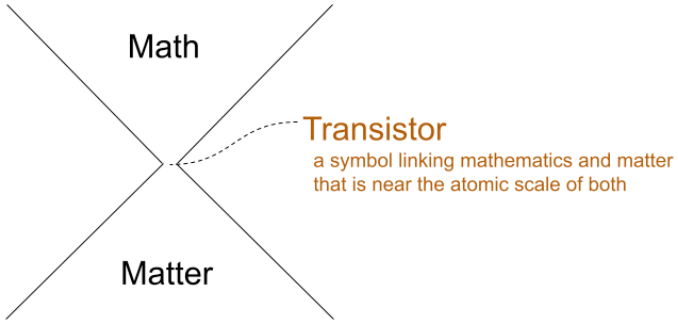


Figure 3: *The transistor serves in a dual role, as both math and matter, and happens to be close to the atomic scale in both: pattern-wise, it is half of a NAND gate (upon which all the logic necessary for a computer proof assistant is built), and matter-wise, as of late 2022, it is about 2 nanometers (the size of 10 silicon atoms) in length.*

Mathematics is purely conceptual, and yet these concepts need to turn into material action in order for the math to affect the world. For example, Claude Shannon carried out some of his most influential work in the 1940s and 50s at Bell Labs. The Bell Telephone company provided organizational infrastructure by which Shannon's purely mathematical ideas, e.g. on the speed and secrecy limits of message passing

[Sha48; Sha49], were put into practice. These ideas subsequently changed the material conditions of the world, both by helping the US and its allies win World War II and by laying a foundation for the information age that so thoroughly permeates society today.

The question that motivates this grant is: how does language, e.g. mathematics, create a such profound differences in physical reality? Next we will discuss other examples of symbols and how they compose to form working language. Then we will explain what this has to do with category theory and turn to the details of our proposed project in Chapter 2.

1.2 Examples of matter-pattern symbols

Besides transistors, there are other examples of symbols, i.e. condensed instantiations of both matter and pattern.

DNA: As an acidic molecule embedded in 3D space that acts according to physical laws, DNA is matter; but as an ordered sequence of four letters whose three-letter words code for amino acids, it is pattern. Reading the DNA and elaborating its meaning involves many other parts of the cell, and we could consider that whole complex to be both matter and pattern, but DNA is more condensed, more symbolic. Is DNA matter or pattern?

Signature: As the process of moving a pen across paper to form an ink stain, a signature is matter; but as a token of people's agreement to regulate their behavior, it is pattern. As we sign a contract in good faith, we've set up our internal state in such a way that we think it's likely that our actions in the material world will follow the pattern dictated by the contract. Is signature matter or pattern?

Grandma neuron: Neuroscientists say that a single neuron can code for a single concept, e.g. neuron X fires if and only if one recognizes their own grandma [Gro02]. As a carbon-based object, a neuron is matter; but as representing one's grandma, it is pattern. If we look at just the grandma neuron, is it matter or pattern?

There are many such examples, and what is perhaps most striking is that there appears to be some process by which examples are perpetually being created: symbols keep getting formed and condensed in our world. Probably early life's control mechanism was far less condensed than "modern" DNA [SM16]. Whatever happened in that prebiotic chemistry on the early Earth, the robust and effective language for building custom proteins seems to have been repeatedly refined over eons. Similarly, coordination of animal activity is ancient, but the condensation of this coordination into a binding contract, or the symbolic signature itself, is quite new. And a similar thing could be said about the grandma neuron or the transistor. In each case the symbol came to exist by way of some sort of natural push or process urging the formation of smaller, more concentrated symbols to instantiate pattern as matter or bind matter to pattern.

We are not interested in these examples *per se*, but rather in the fact that there appears to be a natural process by which hylomorphs are continually being formed and condensed, and the fact that this process seems to work at all scales and regardless of

substrate. But there is another important feature that we have so far mostly left out of the discussion: *compositionality*.

1.3 Compositionality and language formation

Transistors wouldn't be nearly as important in our world today if they didn't form latch circuits and NAND gates, and hence logic gates, adder circuits, CPUs, etc. But all that is purely conceptual, i.e. pattern. The thing that makes this grammar work is that it simultaneously fits the material embodiment as well. The material transistors can be arranged *according* to the conceptual pattern and there is a *congruence* or "functoriality" there:[4] just as the matter instantiates the pattern at the lowest level, so does the matter instantiate the pattern at the higher levels too. Indeed, logic gates are made of NAND gates wired together, both in pattern and in matter, and this analogy keeps holding all the way up. My computer has been programmed to let me read this document, but it is operating on physics in the material world, thanks to the robustness and compositionality of the transistor hylomorph.

Similarly, a single contract is important, but an organization works with a whole slew of contracts. When each is being decided upon, the composition of the whole—how all those agreements are going to be carried out within the one organization—requires a conceptual understanding of roles and activities, all of which will be instantiated by human bodies. Though far less formalized than the case of computers, the composition of agreements within a company reflects the composition of movements by its employees.

The same compositionality story can be told about DNA or neurons. A single nucleotide does not define DNA, nor does a single neuron define the brain. The organization of nucleotides in DNA or of neurons in the brain is extremely important for determining what the DNA or brain function will actually be. The point is that in each case the function is carried out materially in accordance with high-level patterns. This works because the embodied analogy between matter and pattern is *compositional*: the arrangement of matter respects the connection patterns in these organized systems.

So we're not only interested in the hylomorphic symbol at the center of matter and pattern, but in the compositionality—the grammar—by which the composition of conceptual patterns and the composition of material flows maintain their alignment and can thus act in concert to produce the "offspring" we see all around us, in biological life, organizations, technology, etc.

Let's temporarily and loosely define a *working language* to be a compositional symbol system that maintains alignment between matter and pattern. Language occurs materially and affects the material world, and it is also patterned in that it follows conceptual grammatical rules. When I say "pass the salt", 10^{20} atoms including your arm and a salt shaker move through space, resulting in some sodium chloride crystals arriving in my bowl of soup. It is as though language transmits momentum: it makes objects move.

[4] *Functoriality* is a category-theoretic term-of-art indicating a structure-preserving relationship between two different domains.

Language controls so much of our world, especially if we regard things like DNA as language.

Thus the drive toward finding hylomorphic symbols that compositionally bridge matter & pattern seems to be the same as, or at least tightly linked with, language formation. So it is interesting to ask: by what process is language—including the language of thought, the language of computation, the language of life, etc—formed? This formation appears to be a natural multi-scale process, beginning at least as far back as DNA, and continuing to this day.

Human natural language is selected for its expressive power, communicative success, and ease of cognitive processing. It has been put forth that, in this selection process, the replicating units are "schemas for handling grammatical constructions" [SS18]. But the same is true for other languages too, e.g. mathematical languages and programming languages; for example, Dijkstra points out in a famous paper [Dij68] that the GOTO statement should be considered harmful because its use makes cognitive processing difficult. We can postulate that even in the case of DNA, the ability for the genome to express higher-level traits (not just one protein, but whole suites of interacting proteins), such that the translation process is reliable and efficient, is selected for. Hence at many scales and in many disparate substrates, working languages are formed according to analogous criteria.

1.4 Finding the right abstractions

At this point, we may say that we have some *philosophical* explanation of how language works: it offers a symbol system that tightly links matter and pattern, and does so in a compositional way. What we aim for in this grant is a *formal, mathematical* explanation of how language works, how it leads to such far-reaching yet finely-tuned displacements of material objects in space. At the gears-level, what are the moment-by-moment dynamics by which this multi-scale process unfolds? Moreover, within what sorts of "mathematical worlds" would languages—compositional symbol systems—repeatedly come into being?

A fairly good lay-person description of category theory is that it is the theory of "schemas for handling grammatical constructions". Indeed, just as predicted in [SS18], these grammatical schemas are what replicate across mathematical and scientific domains, and hence what (applied) category theory studies. Category theory is thus an appropriate formalism for approaching our question.

However to do so requires that we find the right category-theoretic abstractions with which to consider our question. The PI has been guided by—and looking for good category-theoretic abstractions regarding—similar questions for many years. One important success criterion is that the abstraction should *work*: it should be operationalized in working code, or at least be transmitted to the minds of those who make material change in the world.[5]

[5]Many of the PI's previous papers on categorical databases, metric realization, wiring diagram operads, and several other subjects have been transitioned to industry by a variety of technical companies. For

One may notice that in fact the above criterion—that our category-theoretic language should lead to material change in the world—is in some sense a "fixed point" of our entire proposal. The mathematics itself, at least to the extent it actually works, is an example of *working language*, our object of study. We do not need to get too meta, circular, or "loopy" [Hof07] about this. The reason we mention it is merely that some of our proposed work in this grant application is aimed toward inventing powerful and operational languages, e.g. categorical descriptions of dependent type theories based on polynomial functors. Even if such work does not explain the *dynamics* of working language, it is still relevant in that it instantiates the *structure* of working language.

In the rest of this grant proposal, we will explain various category-theoretic considerations related to our central question, that which has been described above.

2 Categorical structure and dynamics of working language

The question of how compositional symbol systems work—the dynamics by which grammatical structuring of hylomorphic symbols creates such profound effect in our world, as well as by which such grammatical structures seem to perpetually arise and mature as if by an invisible hand—is deep and worthy of devoted contemplation. This question and the considerations of Chapter 1 constitute a guiding theme for this grant.

However, in order to clarify the subject matter and make observable progress, it is valuable to approach the question from a number of distinct directions, to articulate specific examples, and to formalize it in a variety of ways so as to find common themes and bridges between these various approaches.

In the remaining sections, we offer some approaches that seem promising now, at the outset of this endeavor.

2.1 Polynomial functors

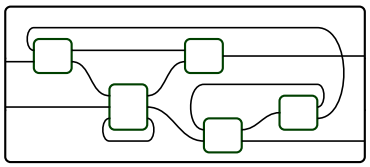
Polynomial functors are a beloved formalism within category theory[6] because of their ubiquity and excellent formal properties. For example, the category **Poly** of polynomial functors has four very useful monoidal products, as well as closures, coclosures, limits, colimits, etc. **Poly** is a cornucopia of mathematical surprises; for example it is striking that the most fundamental object in category theory—categories—are exactly the comonoids in **Poly**.

With all this abundance, polynomial functors provide a foundation upon which one can model a wide variety of subject areas, such as

example, the paper introducing the state-of-the-art dimension-reduction algorithm UMAP credits the PI's work: "We seek to address the issue of uniform data distributions on manifolds through a combination of Riemannian geometry and the work of David Spivak [52] in category theoretic approaches to geometric realization of fuzzy simplicial sets" [MHM18]. Many more examples of transitions are available upon request.

[6] Indeed, in the past two years Topos Institute has held two workshops dedicated entirely to polynomial functors, whose speakers include a laundry list of leading category theorists, such as André Joyal, Ross Street, Richard Garner, Steve Awodey, and many others.

1. Open dynamical systems on a given "time" object, be it discrete, continuous, or branching;[7]
2. Wiring diagrams



   by which open dynamical systems can be composed [Spi20];
3. Cellular automata, e.g. as found in Conway's Game of Life;[8]
4. Deep learning, Hebbian learning, prediction markets, and strategic games [Spi21b; SS22b];[9]
5. Computational effects;[10]
6. Models of dependent type theory [Awo14; AN18];[11]
7. Tactics libraries for proof assistants; and
8. Database schemas, instances, migration functors, and aggregation [Spi21a].

Some of these have been well-articulated, by the PI and others, and others are in need of further development. We will devote some time to developing the theory of polynomial functors, as well as finding new applications of them.

In the following sections, we will explain various proposed extensions of these formalisms and others, which will be useful for exploring the categorical structure and dynamics of working language. Again, some of these themes are more directly related to the problem of understanding how language works, whereas others are simply good examples of language at work.

2.2 Dynamic organizational systems

In recent work [SS22b], the PI and collaborators have developed a category-theoretic framework called *dynamic organizational systems*, e.g. dynamic operads, dynamic categories, dynamic monoidal categories, etc., which is relevant to our exploration. The framework currently has only four examples: deep learning, prediction markets, non-cooperative strategic games, and Hebbian learning.

In each case, there is a multi-scale system for updating the ways that parts interact to form wholes. For example to train a deep learning system, one creates an architecture of artificial neurons. The wiring pattern itself, i.e. the connectivity of these neurons, is an open dynamical system: it changes based on its current state and the values of the

[7] Such a dynamical system is modeled by a pair (C, φ), where C is a polynomial comonoid and $\varphi\colon C \nrightarrow T$ is a cofunctor to the time comonoid T. For example, for discrete time we use $T := y^{\mathbb{N}}$, for continuous time we use $T := y^{\mathbb{R}}$, and for branching time we use some sort of cofree comonoid, e.g. for binary-branching we use the cofree comonoid on y^2.

[8] For details, see these slides.

[9] For details, see this blog post on Hebbian learning and this blog post on strategic games.

[10] For details see this blog post on effect handlers.

[11] For other approaches to dependent type theory using polynomial functors, see https://topos.site/blog/2022/09/nates-adjoint-5-tuple/.

training data and loss function as they propagate through the system. The gradient descent / back propagation algorithm satisfies the definition a dynamic operad because it is compositional: the composite of gradient descenders is again a gradient descender. The same sort of description applies to prediction markets: you can form a predictor as the composite of subordinate predictors, and the way that wealth is updated based on correct or incorrect predictions is compositional. Again, this is described formally in [SS22b].

The relevance of dynamic organizational systems to our proposed research project is two-fold. First, anything with the structure and properties of a dynamic organizational system will exemplify the main idea of our project. Namely, it will consist of a collection of machines whose interaction pattern changes as they communicate with each other. If we think of the interaction pattern—what's connected to what—as the material conditions, then we are witnessing how the language flowing between individual entities causes change to those material conditions.

Second, we will search for a particular dynamic organizational system within which we can define a more specific notion of *language* that flows between the machines, namely one that is refined and condensed through the interaction process itself. Polynomial coalgebras generalize finite state automata, which are well-known to model regular languages. Thus there is good reason to suspect that self-rearranging systems of automata may be capable of this sort of language refinement. So if we can find a sense in which this language-formation-and-refinement process is sufficiently scale-invariant, i.e. compositional, then it would fit into the framework of dynamic organizational systems.

2.3 Effects handlers

In imperative programming, each command has an *effect* on the state of the computer. But effectful programming is difficult to reason about, because effects can happen anywhere at any time, without much in the way of organizational structure. Hence in functional programming—the sort of programming that has the best-developed category-theoretic semantics—effects are considered "impure" and are handled by monads [Mog91]. But in real-world applications, a large number of these monads need to be stacked on top of each other, and the resulting programming constructs are again awkward and tangled.

In order for category theory to help organize the complexity that arises when effectful programs are combined in these often complex ways, it may be useful to put effect-handling at the center of our consideration, rather than at the periphery as though it were a special or unusual case. Polynomial functors are quite versatile when it comes to computational paradigms, such as Moore or Mealy machines [NS22], databases [Spi21a], and dependent types [Awo14; AN18]. Thus it is not much of a surprise that effects-handling also has a special form within the world of **Poly**.[12] With polynomial effects handlers, a category c can "run" programs on various copies of category d, or

[12]Indeed, given any pair of categories $\mathcal{C}, \mathcal{D}$, regarded as polynomial comonoids $(c, \epsilon, \delta), (d, \epsilon, \delta)$, we

on multiple categories $d_1, \ldots, d_k$, and aggregate the results. In particular, when these categories c, d_i are cofree, this framework is easily interpreted in the language of effect handlers. The bicomodule point of view nicely models effect handlers and provides a compositional theory by which to manipulate and combine them.

Effect handlers allow higher-level programming constructs to manipulate low-level machine code. For example, we can construct an assembly language connecting a REPL to a register machine in this way. Thus, even though this does not explain how these languages are created or refined, the polynomial theory of effect handlers does offer an exemplar of our research topic: categorically-structured working language.

2.4 Fitness and selection

It seems that the very same abstraction used in Section 2.3 can be used to model natural selection. Indeed, a morphism $m \triangleleft d \xrightarrow{\sigma} c \triangleleft m$ satisfying the comonad laws can be identified with a cofunctor

$$\begin{bmatrix} m \\ m \triangleleft d \end{bmatrix} \overset{\sigma}{\nrightarrow} c$$

and the domain $\begin{bmatrix} m \\ m \triangleleft d \end{bmatrix}$ of this map is a selection category.

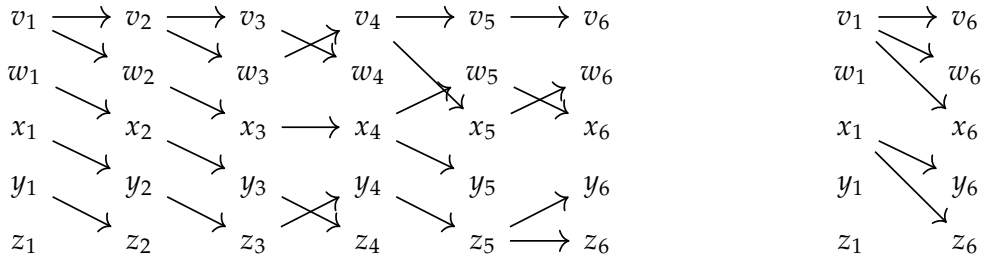


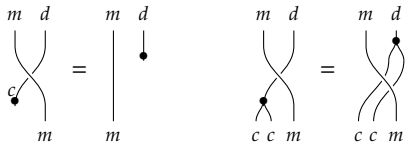
Figure 4: *A string of five composable morphisms in the selection category* $\begin{bmatrix} y^5 \\ y^5 \triangleleft \mathcal{D} \end{bmatrix}$ *(left), and their composite (right), obtained by simply following the paths. Each of the arrows shown represents a map in the base category* $\mathcal{D}$*. One sees that* v_1 *and* x_1 *were the most "fit", in that they had progeny lasting until the sixth epoch, whereas the others* (w_1, y_1, z_1) *all went extinct.*

---

can define an *effect handler from d to c* to be a polynomial functor m : **Poly**, equipped with a map

$$m \triangleleft d \xrightarrow{\sigma} c \triangleleft m$$

satisfying the following comonad laws



relating the counit ϵ and comultiplication δ of c and d. This structure gives rise to a bicomodule $c \xleftarrow{m \triangleleft d} \triangleleft\, d$.

The cofunctor σ can encode selection criteria, e.g. a fitness function by which the epochs are chosen. Moreover, preliminary work suggests that memory and environmental context can be encoded as state, captured by the polynomial m's position set $m(1)$,[13] and this state can be used to decide the current fitness of a genotype.

Whereas it is uncommon to think of natural selection as compositional or as at all related to language, the isomorphism between the formalism here and that in Section 2.3 shows that there can be a hierarchical component to selection. Rather than imagining nature as a monolithic entity that selects, we can encode various levels of selection, e.g. societies selecting cultures that select tribes that select individuals that select habitual behavior patterns, etc. In this way, smaller-scale units are seen as being like "machines" that carry out the work of some larger-scale unit through a language which we can formalize as an effect handler.

2.5 Dependent type theory

In a ground-breaking paper, Martin-Löf showed that all of (constructive) mathematics can be formalized within the language of dependent types, which also serves as a kind of programming language [Mar75]. In [Awo14; AN18], Steve Awodey and later his PhD student Clive Newstead, showed that the syntax and rules of dependent type theories are naturally modeled by polynomial monads and their pseudo-algebras. That is, thinking of a polynomial

$$p = \sum_{i:I} y^{X_i} \tag{1}$$

as being a family I of types, each of which has terms X_i, the various operations on polynomials—sum, product, Dirichlet product, and substitution product—all have meanings in terms of type families. For example, a cartesian monad structure

$$y \xrightarrow{1} p \qquad \text{and} \qquad (p \triangleleft p) \xrightarrow{\Sigma} p$$

endows the type family p with dependent sum types. Awodey and Newstead explained how dependent products are obtained using a pseudo-algebra for the monad $(1, \Sigma)$.

This inspired the PI to find an alternative approach, where the pseudo-algebra is replaced by a distributive law. We have not formally proven this result, so doing so would make a fairly straightforward task for this grant. To make it work one needs to move to categorical polynomials in the sense of [Web07]. There, the polynomial **set** has a Cartesian monad structure, and there is a distributive law

$$\mathbf{set}^{\mathrm{op}} \triangleleft \mathbf{set} \longrightarrow \mathbf{set} \triangleleft \mathbf{set}^{\mathrm{op}}$$

of **set** over its opposite. We propose that this is a good definition of *universe* for dependent type theory, and propose to show that other universes exist (e.g. **cat**).

It would also be good to combine or compare this with other work on polynomials in dependent type theory, e.g. this approach based on ideas of Nate Soares and Jesse

[13] In Fig. 4, the polynomial $m = y^5$ has only $m(1) = 1^5 = 1$ position, but more generally the position-set $m(1)$ encodes the state set of the system.

Liptrap. Regardless of approach, dependent type theory offers perhaps the most concise language for programming available today, and modeling it with polynomial functors—which also model the low-level operations of Turing machines—may enable us to more clearly see a full-stack example of language at work.

2.6 User interfaces

The common sort of user interface for a computer program consists of two parts: some sort of output display—which might include visual, auditory, or haptic output—and some menu of (a possibly huge number of) options from which the user selects an input. For example, in a Windows program, one sees the screen display and can choose from various menu options. In different displays, there may be different menu options available. When the user is at a command line, we could consider the output to be the message just displayed, and the input to be anything that can be typed into the command line. Sometimes that is an arbitrary string of characters, whereas other times it must be "yes" or "no".

Thus a user interface can be considered as a polynomial functor $p = \sum_{i \in I} y^{X_i}$, whose position-set $I = p(1)$ is that of all possible outputs (displays, sounds, etc.), and for each position $i : I$, the set X_i of directions is the menu of all available options. The user themselves can be modeled as having interface $[p, y]$, the internal hom to y.

It can be useful for one user interface to automatically control a collection of k-many other user interfaces, and this is accomplished using maps of polynomials, e.g.

$$p_1 \otimes \cdots \otimes p_k \to q$$

Using polynomial functors to design better and more compositional user interfaces appears to be a rich area for exploration.

2.7 Synthetic programming and factored cognition using dependence categories

In *synthetic programming*, one presents the machine with a type and asks for a term of it. For example, “give me a function that turns strings into integers" is presenting the machine with a type, namely `String` $\to$ `Int`, and a term of it would be some actual such function f. For example, f may be `length`, e.g. f(“hello”) = 5.

Often, the process of finding a term with some specified type can be broken up and farmed out to a collection of other synthetic programs. For example, to find a term of type $A \to (B \times C)$, one needs to find two simpler terms: one of type $A \to B$ and one of type $A \to C$; the results will then be paired. This approach to synthetic programming is also behind *factored cognition*, where a higher-level reasoning task is broken up into simpler reasoning tasks, which are farmed out and solved independently, after which the solutions are combined to form a solution to the higher-level task. It turns out that this workflow has a very nice description in **Poly**.

Just as in Section 2.5, a polynomial $p = \sum_{i:I} y^{X_i}$ consists of a collection I of types and for each one a set X_i of terms. One can think of p as a collection of synthetic

programming problems. A solution to these problems is a map of polynomials $p \to y$, since such a map comprises precisely a term $x_i : X_i$ for each type $i : I$.

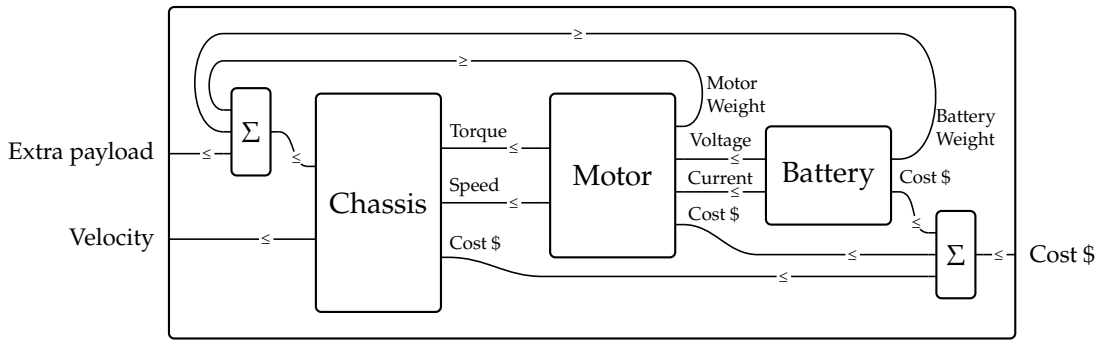
A map $\varphi\colon p \to p_1 \otimes p_2$ would constitute a way to take any problem (type) in p and produce a problem in each of p_1 and p_2, together with a way to take any pair of solutions, given by p_1 and p_2, and return a solution for p. The formula for all of this is precisely what defines the map φ. One can then nest this arbitrarily, e.g. $p_1 \to p_{11} \otimes p_{12} \otimes p_{13}$ can be a way to farm out p_1's problems to its own subordinates. This may provide a new formalization of Google's Map-Reduce algorithm [DG08].

Moreover, one can do not only parallel composition, but also serial composition. Once a cohort $p_1, \ldots, p_k$ of synthetic sub-programmers has completed their tasks, the results may be sent to a next synthetic sub-programmer p', whose solution would then be sent to the master synthetic programmer p. Such a scenario is captured by a map $p \to (p_1 \otimes \cdots \otimes p_k) \triangleleft p'$, and it should be clear that this scenario was entirely arbitrary and that many such combinations are possible. Indeed, for any partially ordered set of dependencies [SS22a], one obtains a synthetic programming setup.

We aim to explore this idea further as part of the research effort on this grant. To see the relevance, note that each step in the above process could be understood as a communication protocol—a convenient language for conveying a problem and aggregating the solutions—among synthetic programmers (or factoring-cognizers) at different levels.

2.8 Co-design of cyberphysical systems

Co-design [Cen15] is a mathematical theory for collaborative design, e.g. collaboratively designing a robot



The idea is that each box represents a *design problem*, a way to transform resources as part of a larger economically-designed system. On the box's left are the resources provided by the box, and on its right are the resources required by the box. The arrows between boxes indicate the economy by which the resources required by one box are provided by another.

Mathematically, a design problem (box) represents a boolean profunctor $P \nrightarrow R$, i.e. a monotone map $P^{\mathrm{op}} \times R \to \{\bot, \top\}$, where P is the poset of provided resources and R is the poset of required resources. This profunctor indicates a *feasibility* relationship: a true/false value for every pair (p, r), saying whether p is feasibly provided given r.

It's a profunctor because if (p, r) is feasible then receiving a better $r' \geq r$ or providing a less-good $p' \leq p$ will still render the situation feasible.

There is a natural way to use co-design to study interconnected dynamical systems. Here, the resources are much more general than "current" or "torque", but instead are dynamical system behaviors, which we can imagine as behavioral flow charts. A resource is a system that satisfies a certain behavior contract [SSV16; BVF21], i.e. promises to follow the given flow chart. Such a behavior might be like that of a vending machine, which takes money and selections and returns snacks, or a company that is satisfying some sort of contractual obligation. The behavior contracts are modeled by subterminal coalgebras—which form a logical system called a *bi-Heyting algebra*—and we showed in this blog post that when subsystems are wired together, even if that wiring diagram changes in time, there is a feasibility relationship between the behaviors of the inner systems and that of the outer system.

It would be worthwhile to formally write out the proofs, as well as to work out examples, possibly with practitioners. This formalism offers a language for combining precisely-written legal contracts, ensuring that if the subordinate entities satisfy their contract, then our dynamic arrangement for shuttling outputs and inputs is sufficient to guarantee that the subsuming entity will also satisfy its contract.

2.9 Compositionality of attractor lattices

A continuous dynamical system, such as a vector field, has a lattice of attractors ordered by containment. As the system evolves, it converges to a particular attractor, independent of noise in the system. Hence attractors structure the observed dynamics of the system and have been used to study how a continuous system is the "wetware" in which a computation is implemented [Man+13; SB13; DSS22]. Attractors include fixed points, limit cycles, and other invariant subsets; for example, the whole state space of the system counts as an attractor.

In open continuous dynamical systems, the current input acts as a parameter that controls the dynamics. For example, in the system of parameterized ODEs

$$\dot{x} = f(x, u) \qquad x : \mathbb{R}^n$$

the parameter, in this case u, controls the vector field. In a fixed vector field, the system once in the basin of attraction of a particular attractor remains there forever. However, in a parameterized vector field, modulating the system's parameters can move the system from one attractor to another. In the context of computation, this movement between attractors implements a discrete shift in the pattern domain, representing one step in an algorithmic process.

In open systems, we consider not only inputs but also outputs: some function of system X's state can be output and sent to system Y as input. The evolution of system X therefore modulates the parameters used by system Y and in doing so moves system Y between its different attractors. At the level of computation, composing such systems would be completely unreliable if the input to Y depended on the precise state $x : X$ or

if the evolution of X's attractors caused rapid, uncontrolled bifurcations in Y. Instead the attractor that system Y is in should rely only on the attractor that system X is in. It may take an infinitely long time for a state to reach some asymptote, but it is in that asymptote's basin of attraction the entire time. Thus system X communicating to system Y information about the *attractor* that X is in is more robust than communicating to Y the precise *state* that X is in.

It appears then that the compositionality of continuous dynamical systems works best when there are "spec sheets" that coordinate the attractor lattices of the various interconnected systems. A spec sheet constitutes a simple sort of behavioral contract for a system—what it expects as input and what it guarantees as output—that aligns with its attractors. It can be shown that the computations implemented by two systems only compose as expected if the systems are aligned by such a spec sheet. Therefore, the spec sheet is a key ingredient in a compositional theory for continuous systems that compute. We propose that spec sheets define a language that coordinates the high-level operations between continuous systems.

In "artificial" systems—ones where human thinking dictates the connections between things—engineers are tasked with ensuring that the spec sheets of interconnected systems align, or more precisely, avoid the ambiguities that lead to uncontrolled bifurcations. We think of this as a language formation and refinement process, as it allows different subsystems to communicate more effectively.

One goal of this project is to get a more formal understanding of that relationship and its compositional properties as more complex systems are formed from simpler ones. But this spec-sheet development and alignment process also exists in natural systems: for one thing, humans and human processes can also be viewed as natural, subject to the same laws as the rest of nature; but more generally, non-human processes have repeatedly produced interaction patterns that avoid such ambiguities, e.g. in the coding of amino acids from nucleotides [SM16]. We will consider high-level descriptions of how such natural language formation and refinement processes can arise.

2.10 Summary

Language works—grammatical constructs carry pattern into material existence—at all scales, from DNA to multinational organizations, and languages are continually refined by natural and human-led processes. Finding the right abstractions by which to study this process is crucial for seriously addressing today's biggest challenges, enhancing our capacity for accountability and enabling new engineering capabilities.

The above sections outline a variety of approaches to this study. Some take a deep look at the dynamics of working language, others investigate the way that working languages are created and refined. Still others consider existing exemplary working languages or suggest new languages with clear category-theoretic descriptions.

Category theory is perhaps humanity's best thought-compression language. As such it is both an excellent formalism in which to articulate implementable descriptions of our subject—how languages *work* in general—as well as a particularly interesting

model upon which we can focus our study while simultaneously producing useful mathematical and computational artifacts.

References

[Ain20] Thomas Ainsworth. "Form vs. Matter". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2020. Metaphysics Research Lab, Stanford University, 2020 (cit. on p. 7).

[AN18] Steve Awodey and Clive Newstead. "Polynomial pseudomonads and dependent type theory". In: *arXiv* (2018). eprint: 1802.00997 (cit. on pp. 5, 13, 14, 16).

[Awo14] Steve Awodey. "Natural models of homotopy type theory". In: (2014). eprint: arXiv:1406.3219 (cit. on pp. 5, 13, 14, 16).

[BVF21] Georgios Bakirtzis, Christina Vasilakopoulou, and Cody H Fleming. "Compositional cyber-physical systems modeling". In: *Electronic Notes in Theoretical Computer Science* (2021) (cit. on p. 19).

[Cen15] Andrea Censi. "A mathematical theory of co-design". In: (2015). eprint: arXiv:1512.08055 (cit. on p. 18).

[DG08] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on p. 18).

[Dij68] Edsger W Dijkstra. "Letters to the editor: go to statement considered harmful". In: *Communications of the ACM* 11.3 (1968), pp. 147–148 (cit. on p. 11).

[DSS22] Laura Driscoll, Krishna Shenoy, and David Sussillo. "Flexible multitask computation in recurrent networks utilizes shared dynamical motifs". en. In: (Aug. 2022) (cit. on p. 19).

[Gro02] Charles G Gross. "Genealogy of the "grandmother cell"". In: *The Neuroscientist* 8.5 (2002), pp. 512–518 (cit. on p. 9).

[Hof07] Douglas R Hofstadter. *I am a strange loop*. Basic books, 2007 (cit. on p. 12).

[Hor+14] Clare Horsman, Susan Stepney, Rob C Wagner, and Viv Kendon. "When does a physical system compute?" In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 470.2169 (2014), p. 20140182 (cit. on p. 5).

[Jac17] Bart Jacobs. *Introduction to Coalgebra*. Vol. 59. Cambridge University Press, 2017 (cit. on p. 5).

[Lib20] Sophie Libkind. "An Algebra of Resource Sharing Machines". In: *arXiv* (2020). eprint: 2007.14442 (cit. on p. 5).

[Man+13] Valerio Mante, David Sussillo, Krishna V. Shenoy, and William T. Newsome. "Context-dependent computation by recurrent dynamics in prefrontal cortex". eng. In: *Nature* 503.7474 (Nov. 2013), pp. 78–84 (cit. on p. 19).

[Mar75] Per Martin-Löf. "An intuitionistic theory of types: Predicative part". In: *Studies in Logic and the Foundations of Mathematics*. Vol. 80. Elsevier, 1975, pp. 73–118 (cit. on p. 16).

[MHM18] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018) (cit. on p. 12).

[Mog91] Eugenio Moggi. "Notions of computation and monads". In: *Information and computation* 93.1 (1991), pp. 55–92 (cit. on p. 14).

[NS22] Nelson Niu and David I. Spivak. *Polynomial functors: a general theory of interaction*. *In preparation*. 2022 (cit. on pp. 5, 14).

[SB13] David Sussillo and Omri Barak. "Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks". en. In: *Neural Computation* 25.3 (Mar. 2013), pp. 626–649 (cit. on p. 19).

[Sha48] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423 (cit. on p. 9).

[Sha49] Claude E Shannon. "Communication theory of secrecy systems". In: *The Bell system technical journal* 28.4 (1949), pp. 656–715 (cit. on p. 9).

[SM16] Eric Smith and Harold J Morowitz. *The origin and nature of life on earth: the emergence of the fourth geosphere*. Cambridge University Press, 2016 (cit. on pp. 9, 20).

[Smi21] Toby St Clere Smithe. "Polynomial Life: the Structure of Adaptive Systems". In: *Proceedings of the 4th Annual Conference on Applied Category Theory*. ACT. Cambridge, UK: EPTCS, 2021 (cit. on p. 5).

[Spi20] David I. Spivak. "Poly: An abundant categorical setting for mode-dependent dynamics". In: *arXiv* (2020). eprint: 2005.01894 (cit. on pp. 5, 13).

[Spi21a] David I. Spivak. *Functorial aggregation*. 2021. arXiv: 2111.10968 [math.CT] (cit. on pp. 13, 14).

[Spi21b] David I. Spivak. "Learners' languages". In: *Proceedings of the 4th Annual Conference on Applied Category Theory*. ACT. Cambridge, UK: EPTCS, 2021 (cit. on p. 13).

[SS18] Luc Steels and Eörs Szathmáry. "The evolutionary dynamics of language". In: *Biosystems* 164 (2018). Code Biology, pp. 128–137. ISSN: 0303-2647 (cit. on pp. 5, 11).

[SS22a] Brandon T. Shapiro and David I. Spivak. "Duoidal Structures for Compositional Dependence". In: *arXiv* (2022). eprint: 2210.01962 (cit. on pp. 5, 18).

[SS22b] Brandon T. Shapiro and David I. Spivak. "Dynamic categories, dynamic operads: From deep learning to prediction markets". In: *Electronic Proceedings in Theoretical Computer Science* (2022) (cit. on pp. 5, 13, 14).

[SSV16] Patrick Schultz, David I Spivak, and Christina Vasilakopoulou. "Dynamical systems and sheaves". In: *Applied Categorical Structures* (2016), pp. 1–57 (cit. on p. 19).

[Web07] Mark Weber. “Familial 2-Functors and Parametric Right Adjoints”. In: *Theory and Applications of Categories* 18 (2007), Paper No. 22, 665–732 (cit. on p. 16).

Assurances

A Assurances

A.1 Environmental impacts

This research is purely mathematical, and thus it will have no environmental impacts; compliance with environmental statutes and regulations is thereby assured.

A.2 Principle investigator (PI) time

The principle investigator plans to spend 58.33% of his effort each year of the grant.

Current Projects and Pending Proposals The PI has a current AFOSR research grant, FA9550-20-1-0348, which ends 2023/09/14. We aim for the project in the present proposal to begin 2023/09/15.

The PI is also working on two DARPA projects, called ASKEM and PTG.

The PI is part of a pending proposal for a research grant paid for by Chevron.

A.3 Facilities

Topos Institute provides basic office space and has a budget for books and other resources as requested by its faculty and research staff.

A.4 Special test equipment

None.

A.5 Equipment

One laptop computer at $1600 each year, and $400 in miscellaneous expenses.

A.6 High performance computing availability

Not needed.