Thinking Recursively, Rethinking Corecursively

David Jaz Myers

June 19, 2017

Mathematical Metaphors

This talk will be about two specific mathematical metaphors, but

- what are mathematical metaphors,
- why make them,
- and how can they be misused?

Mathematical Metaphors

In this talk, we will look closely at the mathematical metaphor between

Complex Systems and **Recursive Functions**

We will see how this metaphor a lot of standard theories in science and philosophy, usually those that fall under the rubrik of “realism”. We will also find that this metaphor can lead us to some shaky philosophical positions.

Mathematical Metaphors

In this talk, we will look closely at the mathematical metaphor between

Complex Systems and **Recursive Functions**

We will see how this metaphor a lot of standard theories in science and philosophy, usually those that fall under the rubrik of “realism”. We will also find that this metaphor can lead us to some shaky philosophical positions.

Outline

What is a function

A function is a process that turns an **input** into an **output**.

$$\mathbf{F}(\mathbf{input}) = \mathbf{output}$$

If a function takes inputs of a type **Inputs** and gives outputs of a type **Outputs**, we write

$$\mathbf{F} : \mathbf{Inputs} \to \mathbf{Outputs}$$

What is a function

A function is a process that turns an **input** into an **output**.

$$\mathbf{F}(\mathbf{input}) = \mathbf{output}$$

If a function takes inputs of a type **Inputs** and gives outputs of a type **Outputs**, we write

$$\mathbf{F} : \mathbf{Inputs} \to \mathbf{Outputs}$$

For example,

$$\mathbf{F} : \mathbf{Numbers} \to \mathbf{Numbers}$$
$$\mathbf{F}(n) = 2n + 1$$

What is Recursion?

A function is **recursive** when its output on a complicated input is determined by its output on simpler inputs.

What is Recursion?

A function is **recursive** when its output on a complicated input is determined by its output on simpler inputs.

Ultimately, the output of a recursive function is determined by its *simplest* inputs.

What is Recursion?

A function is **recursive** when its output on a complicated input is determined by its output on simpler inputs.

Ultimately, the output of a recursive function is determined by its *simplest* inputs.

We call these simplest inputs **atoms**, or base cases, and the rules for building them up **constructors**.

What is Recursion?

So to define a recursive function we need

- to know how to break apart complicated inputs into simpler ones,

What is Recursion?

So to define a recursive function we need

- to know how to break apart complicated inputs into simpler ones,
- *simplest* inputs (so we eventually stop breaking things apart),

What is Recursion?

So to define a recursive function we need

- to know how to break apart complicated inputs into simpler ones,
- *simplest* inputs (so we eventually stop breaking things apart),
- to know how to put outputs together

What is Recursion?

So to define a recursive function we need

- to know how to break apart complicated inputs into simpler ones,
- *simplest* inputs (so we eventually stop breaking things apart),
- to know how to put outputs together in a way that relates to taking inputs apart!

What is Recursion?

So to define a recursive function we need

- to know how to break apart complicated inputs into simpler ones,
- *simplest* inputs (so we eventually stop breaking things apart),
- to know how to put outputs together in a way that relates to taking inputs apart!

Or, more pithily, we need:

- to know how to **analyze inputs**,
- into their **atomic components**,
- so that we can **construct** outputs.

A Lengthy Example

Let's calculate the length of a list! This is a function which takes a list as input and gives a number as output.

$$\textbf{Length} : \textbf{Lists} \to \textbf{Numbers}$$

A Lengthy Example

Let’s calculate the length of a list! This is a function which takes a list as input and gives a number as output.

$$\textbf{Length} : \textbf{Lists} \to \textbf{Numbers}$$

A list is something like:

$$[\text{first item, second item, third item} \ldots \text{last item}]$$

We can break down a list like this:

$$\textbf{A List} = [\text{first item, } \textbf{Rest of the List}]$$

or the list is **Empty**.

A Lengthy Example

Let's calculate the length of a list! This is a function which takes a list as input and gives a number as output.

$$\textbf{Length} : \textbf{Lists} \to \textbf{Numbers}$$

Numbers can be built up by counting:

$$\begin{aligned}&0 \text{ is a number, and}\\&(1 + \text{a number}) \text{ is a number.}\end{aligned}$$

This is related to taking lists apart because, secretly, numbers are like lists of tally marks:

$$4 = \Big|, \Big|, \Big|, \Big|$$

A Lengthy Example

Definition (Length of a List)

The length of a list is given by the function defined by:

$$\begin{aligned}\textbf{Length}(\textbf{Empty}) &\equiv 0 \\ \textbf{Length}([\text{first item}, \textbf{Rest of List}]) &\equiv 1 + \textbf{Length}(\textbf{Rest of List})\end{aligned}$$

A Lengthy Example

Definition (Length of a List)

The length of a list is given by the function defined by:

$$\begin{aligned} \textbf{Length}(\textbf{Empty}) &\equiv 0 \\ \textbf{Length}([\text{first item}, \textbf{Rest of List}]) &\equiv 1 + \textbf{Length}(\textbf{Rest of List}) \end{aligned}$$

This works because

- **Empty** is an atom. There are no simpler lists, so we can stop breaking things apart.
- The **Rest of the List** is simpler (i.e. smaller) than the list we started with. This means we eventually get to the **Empty** list.

Running a Recursive Program

We can run a recursive program **greedily**:

Every time we see something we don't understand, we compute it.

$$
\begin{aligned}
\textbf{Length}([1,2,3]) &= 1 + \textbf{Length}([2,3]) \\
&= 1 + (1 + \textbf{Length}([3])) \\
&= 1 + (1 + (1 + \textbf{Length}(\textbf{Empty}))) \\
&= 1 + (1 + (1 + 0)) \\
&= 1 + (1 + 1) \\
&= 1 + 2 \\
&= 3
\end{aligned}
$$

Thinking Recursively About Everything

This way of thinking should be familiar to you from popular ways of thinking about physics.

Claim

Physics is like a recursive function

$$\textbf{\textit{Physics}} : \textbf{\textit{Systems}} \to \textbf{\textit{Systems}}$$

*which recurses all the way to the **fundamental particles**, and then builds more complicated phenomena out of the way they behave.*

Thinking Recursively About Everything

Or from philosophy of language:

Claim

Meaning is like a recursive function

$$\textbf{\textit{Meaning}} : \textbf{\textit{Utterances}} \to \textbf{\textit{Meanings}}$$

which builds the meaning of, say, sentences out of the meaning of words.

Thinking Recursively About Everything

Or from sociology

Claim

A society is like a recursive function

$$\textbf{\textit{Society}} : \textbf{\textit{Societies}} \to \textbf{\textit{Societies}}$$

which is determined by the behavior of individuals which are, of course, indivisible.

Thinking Recursively About Everything

Or from economics

Claim

The economy is like a recursive function

$$\textbf{\textit{Economy}} : \textbf{\textit{Markets}} \to \textbf{\textit{Markets}}$$

which is determined by the behavior of agents who act rationally.

Analysis is Recursive

Definition

*[**Analysis**] might be defined as a process of isolating or working back to what is more fundamental by means of which something, initially taken as given, can be explained or reconstructed. – Stanford Encyclopedia*

A Philosophical Problem

In his book *The Case for Idealism*, John Foster argues that some things must have inscrutable, intrinsic properties.

Foster's argument for inscrutable intrinsic properties

Suppose that all properties of all things were *extrinsic*, that is, defined in relation to other things.

$$A\Bigg)\bigg)\big)) \qquad ((\big(\bigg(\Bigg(B$$

Now, consider a world containing two things, A and B, each defined only by their disposition to repel the other.

- Foster claims this leads to an infinite regress, and therefore a contradiction.

A Philosophical Problem

Foster's argument for inscrutable intrinsic properties (cont'd)

The back and forth must stop somewhere:

"A is the thing which . . . X"

X is the end of the line, it is not defined in relation to anything else. Therefore, it is both

- inscrutable, and
- intrinsic.

This argument rests on two (recursive) assumptions:

1. We must 'evaluate' greedily.
2. There must be a base case.

Do We Have to Make Those Assumptions?

. . . is there another way?

Corecursion

A function is **corecursive** when its output is determined by simpler *outputs*.

Corecursion

A function is **corecursive** when its output is determined by simpler *outputs*.

We call the rules for breaking apart the output **observers**.

What is Corecursion

So, to define a corecursive function we need

- to know how to observe the output of our function in simpler ways,

What is Corecursion

So, to define a corecursive function we need

- to know how to observe the output of our function in simpler ways,
- that relate to how we observe our inputs!

What is Corecursion

So, to define a corecursive function we need

- to know how to observe the output of our function in simpler ways,
- that relate to how we observe our inputs!

We can think of the observers as being experimental setups with which we will test the output of our function.

What is Corecursion

The main idea behind corecursion is:

If we know how our function behaves in all experimental setups, we know what it does.

What is Corecursion

The main idea behind corecursion is:

If we know how our function behaves in all experimental setups, we know what it does.

This is essentially the same as one of the fundamental principles of science:

If we can predict how something behaves in all experimental setups, then we know what it is.

What is Corecursion

The main idea behind corecursion is:

If we know how our function behaves in all experimental setups, we know what it does.

This is essentially the same as one of the fundamental principles of science:

If we can predict how something behaves in all experimental setups, then we know what it is.

So long as we believe that **a function is what it does.**

Stream and Chill

Let's have some fun with streams to get our heads around corecursion.

A stream is an infinite list, so we can't keep the whole thing in memory, but we can observe it piece by piece.

Stream and Chill

Let's have some fun with streams to get our heads around corecursion.

A stream is an infinite list, so we can't keep the whole thing in memory, but we can observe it piece by piece.

So, let's set up two experiments:

1. **Head**, where we test what the first thing in the stream is.
2. **Tail**, where we see what's left.

Now we can define functions corecursively, since we know how to observe their behavior.

Stream and Chill

Let's define a function

$$\textbf{Every Other} : \textbf{Streams} \to \textbf{Streams}$$

that will take a stream and return the stream of only every other value. For example:

$$\textbf{Every Other}(0, 1, 2, 3, 4, \ldots) = (0, 2, 4, \ldots)$$

Stream and Chill

Let's define a function

$$\textbf{Every Other} : \textbf{Streams} \to \textbf{Streams}$$

that will take a stream and return the stream of only every other value. For example:

$$\textbf{Every Other}(0, 1, 2, 3, 4, \ldots) = (0, 2, 4, \ldots)$$

To define this, we just need to define how it looks in all the experiments.

Stream and Chill

Definition (The Every Other Function)

Define the **Every Other** function by

$$\begin{aligned} \mathbf{EO}(\text{stream}).\mathbf{Head} &= \text{stream}.\mathbf{Head} \\ \mathbf{EO}(\text{stream}).\mathbf{Tail} &= \mathbf{EO}(\text{stream}.\mathbf{Tail}.\mathbf{Tail}) \end{aligned}$$

Stream and Chill

Definition (The Every Other Function)

Define the **Every Other** function by

$$\begin{aligned} \mathbf{EO}(\text{stream}).\mathbf{Head} &= \text{stream}.\mathbf{Head} \\ \mathbf{EO}(\text{stream}).\mathbf{Tail} &= \mathbf{EO}(\text{stream}.\mathbf{Tail}.\mathbf{Tail}) \end{aligned}$$

This works because

- **EO**(stream) is **covered** by the observers **Head** and **Tail**, they tell us all we need to know about it.

Running a Corecursive Program

We can't evaluate a corecursive program greedily, because the calculation would never end! We have to be **lazy**:

Only compute things when we absolutely need to.

So if you wrote down

$$\mathbf{EO}((0, 1, 2, 3, ...))$$

That would be totally chill.

Running a Corecursive Program

But, if we want to know a specific value of $\mathbf{EO}((0,1,2,3,\dots))$, then we can calculate

$$
\begin{aligned}
&\mathbf{EO}((0,1,2,3,\dots)).\mathbf{Tail}.\mathbf{Tail}.\mathbf{Head}\\
&= \mathbf{EO}((0,1,2,3,\dots).\mathbf{Tail}.\mathbf{Tail}).\mathbf{Tail}.\mathbf{Head}\\
&= \mathbf{EO}((0,1,2,3,\dots).\mathbf{Tail}.\mathbf{Tail}.\mathbf{Tail}.\mathbf{Tail}).\mathbf{Head}\\
&= (0,1,2,3,\dots).\mathbf{Tail}.\mathbf{Tail}.\mathbf{Tail}.\mathbf{Tail}.\mathbf{Head}\\
&= (1,2,3,4,\dots).\mathbf{Tail}.\mathbf{Tail}.\mathbf{Tail}.\mathbf{Head}\\
&= (2,3,4,5,\dots).\mathbf{Tail}.\mathbf{Tail}.\mathbf{Head}\\
&= (3,4,5,6\dots).\mathbf{Tail}.\mathbf{Head}\\
&= (4,5,6,7\dots).\mathbf{Head}\\
&= 4
\end{aligned}
$$

Corecursion and Différance

If someone asks you what “**EO**” *means*, you could tell them that its meaning is **deferred** until we test it with the observers **Head** and **Tail**.

Corecursion and Différance

If someone asks you what “**EO**” *means*, you could tell them that its meaning is **deferred** until we test it with the observers **Head** and **Tail**.

If they ask you what “**Head**” and “**Tail**” mean, you could only tell them the **different** ways you end up using them.

Definition

Différance is Derrida’s pun on the words *defer* and *differ*.

Thinking Corecursively

Who am I?

Thinking Corecursively

Who am I?

How can I find out?

Thinking Corecursively

Who am I?

How can I find out?

Do I have to find my ‘true self’, the core of my being, to know who I am?

Thinking Corecursively

Who am I?

How can I find out?

Do I have to find my ‘true self’, the core of my being, to know who I am?

Or do I only have to look at the way I affect the people and places around me?

Thinking Corecursively

Who am I?

How can I find out?

Do I have to find my ‘true self’, the core of my being, to know who I am?

Or do I only have to look at the way I affect the people and places around me?

Thinking corecursively, we don’t have to be anxious about finding our true selves.

Revisiting Foster

Let's look back at Foster's argument for inscrutable intrinsic properties. He claims that the world in which

$$A \text{ only repels } B \text{ and } B \text{ only repels } A$$

cannot exist because it leads to an infinite regress.

- Only leads to infinite regress if we are greedy.

Revisiting Foster

Let's look back at Foster's argument for inscrutable intrinsic properties. He claims that the world in which

A only repels B and B only repels A

cannot exist because it leads to an infinite regress.

- Only leads to infinite regress if we are greedy.
- If we are lazy, this is a perfectly fine definition.

There is nothing inscrutable about it.

Revisiting Foster

Foster's argument shows a fundamental confusion that often underlies recursive thinking:

the confusion between **names** and **things**

Revisiting Foster

Foster's argument shows a fundamental confusion that often underlies recursive thinking:

the confusion between **names** and **things**

- **Names** are like atoms, we don't break them apart.
- **Things** (such as functions) can be *named*, even when we define them corecursively.
- But that doesn't mean that they have base cases!

Limits of the Metaphor

To define a function corecursively, we must **cover** it by observers.

- **Head** and **Tail** tell us all there is to know about a stream.

Limits of the Metaphor

To define a function corecursively, we must **cover** it by observers.

- **Head** and **Tail** tell us all there is to know about a stream.

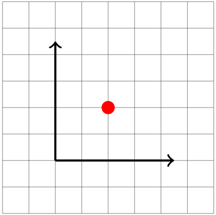
But in the informal world, we never have access to all the contexts in which an object appears,

We can never get all sides of the story.

Going Forward: Physics

Physicists have been thinking corecursively for a long time:

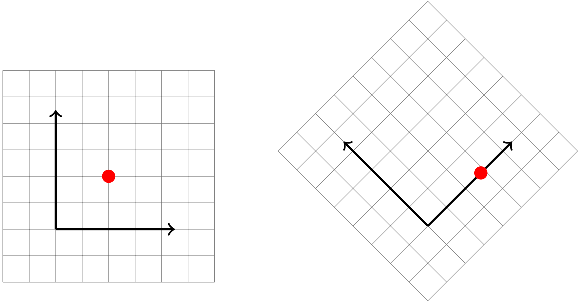
- A physical quantity can only be assigned specific values given a local coordinate system, or **gauge**.



Going Forward: Physics

Physicists have been thinking corecursively for a long time:

- A physical quantity can only be assigned specific values given a local coordinate system, or **gauge**.



Going Forward: Physics

Principle of Relativity

The physical laws have the same form in all choices of gauge.

A change in gauge is called a **gauge symmetry**.

In other words, if we rotate our experimental setup, we get a rotated result.

$$\Psi.r(X) = r(\Psi.X)$$

The relationship between the observations $\Psi.X$ and $\Psi.r(X)$ depends on *how* X was rotated to $r(X)$.

Going Forward: Physics

Principle of Relativity

The physical laws have the same form in all choices of gauge.

A change in gauge is called a **gauge symmetry**.

In other words, if we rotate our experimental setup, we get a rotated result.

$$\Psi.r(X) = r(\Psi.X)$$

The relationship between the observations $\Psi.X$ and $\Psi.r(X)$ depends on *how* X was rotated to $r(X)$.

To fully know an object, we must not only know how it behaves in various contexts,

we must also know how those contexts relate.

In Conclusion

Thinking recursively makes us believe that

- There are basic objects and basic truths about them at the bottom of all phenomena, and
- To know anything at all, we need to know about these basic things.

In Conclusion

Thinking recursively makes us believe that

- There are basic objects and basic truths about them at the bottom of all phenomena, and
- To know anything at all, we need to know about these basic things.

Thinking corercursively makes us believe that

- Things only make sense in context (in an experiment, relative to an observer, etc.), and
- Knowing how a thing behaves in context is all there is to know about it

In Conclusion

Thinking recursively makes us believe that

- There are basic objects and basic truths about them at the bottom of all phenomena, and
- To know anything at all, we need to know about these basic things.

Thinking corercursively makes us believe that

- Things only make sense in context (in an experiment, relative to an observer, etc.), and
- Knowing how a thing behaves in context is all there is to know about it

There are no basic objects or basic truths

Bridging the Divide

In this talk, I made a stark division between recursive and corecursive thinking.

Bridging the Divide

In this talk, I made a stark division between recursive and corecursive thinking.

But in actually programming languages (like Haskell), you can use recursion and corecursion together depending on which is more convenient.

Bridging the Divide

In this talk, I made a stark division between recursive and corecursive thinking.

But in actually programming languages (like Haskell), you can use recursion and corecursion together depending on which is more convenient.

We should use recursive and corecursive thinking together, depending on what needs to be done.

Bridging the Divide

In this talk, I made a stark division between recursive and corecursive thinking.

But in actually programming languages (like Haskell), you can use recursion and corecursion together depending on which is more convenient.

We should use recursive and corecursive thinking together, depending on what needs to be done.

But most importantly, we need to remember that metaphors matter.

Don’t get trapped in a single metaphor

References I

- Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer, *Copatterns: Programming infinite structures by observations*, SIGPLAN Not. **48** (2013), no. 1, 27–38.
- Michael Beaney, *Analysis*, The Stanford Encyclopedia of Philosophy (Edward N. Zalta, ed.), spring 2015 ed., 2015.
- J. Rutten. C. Kupke, M. Niqui, *Stream differential equations: concrete formats for coinductive definitions.*, Technical Report No. RR-11-10 (2011), 1 – 28.
- Barry Dainton, *Time and space: Second edition*, Mcgill-Queens University Press, 2010.
- Dexter Kozen and Alexandra Silva, *Practical Coinduction*, 2014.

References II

J. Rutten, *An introduction to (co)algebra and (co)induction.*, Advanced topics in bisimulation and coinduction. (D. Sangiorgi and J. Rutten, eds.), Cambridge University Press, Cambridge, 2011.