Mathematical Notation

Daniel Nikpayuk

2018

Abstract

Very simply put: Mathematical notation is a mathematical object worthy of its own theory.

This informal essay explores this idea by means of solving for the antiderivative of

$$\int (x \ln x)^n dx$$

using the proof as a case study to draw out the basic relevant concepts.

Motivation

The claim here is that math notation is not only used to discuss math, *it is math*.

Truth be told though, we really don't look at it this way in our daily lives. We take it for granted, first because we'd rather use it than think about it, but more importantly because math notation is quite often very elegant and well thought out to begin with. This is generally so because traditionally it is mathematicians themselves who design the notation—after having years of knowledge and experience interacting with elegant mathematical structures, constructs, and paradigms. The claim I am making then is: For as well thought out as our notation is, it is not *rigourously* thought out to the standards that mathematicians hold for every other area of our mathematical lives.

So what is the antiderivative of the following expression?

$$\int (x \ln x)^n dx$$

I'll tell you now it does in fact exist in a closed form, if a not entirely simple one.

You might start by finding the antiderivative for some initial cases in hopes you'll find a pattern [1] :

$$\begin{aligned}
\int x \ln x dx &= \tfrac{1}{2}x^2 \ln x - \int \tfrac{1}{2}x^2(\tfrac{1}{x})dx \\
&= \tfrac{1}{2}x^2 \ln x - \tfrac{1}{2}\int x dx \\
&= \tfrac{1}{2}x^2 \ln x - \tfrac{1}{4}x^2 \\
&= \tfrac{1}{2}x^2(\ln x - \tfrac{1}{2})
\end{aligned}$$

$$\begin{aligned}
\int (x \ln x)^2 dx &= \int x^2 (\ln x)^2 dx \\
&= \tfrac{1}{3}x^3(\ln x)^2 - \tfrac{2}{3}\int x^2 \ln x dx \\
&= \tfrac{1}{3}x^3(\ln x)^2 - \tfrac{2}{3}(\tfrac{1}{3}x^3 \ln x - \tfrac{1}{3}\int x^2 dx) \\
&= \tfrac{1}{3}x^3(\ln x)^2 - \tfrac{2}{9}x^3 \ln x + \tfrac{2}{27}x^3 \\
&= \tfrac{1}{3}x^3((\ln x)^2 - \tfrac{2}{3}\ln x + \tfrac{2}{9})
\end{aligned}$$

Sometimes that works—and it might even here—but let's look for a deeper understanding of design as ad-hoc luck isn't necessarily sustainable in the long run. As for strategies for a solution, one expects to use

[1] I'm ignoring the usual antiderivative constant(s), I see no harm in doing so for this essay.

integration by parts as done above, but *change of variables* seems like it might bear some fruit as well. If you play with it a little, you could change it to the following form:

$$\frac{1}{n^{n+1}} \int e^{\frac{n+1}{n}t}\, t^n dt, \qquad t = n \ln x \quad (x = e^{\frac{t}{n}})$$

which simplifies it in one perspective, but complicates it in another.

Integration by parts seems to be the main strategy after all. Looking at the original $\int (x \ln x)^n dx$, the reason for this is by taking repeated derivatives of $(\ln x)^n$ we will eventually get rid of $\ln x$ (in exchange for $\frac{1}{x}$) and then, even though we'll have accumulated some summation terms along the way, all we'll be left with in our final term is a polynomial to integrate which is something we know how to do. This approach at least guarantees a solution, so let's follow it through:

$$\begin{aligned}
\int (x \ln x)^n dx &= \int x^n (\ln x)^n dx \\
&= \tfrac{1}{n+1} x^{n+1} (\ln x)^n - \int \tfrac{1}{n+1} x^{n+1} \cdot n (\ln x)^{n-1} (\tfrac{1}{x}) dx \\
&= \tfrac{1}{n+1} x^{n+1} (\ln x)^n - \tfrac{n}{n+1} \int x^n (\ln x)^{n-1} dx
\end{aligned}$$

---

which if we let	simplifies (almost) to	but not quite, because
$a_n := \int x^n (\ln x)^n dx$	$a_n = b_n + c_n \cdot a_{n-1}$	$a_{n-1} \neq \int x^n (\ln x)^{n-1} dx$
$b_n := \frac{1}{n+1} x^{n+1} (\ln x)^n$		
$c_n := -\frac{n}{n+1}$		

Obviously we don't need to replace our equation with $a_n = b_n + c_n a_{n-1}$, we could continue with the raw "low-level" form, but it gets pretty tedious pretty quickly. It would be nice if we could translate our derived identity to a minimalist recurrence relation, but the notation as it currently exists blocks us from doing so. One could again take ad-hoc approaches to get around this, but let's not. It's time to introduce more systematic ways of thinking about our notation.

Intuition

An **interface** is a means to interacting with a context, a system, an environment, something exterior to you. If the context you want to interact with is simple enough, you interact with it directly, with your biological senses (seeing, hearing, touch, etc). If we're using an interface at all, it's because the context we're trying to interact with is too complicated (or risky) in some way to interact with directly.

From this point of view, at its simplest, an interface is a *tool*. Really though, interfaces tend to be collections of tools, and when the context is sufficiently complex, it becomes a framework of tools, a *technology*. The most powerful technology humans have are our brains. Hear me out: Our ability to simulate reality, being able to predict what will happen at a lower cost and lower risk than actually taking action allows us to manipulate the world around us. If that's not a technology, I don't know what is. The thing is, we don't communicate thoughts directly to each other, so we need a medium to pass our thought messages—a common communications infrastructure. Therefore the next most powerful technology humans have (which we can share) is language.

Language is a lot of things to a lot of people, but the way we're looking at it right now is as a technology. Since we're focused on language this way, we need a common and really well designed language of interface. As it turns out, the most powerful medium of expression for interface design is predicate logic:

$$[\ \forall x\ .\ P(x) \wedge \neg Q(x) \implies R(x)\ ] \qquad \vee \qquad [\ A \wedge \exists y\ .\ B(y)\ ]$$

The thing to keep in mind though is that symbolic logic as shown above is a *weak specification*. It becomes a strong specification when we actually interpret what our predicates $P(\cdot), Q(\cdot), R(\cdot), A, B(\cdot)$ are allowed to

talk about (currently they're just uppercase letters of the latin alphabet). For example if we choose our context and discourse to be the real numbers, we might have

$$P(x) \quad := \quad \text{“}x \text{ is compact”}$$

To summarize this idea: **An interface is a predicate logic equipped with a space.**

When you enter a new terrain, and you're looking to interface with it, the safest bet is to pack everything tool-wise (including the kitchen sink). Keeping things intentionally vague here, let's say for example our interface language is the natural numbers:

$$\{0, 1, 2, 3, 4, 5, 6, 7, \ldots\}$$

What if over time we realized we never use the *odd* numbers when interacting with our context?

$$\{0, \mathbf{1}, 2, \mathbf{3}, 4, \mathbf{5}, 6, \mathbf{7}, \ldots\}$$

We then optimize our interface, we prune, we cleave, we get rid of parts of the interface we don't use:

$$\{0, 2, 4, 6, 8, 10, 12, 14, \ldots\}$$

This is called *condensation*.

For example, condensation might happen if we see the following pattern all the time:

$$\forall \epsilon > 0,\ \exists \delta > 0 \text{ such that } 0 < |x - a| < \delta \quad \Longrightarrow \quad |f(x) - \ell| < \epsilon$$

we might wish to simplify it as:

$$\lim_{x \to a} f(x) = \ell$$

At first our condensation of notation might just be “syntactic sugar”—simplified notation for convenience—but it is often the case that we end up lifting enough patterns that we build some variety of algebra and then we end up creating a new layer of abstraction altogether.

The fundamental realization (theorem) of interface theory is that the way in which an interface (weak predicate specification) condenses is proportional to—or reflective of—the equipped space (strong predicate specification). There's a direct correspondence.

Information Theory

I'm only going to give a brief informal overview of information theory and how it could relate to a theory of notation.

Information theory very much deals with *compression*. Even its primary statistical measure of **entropy**:

$$H(X) \quad = \quad -\sum_{x \in X} p(x) \log_2 p(x)$$

offers a boundary value of how much a system of information can be compressed. As for the corresponding strategy of compression: When it comes to discrete signals with complete information we can look at the frequency of word use and assign shorter codewords (condensed notation) to frequent words and longer codewords (vaporized notation) to infrequent words. You'll convey the same message—the same information—but you're using less materials on average to do so. Entropy tells you the theoretical limits of compression in this manner.

The key realization is that an interface is a form of compression [2] , and so it is natural to pull concepts and results from information theory into interface theory. On the flip side, an interface isn't only about compression, it is also about usability, user-friendliness. If information theory is about compression, it is about structural compression. For an interface theory to be meaningful, we would also need to pull concepts and results from a theory of functional compression.

[2] A tool should not be more complicated than that which it's meant to interact with.

Complexity Theory

Complexity theory deals with the functionality of compression. What does that mean? Compress! ... but do it in such a way that the compressed form is its own technology (in math that usually translates to an algebra). As example:

$$\langle 3k+4\rangle_{k\geq 0}$$

we have an infinite sequence, for which we only used 10 character symbols to represent, to interface with. That's quite the compression. What's more, our interface notation is such that it is reasonably functional, it is able to interact with other sequences of similar notation to form algebras and other functional behaviours. It's also fairly functional because we can alter the notation slightly in many ways so it's flexible in its use:

$$\langle 3k+4\rangle_{k=1}^{11} \quad = \quad \langle 7,10,13,16,19,22,25,28,31,34,37\rangle$$

Not including *whitespaces*, the representation on the left is 11 characters while the one on the right is 33 characters. That's 3x the savings already, and the notation is such that you can shift, grow, shrink the sequence, or combine it with other sequences in other ways.

Another example, a classical one, is *Cantor's diagonalization argument*. This can be interpreted as trying to compress the real number system by means of the natural numbers, and offering a proof that it is *absolutely* impossible. This is to say: The real number system is *complex* with respect to the natural numbers. This is the heart of complexity theory in regards to interfaces. [3]

The main realization of complexity theory is that functional complexity is not an objective measure, it is not a property belonging internally to an object, it is measure of comparison between two objects or two systems. Again, this is relevant to interface theory in that an interface does not exist in a vacuum, it exists relative to the context it's meant to represent. Going forward, we would need standard measures and best practices for evaluating the merit of equipping a given interface to a given context, or rather a given condensation of predicate logic notation to a given space. [4]

Grammar as Data Structure

So I've pointed to some foundational wells we can draw from in getting a feel for what interface theory is about. When it comes to mathematical interfaces, and notation in particular, the first big realization is that formulas, identities, equations can be interpreted as data structures [5] —spaces which we can navigate, for which we should devise a standardized powerful toolset.

This might not be clear at first, so I will offer an example:

$$e \quad := \quad (x+1)(x^2-1)$$

The natural question to ask is: How do we represent this as a data structure, and how do we navigate it?

Taking a global approach with the structure, we can summarize the whole of e as follows:

$$e \quad \equiv \quad \mu(\sigma[x]+\sigma^2[1])+\mu^2(\sigma(\epsilon[x]+\epsilon^2[2])+\sigma^2[-1])$$

Admittedly this is scary looking upon first glance, but it's actually pretty simple. We navigate by taking the name e as the root of the structure, and use it to refer to substructures and content as follows:

$$\begin{array}{lcl} e_{/\mu} & = & (x+1) \\ e_{/\mu/\sigma} & = & x \\ e_{/\mu/\sigma^2} & = & 1 \end{array} \qquad \begin{array}{lcl} e_{/\mu^2} & = & (x^2-1) \\ e_{/\mu^2/\sigma} & = & x^2 \\ e_{/\mu^2/\sigma^2} & = & -1 \end{array} \qquad \begin{array}{lcl} e_{/\mu^2/\sigma/\epsilon} & = & x \\ e_{/\mu^2/\sigma/\epsilon^2} & = & 2 \end{array}$$

[3] As an aside, at first I had thought *interface complexity* and *functional compression* were "two sides of the same coin" so to speak, but I've done enough independent research (thought experiments) to say they *almost* are, but not quite. There's also an idea of *relative complexity*, an example from math being "conformal mappings" in complex analysis, which intuitively are mappings which preserve angles between curves. An elementary result shows that the conjugate function $f(z)=\bar{z}$ is bijective in $\mathbb{C}$ but is not conformal anywhere as it reverses all its angle orientations. Here we can say the *absolute complexity* is equipotent (of equivalent complexity and potential), but the measure of complexity is such that compression is heuristic: When we compress heuristically, we accept that sometimes our compression may work and sometimes our compressed representation ends up using more information than the original, it backfires.

[4] Mathematicians and computing scientists are already doing this implicitly using category theory to conceptualize structures and type theory to formalize grammars.

[5] I borrow the term "data structure" from computing science but only loosely. Although there is overlap, there is also no computational nature assumed.

If it helps, you can think of these navigational paths as a *url* on a web-browser or a *pathname* from a filesystem. In particular, our ϵ is shortform for exponentiation '$*$', our μ then is multiplication '$*$', and the σ then is addition '$+$'. [6] The scary looking structure (repeated here)

$$e \quad \equiv \quad \mu(\sigma[x] + \sigma^2[1]) + \mu^2(\sigma(\epsilon[x] + \epsilon^2[2]) + \sigma^2[-1])$$

then is the whole web-directory or filesystem at a glance. The parentheses $(\cdot)$ reveal substructures, while the brackets $[\cdot]$ allow you to peek at the content at the leaves (terminal locations).

In the broadest linguistic terms, you can think of e as an expression belonging to a grammar with verbs and nouns. In our case the verbs are "exponentiation", "multiplication", "addition". Moreover, as verbs they have valency—the number of arguments you can assign—which gives us a natural (numerical) mechanism for referring to their respective arguments (σ^2 for example). Finally, to reinforce these ideas let me express one such example $e_{/\mu^2}$ in words:

"The second argument of the (root) multiplication of the expression e."

which in this case corresponds to $(x^2 - 1)$.

With that said, we can now take things further: We can simplify our notation still. For example looking at

$$\mu(\sigma[x] + \sigma^2[1])$$

the use (recycling) of the plus sign '$+$' along with the parentheses "()"—are intentional and—suggest the distributive law:

$$\mu(\sigma[x] + \sigma^2[1]) = \mu\sigma[x] + \mu\sigma^2[1]$$

Although I don't formally prove it here, it holds for the simple reason that no ambiguity is introduced in doing so. As such our original scary data structure simplifies as

$$e \quad \equiv \quad \mu\sigma[x] \ + \ \mu\sigma^2[1] \ + \ \mu^2\sigma\epsilon[x] \ + \ \mu^2\sigma\epsilon^2[2] \ + \ \mu^2\sigma^2[-1]$$

This not only reduces the use of parentheses, it also minimizes use of forward slashes '/':

$$e_{/\mu^2\sigma\epsilon} = x \qquad \text{(compare with } e_{/\mu^2/\sigma/\epsilon} = x\text{)}$$

Keep in mind, just because you *can* reveal substructures or peek at the content doesn't mean you have to:

$$e \quad \equiv \quad \mu\sigma \ + \ \mu\sigma^2 \ + \ \mu^2\sigma\epsilon \ + \ \mu^2\sigma\epsilon^2 \ + \ \mu^2\sigma^2$$

or

$$e \quad \equiv \quad \mu \ + \ \mu^2(\sigma \ + \ \sigma^2)$$

will do just fine, for example.

Let's take a momentary breather, and reflect upon all we've learned so far.

1. Mathematical notation is a mathematical object.
2. In particular, notation is an interface, meaning it is a condensed predicate logic equipped with a space.
3. Notation is a navigable data structure.

If this is the case, what is the space we're looking at here? What are we compressing with these expressions?

Identity Type theory

So we've explored notation, formulas, equations, as data structures, but up until now they've been *static*. Usually in math though, we're solving for a particular variable and in the process we tend to mutate and manipulate the original expression we started out with. We don't do this haphazardly though, we maintain "identity rules" which catalyze our manipulations.

[6] This begs the question: Why not just use the traditional 'exp' '$*$', '$+$' forms directly? They're less elegant to look at, and in fact '$+$' in particular gets confusing as we recycle it for reuse in another way.

Let's take a look at the classical three everyone knows, starting with *the associative law* $x + (y + z) = (x + y) + z$ for addition:

$$\sigma[x] + \sigma^2(\sigma[y] + \sigma^2[z]) \quad \equiv \quad \sigma(\sigma[x] + \sigma^2[y]) + \sigma^2[z]$$

next we have *the commutative law* $xy = yx$ for multiplication:

$$\mu[x] + \mu^2[y] \quad \equiv \quad \mu[y] + \mu^2[x]$$

and finally we have *the distributive law* $x(y + z) = xy + xz$:

$$\mu[x] + \mu^2(\sigma[y] + \sigma^2[z]) \quad \equiv \quad \sigma(\mu[x] + \mu^2[y]) + \sigma^2(\mu[x] + \mu^2[z])$$

It would not be unnatural here to say that if we specify a collection of identities (as above), it induces an equivalence relation for any formulaic expression. Two such expressions are then equivalent if you can derive one from the other using a sequence of identity equations within the collection. The thing worth noting here, given our "data structure" approach, is the structure changes, but there will always be *at least* one content object remaining the same among all expressions within an equivalance class. This content object being the image value, the return value if you were to evaluate the expression.

This brings up a part of the notation we haven't discussed yet. We had always started our paths at index 1: $\mu^1 + \mu^2(\sigma^1 + \sigma^2)$, etc. In the case we want to signify this *image value* we can display it with the 0 index:

$$r \equiv 1 + \mu[7] + \mu^2[3] \quad \text{or} \quad r_{/1} = 21$$

Notice here instead of μ^0 we write 1, which isn't an arithmetic law to be clear, rather just a notational convention as no ambiguity arises, and it otherwise keeps in line with arithmetic sensibilities. In a similar manner we have so far always omitted the exponent 1 in μ^1 as well, for example.

Looking at our above arithmetic identities, we can go further still and change our perspective a little: Given our previous success in breaking down the grammars of globally viewed expressions by means of paths, we can do the same here:

associative law			commutative law			distributive law		
$x \star (y \star z)$	$=$	$(x \star y) \star z$	$x \star y$	$=$	$y \star x$	$x(y + z)$	$=$	$xy + xz$
$o[x]$	$\equiv$	$oo[x]$	$o[x]$	$\equiv$	$o^2[x]$	$\mu[x]$	$\equiv$	$\sigma\mu[x]$
$o^2o[y]$	$\equiv$	$oo^2[y]$	$o^2[y]$	$\equiv$	$o[y]$		$\equiv$	$\sigma^2\mu[x]$
$o^2o^2[z]$	$\equiv$	$o^2[z]$				$\mu^2\sigma[y]$	$\equiv$	$\sigma\mu^2[y]$
						$\mu^2\sigma^2[z]$	$\equiv$	$\sigma^2\mu^2[z]$

we use '$\star$' as a generic operator and 'o' as its corresponding path filter. This is done as the associative and commutative laws holds true for many operators other than just σ, μ.

So now we have identity manipulations which we can intepret through the lens of paths. As such, we can also induce equivalence relations for these paths, creating a form of homotopy theory—intuitively at least. Realizing this, we all of a sudden have access to type theory and category theory as well to describe these patterns. For example the paths can be considered types or objects with functions or morphisms between them respectively.

Taking this approach, you might ask what's the point? You may have noticed as of yet I have not especially introduced any level of rigour within this essay while spinning these various purported theories. [7] I'll say this then: We've deconstructed the idea of notation and turned it upside down. Now we're starting with notational paths, say $\mu\sigma, \mu\sigma^2$ and we're able to combine them into larger patterns such as $\mu\sigma + \mu\sigma^2$ which refactors as $\mu(\sigma + \sigma^2)$. This allows us to view notation from type theoretic and category theoretic lenses—which specialize in mathematically complete grammars, so we can also now design notation which holds up to mathematical scrutiny. That's the point.

[7] Information Theory is real. Check it out. It's great!

The Solution

The intention of this essay was never to formalize and define specific mathematical spaces of notation with any level of rigour, only to introduce the idea that it could be done, while offering a few basic tools to get you started.

In any case, I would be remiss if I didn't demonstrate these concepts as being useful even when looking at ad-hoc notation in the wild so to speak.

To review, we had started with:

$$\int (x \ln x)^n dx \quad = \quad \int x^n (\ln x)^n dx$$

but ran into trouble forming a proper recurrence relation. I mentioned we might come up with an ad-hoc workaround, but that it's better to come up with a more systematic approach in the long run. If we were to take an ad-hoc approach though, we might generalize our original integral to the following:

$$\int x^m (\ln x)^n dx$$

for which we could then derive

$$\begin{aligned} \int x^m (\ln x)^n dx &= \tfrac{1}{m+1} x^{m+1} (\ln x)^n - \int \tfrac{1}{m+1} x^{m+1} \cdot n (\ln x)^{n-1} (\tfrac{1}{x}) dx \\ &= \tfrac{1}{m+1} x^{m+1} (\ln x)^n - \tfrac{n}{m+1} \int x^m (\ln x)^{n-1} dx \end{aligned}$$

---

which allows us to condense to

$$a_{m,n} = b_{m,n} + c_{m,n} \cdot a_{m,n-1}$$

This works! It is as minimalist as it can be, but it's also more complicated than what we were going for. However you might feel about this, it also can be said to point out a design flaw—a limitation—of standard math notation. The limitation comes from comparing the two recurrence relation models of notation: a_n, $a_{n,m}$. Our traditional notation a_n replaces $\int x^n (\ln x)^n dx$ with a in the original, and then factors out n (the variable of focus) as a subscript. The thing is, this notational design doesn't distinguish between structural locations which hold n as a value, even if these instances of n are independent of each other, and so we're required to force the issue by adding an independent variable m.

Fortunately, with our newer more rigorous notation, we can view the integral itself as a grammatical data structure:

$$\int x^n (\ln x)^n dx \quad :\equiv \quad \iota(\epsilon[x] + \epsilon^2[n]) + \iota^2(\epsilon[\ln x] + \epsilon^2[n]) + \iota^3[dx]$$

using ι as the integral verb. The location of interest would then in fact be ι^2/ϵ^2 and not ι/ϵ^2 even though they both carry copies of the same value n. Our recurrence could then be expressed as:

$$a_{/\iota^2\epsilon^2[n]} = b_n + c_n \cdot a_{/\iota^2\epsilon^2[n-1]}$$

though if we shortformed our navigational path as p this would simplify to $a_{/p[n]}$. Then again, if the context is clear we could even reduce the recurrence back down to

$$a_n = b_n + c_n a_{n-1}$$

so long as we know the difference conceptually.

In any case, Our antiderivative strategy now becomes a matter of solving for the recurrence relation itself. The question then to ask is: Does this recurrence have a closed form? I'll tell you the answer now is a yes. But, as we've abstracted the problem and we're only dealing with a_n now, there's no value in looking for patterns in the sequence $a_0, a_1, a_2, \ldots$ as they're just the letter 'a' with numerical subscripts. We instead have to work the problem from the other end. Let's try a few cases in an analogous manner to the general

strategy we held with $(\ln x)^n$ within our integral:

$$\begin{aligned}
\mathbf{a}_n &= b_n + c_n \mathbf{a}_{n-1} \\
&= b_n + c_n(b_{n-1} + c_{n-1}\mathbf{a}_{n-2}) \\
&= b_n + c_n b_{n-1} + c_n c_{n-1}\mathbf{a}_{n-2} \\
&= b_n + c_n b_{n-1} + c_n c_{n-1}(b_{n-2} + c_{n-2}\mathbf{a}_{n-3}) \\
&= b_n + c_n b_{n-1} + c_n c_{n-1} b_{n-2} + c_n c_{n-1} c_{n-2}\mathbf{a}_{n-3}
\end{aligned}$$

I've boldfaced our a to clarify focus. Regardless, this doesn't look too promising. As it turns out, even this is a fair bit complicated, but we've run out of other options, so what do we do now?

Self-similarity

If interface theory's only offering was "notation as data structure", it wouldn't be worth your time. What else is there then? The answer to that is **self-similarity**.

Going back to information theory and complexity theory, the running theme for both was *compression*. Energy conservation is a necessity of life, and compression allows for such conservation. Patterns such as *symmetry* and more broadly self-similarity conserve informational energy—they give us ways to reduce the amount of information needed to represent an object while allowing us to reproduce the original as needed. [8] For example if an informational object has a reflective symmetry, then the amount of information needed to represent this object can be cut in half. [9] If you have rotational symmetries, then by way of group theory, all you need are generators and the rotational operator to reconstruct the object as a whole. Again, this allows for an idealized compression. Fractal self-similarity can be categorized similarly.

The other advantage of self-similarity is encoded within general patterns of design. As data structures (source code for example) are iteratively extended—as they grow and become more complicated—we end up needing ways to mitigate their complexity to keep them balanced. In general, we appeal to strategies from theories of design and engineering. In particular, the underlying truth in any modularization scheme of design is: You guessed it, self-similarity. The reason for this is simple: If our structure has inherent (native support for) self-similar patterns, then as the structure grows we can either fold the growth back into the structure (compression), or it extends repetitively without interfering with its surroundings. It possesses *scalability*.

Getting back to our recurrence relation, we're now on the lookout for structural patterns of self-similarity:

$$\begin{aligned}
a^{(0)} &= b_n + c_n \mathbf{a}_{n-1} \\
a^{(1)} &= b_n + c_n(b_{n-1} + c_{n-1}\mathbf{a}_{n-2}) \\
&= (b_n + c_n b_{n-1}) + (c_n c_{n-1})\mathbf{a}_{n-2} \\
a^{(2)} &= (b_n + c_n b_{n-1}) + (c_n c_{n-1})(b_{n-2} + c_{n-2}\mathbf{a}_{n-3}) \\
&= (b_n + c_n b_{n-1} + (c_n c_{n-1})b_{n-2}) + (c_n c_{n-1} c_{n-2})\mathbf{a}_{n-3}
\end{aligned}$$

[8] This is only personal conjecture here, but I suspect this is the reason humans find symmetry as aesthetic and artistically pleasing as we do.

[9] Technically it's half plus the small amount of overhead needed to specify reconstruction based on the reflection.

This time around, a few well placed parentheses and our pattern stands out:

$$\begin{aligned} a^{(0)} &\equiv \sigma + \sigma^2(\mu + \mu^2) \\ a^{(1)} &\equiv \sigma + \sigma^2(\mu + \mu^2) \\ a^{(2)} &\equiv \sigma + \sigma^2(\mu + \mu^2) \end{aligned}$$

We didn't notice it earlier because we tend to take the associative law for granted. It's not difficult to prove this property scales to all iterations using mathematical induction. So our recurrence relations all consist of the grammatical paths

$$a^{(s)} \equiv \sigma + \sigma^2\mu + \sigma^2\mu^2$$

At this point, we might be willing to make a guess that

$$a^{(s)}_{/\sigma^2\mu^2} = a_{n-(s+1)}$$

The inductive proof is outlined as follows:

$$\begin{aligned} a^{(s)} &\equiv \sigma^2\mu^2\ [a_{n-(s+1)}] \\ a^{(s+1)} &\equiv \sigma^2\mu^2\sigma^2\mu^2\ [a_{n-(s+2)}] \\ &\equiv \sigma^2\sigma^2\mu^2\mu^2 \\ &\equiv \sigma^2\mu^2\ [a_{n-(s+2)}] \end{aligned}$$

Notice how we've hidden the majority of the grammatical structure in these notational expressions? With this I am finally able to demonstrate some of the advantages in developing a refined knowledge of notation. When we're able to navigate any part of the structure, we can isolate our focus accordingly.

I admit the above algebra may look foreign at first, but at one point the associative, commutative, and distributive laws themselves looked foreign at first as well. I assure you, a little practice with the reference table from earlier and these manipulations will begin to look pretty evident.

There are two more locations to derive. Let's start with $\sigma^2\mu$:

$$\begin{aligned} a^{(0)}_{/\sigma^2\mu} &= c_n \\ a^{(1)}_{/\sigma^2\mu} &= c_n c_{n-1} \\ a^{(2)}_{/\sigma^2\mu} &= c_n c_{n-1} c_{n-2} \end{aligned}$$

If we were to take a guess, it's a product of c's. True, in particular it's:

$$a^{(s)}_{/\sigma^2\mu} = \prod_{0 \le p \le s} c_{n-p}$$

which is easily proven with mathematical induction. The pattern within our final path location σ is a bit more complicated:

$$\begin{aligned} a^{(0)}_{/\sigma} &= b_n \\ a^{(1)}_{/\sigma} &= b_n + c_n b_{n-1} \\ a^{(2)}_{/\sigma} &= b_n + c_n b_{n-1} + c_n c_{n-1} b_{n-2} \end{aligned}$$

We could guess as with the previous, but let's take a bigger picture approach once again. The key realization if we look at the previous two cases, is this grammatical self-similarity allows us to model recurrence relations within our recurrence relation. So let's start over from this perspective, and see where this idea takes us:

$$\begin{aligned}
a^{(s)} &\equiv \sigma[B_s] + \sigma^2(\mu[C_s] + \mu^2[A_s]) \\
&= B_s + C_s \cdot A_s \\
&= B_s + C_s \cdot a_{n-(s+1)} \\
a^{(s+1)} &= B_s + C_s(b_{n-(s+1)} + c_{n-(s+1)} \cdot a_{n-(s+2)}) \\
&= (b_{n-(s+1)}C_s + B_s) + (c_{n-(s+1)}C_s)a_{n-(s+2)} \\
&\equiv \sigma[b_{n-(s+1)}C_s + B_s] + \sigma^2(\mu[c_{n-(s+1)}C_s] + \mu^2[a_{n-(s+2)}]) \\
&\equiv \sigma[B_{s+1}] + \sigma^2(\mu[C_{s+1}] + \mu^2[A_{s+1}])
\end{aligned}$$

Pulling out these derivative recurrences, and shifting by one we obtain:

$$\begin{aligned}
A_s &= a_{n-(s+1)} \\
C_s &= c_{n-s}C_{s-1} \\
B_s &= b_{n-s}C_{s-1} + B_{s-1}
\end{aligned}$$

Given our $C_s = \prod_{0 \le p \le s} c_{n-p}$, it's not too hard to figure out

$$B_s = \sum_{0 \le \ell \le s} b_{n-\ell} \prod_{1 \le p \le \ell} c_{n+1-p}$$

which technically holds if by convention you define

$$\prod_{\text{false}} = 1$$

In any case, the solution to our recurrence can now be summarized as:

$$a_n = b_n + c_n a_{n-1} \quad \Longrightarrow \quad a_n = \sum_{0 \le \ell \le s} b_{n-\ell} \prod_{1 \le p \le \ell} c_{n+1-p} \quad + \quad a_{n-(s+1)} \prod_{0 \le p \le s} c_{n-p}$$

which if we let $s = n - 1$, our final term ends in a_0. In this case, now returning to our original integral we can finally solve for the antiderivative:

$$\int (x \ln x)^n dx \quad = \quad \frac{(-1)^n \, n! \, x^{n+1}}{(n+1)^{n+1}} \sum_{0 \le j \le n} \frac{[-(n+1)\ln x]^j}{j!}$$

Sophomore's Dream and Tetration

We'll end with tetration and the sophomore's dream.

Tetration if you don't know, is the next arithmetic operator in line after exponentiation. For example if multiplication is addition n times (say $n \times m$), and exponentiation is a product n times (say m^n), then tetration is exponentiation n times:

$$m^{m^{m^{\cdots^m}}} \quad (n \text{ times})$$

I highly recommend looking into tetration, it's a fascinating operator (if not as well known as the common three). In any case, we arrive at the sophomore's dream when looking at the simplest (non-trivial) case, $n = 2$:

$$m^m$$

In particular, extending this to the reals

$$x^x$$

one asks what the derivative is:

$$\begin{aligned}\frac{d}{dx}x^x &= \frac{d}{dx}e^{x\ln x}\\ &= e^{x\ln x}\frac{d}{dx}x\ln x\\ &= x^x(\ln x + 1)\end{aligned}$$

Then how about its antiderivative? Assuming uniform convergence, we have:

$$\begin{aligned}\int x^x dx &= \int e^{x\ln x}dx\\ &= \int \sum_{n\geq 0}\frac{(x\ln x)^n}{n!}dx\\ &= \sum_{n\geq 0}\frac{1}{n!}\int (x\ln x)^n dx\end{aligned}$$

which, interestingly leads us to the antiderivative we've already solved for. In particular, if you extend our antiderivative back to the form $\int x^m(\ln x)^n dx$, we have:

$$\int x^m(\ln x)^n dx \quad = \quad \frac{(-1)^n\, n!\, x^{m+1}}{(m+1)^{n+1}}\; e_n^{-(m+1)\ln x}, \qquad \text{where} \quad e_n^t = \sum_{0\leq j\leq n}\frac{t^j}{j!}$$

The sophomore's dream is the "too good to be true" identity (which is in fact true):

$$\int_0^1 \frac{1}{x^x}dx \quad = \quad \sum_{k\geq 0}\frac{1}{(k+1)^{k+1}} \quad = \quad 1 + \frac{1}{2^2} + \frac{1}{3^3} + \frac{1}{4^4} + \dots$$

I recommend looking into the sophomore's dream as well as tetration. It's been thoroughly studied. I

myself have derived the following identities using our previously solved antiderivative:

$$\begin{aligned}
\int_0^1 \frac{x^n}{(x^x)^w}dx &= \sum_{k\geq 0}\frac{w^k}{(n+k+1)^{k+1}} \\
&= \frac{1}{n+1}+\frac{w}{(n+2)^2}+\frac{w^2}{(n+3)^3}+\dots \\
\int_0^1 f(\ln\left(\frac{1}{x^x}\right)^w)dx &= \sum_{m\geq 0}\frac{f^{(m)}(0)}{(m+1)^{m+1}}w^m \\
&= -\int_0^1 \ln x\ f(\ln\left(\frac{1}{x^x}\right)^w)dx \\
\int_0^1 \frac{x^m(\ln x)^n}{(x^x)^w}dx &= (-1)^n\, n!\sum_{k\geq 0}\binom{n+k}{n}\frac{w^k}{(m+k+1)^{n+k+1}} \\
\int_0^1 \frac{dx}{1+wx\ln x} &= \sum_{m\geq 0}\frac{m!}{(m+1)^{m+1}}w^m, \quad |w|<e
\end{aligned}$$

but others have found other interesting results, not to mention more elegant solutions to the sophomore's dream using the gamma (factorial) function for example.