Normal Paths

by Sven Nilsen, 2019

A normal path is what I consider the most important idea in mathematics. Like most important ideas, it is very simple in one way, but very complex and deep in other ways. Normal paths were the reason I started studying path semantics in the first place, but even at the time, it was unknown how important they turned out to be. In some sense normal paths is the primary motivation, the idealized goal, of using mathematics itself. This is because finding normal paths is very desirable in a world where cheap computers exist, for economic or intellectual purposes.

> *Normal paths could be called "The Secret of Mathematics" - Sven Nilsen, 2019*

A normal path represents the idea of making perfect predictions in general. A perfect prediction has no uncertainty in it, even if the thing we are talking about is a very complex system. It is not always possible to make perfect predictions, but we would prefer to make them perfect, if we could. The rest of mathematics is basically about creating abstractions that approximate the semantics of normal paths. All "paths" are essentially analogies of normal paths suited for various domains of reasoning.

Normal paths are classified in two ways:

- "Symmetric" or "Asymmetric"
- "Unconstrained" or "Constrained"

The word "normal" is used since when e.g. talking about paths in general, it is assumed that the path is normal unless specified otherwise. E.g., a "symmetric path" is actually a "symmetric normal path".

A symmetric path is when you think about some property of a system. It is symmetric because even the underlying system changes over time, what you mean, reason, observe and measure stays the same. Symmetric paths are useful because it lets us simplify the world around us and make predictions.

For example, the energy of a battery can be written as a sub-type:

```
x : [energy] y

energy : battery → joule
```

The `energy` function takes some variable `x` and computes a variable `y`. Here, `x` represents some battery and `y` represents the available energy measured in Joule. We say that the `energy` function is a "property" of the battery.

We measure energy the same way over time, to make it easy to reason about systems such as batteries.

While measuring the energy of batteries is fine, it does not actually make sense on its own. Why are we measuring the energy? It is because we are not just measuring, but comparing or tracking objects over time. Mathematically, we speak of "functions of systems" as the underlying abstraction. This abstraction might be unknown, e.g. too complex to write down. Yet, this has not stopped us at all.

The really genius thing about mathematics is that it has a secret. A BIG SECRET. It does not matter that some system is too complex to describe in detail, when it is sufficient that some property of the system can be fully described and predicted perfectly!

The trick is to assume that even if the system is too complex, one can *pretend* that is simple.

For example, one can imagine a world where nothing changes. Whatever a battery is, one can think about this world as a function `id` that takes any input and returns it as output:

$$id_T(x : T) = x$$

$$id : T \to T$$

The symmetric path of `id` with respect to the property `energy` is the following:

$$id[energy] <=> id$$

The `id` function on the left side takes a `battery` and returns a `battery`, but the `id` function on the right side takes `joule` and returns `joule`:

$$id_{battery}[energy] <=> id_{joule}$$

Whatever available energy the battery has in one moment, it will have the same available the next moment. It does not matter what the available energy is. In some sense, we are not even thinking about a particular world, but about any world that remains unchanged over time. It does not even matter what units of time we use, or what “the next moment” means.

Whenever we have a world that remains unchanged over time, we can predict the energy of batteries in this world by substituting:

$$id_{battery}[energy]$$

With:

$$id_{joule}$$

These two functions are identical. Since they are identical, they also have the same type:

$$id_{battery}[energy] : joule \to joule$$
$$id_{joule} : joule \to joule$$

The `$id_{battery}$` function is too complex to write down, because we do not know how to describe a battery in full detail. Yet, by “focusing” on the property `energy`, the world can be simplified.

Mathematics is very flexible when used this way, because it lets us reason about things that otherwise would be too complex to comprehend. Our brains are too simple to fully understand what objects like batteries are in the physical world. On the other side, representing the energy of some battery with a number is much easier.

Notice that I try to explain how we go from the complex physical world to some mathematical function. Usually, when you learn mathematics in school, the teacher writes down the energy like this:

    y : joule

This does not explain where `y` comes from. It does not explain what it means. The mathematics taught in schools today does not teach students what mathematics means, why we are using mathematics. Students learn what it means from experience, so they have to guess how it works.

The proper way of explaining `y` is by writing:

    x : [energy] y

    x : battery
    y : joule

From this we understand that the `energy` function has the type `battery → joule`.

Similarly, in any world where nothing changes, the energy of batteries do not change:

    $id_{battery}$[energy] <=> id_{joule}

In general, one can imagine some world represented by `f` and predicted by `h`:

    f[energy] <=> h

    f : battery → battery
    h : joule → joule
    energy : battery → joule

If we replace `energy` with some function `g`:

    f[g] <=> h

    f : A → A
    g : A → B
    h : B → B

This is a *symmetric path*.

Symmetric paths are much more generic than most things humans are used to reason about. This is why they are less intuitive. However, once you get used to think this way, you learn to understand that any system where some property is measured the same way and predicted perfectly, can be formalized as a symmetric path.

It does not matter whether the underlying system is complex or not, as long the property of the system is perfectly predictable. For example, it is known that space has 3 dimensions and time has 1 dimension in the Euclidean approximation of local space-time. This property is perfect predictable and holds for most practical situations, but might break down e.g. when traveling near the speed of light.

Physical particles in the Standard Model have properties such as charge and mass that are perfectly predictable. At least they give predictions that are correct within measurement error. From these particles one can create any normal matter, such as the building blocks of DNA, cells, bacteria and humans. By summing up the mass of individual particles and the mass from energy bonds, one can calculate the mass of a human being.

Perfect predictions are not as uncommon as many people think. There are some predictions that are hard to make, such as rolling dices, but there are other predictions that are easy to make, such as whether the Earth will rotate tomorrow. When we substitute highly probabilistic events with functions used for perfect predictions in other contexts, the meaning of the prediction changes from perfect to heuristic. In many cases we can not tell the difference between a likely heuristic prediction and a perfect one, not even if we wait a billion years.

It would be ideal to be able to predict anything perfectly, but unfortunately life is not that easy. The reason we build all these mathematical abstractions is because normal paths do not work all the time. Either it is because they do not fit observations, or it is because a perfect prediction might be impossible, even in principle. This is the case in quantum physics. There are also problems where the meaning of the prediction we want, is not the same as for normal paths, such as in calculus of derivatives. However, the most common way of using calculus is to create approximate solutions that are used to make predictions about the physical or some hypothetical world.

Normal paths fit into the very motivation of using mathematics. If complex systems could not be simplified to math and there was no way of keeping track of what we were describing, then mathematics would not been developed historically. The reason mathematics was developed is because of the very structure of normal paths, existing within mathematics. Most of the practical applications of mathematics use normal paths, and we create new abstractions because normal paths do not fit the problem. However, the intention is to find something that looks like, or helps us construct something that behaves similar to, normal paths.

I use normal paths as a way of seeing mathematics. It works both as an explaination tool to learn why mathematics makes sense, and as a filter to see other things that do not fit into this picture. Normal paths contains infinite number of mathematical objects, that are practical in some sense, such that all other mathematical objects are not practical in the same way. From this it is possible to argue that whenever we want to solve a problem, the solution includes the meaning of some normal path implicitly. Therefore, I see normal paths as “The Secret of Mathematics”. I believe that mathematics on a practical level is dominated by normal paths, like a skeleton, or a high road, a central theme, while the rest is filling out the details or to overcome difficult problems. Normal paths are not easy, they are intrinsicially hard, but they are not as hard as the difficult and less used parts of mathematics.

The notation of asymmetric path is a generalization of symmetric paths.

For example:

$$id_{battery}[energy] <=> id_{joule}$$

Can be written in asymmetric notation:

$$id_{battery}[energy \rightarrow energy] <=> id_{joule}$$

A symmetric path means that the property applies to every input argument and the output.

For example, in Boolean algebra:

$$\text{and}[\text{not}] <=> \text{or}$$

By switching every 0 into 1 and every 1 into 0, AND can be predicted by OR.

This can also be written as an equation (`∀` means “for all”):

$$\forall\ a, b\ \{\ \text{not}(\text{and}(a, b)) = \text{or}(\text{not}(a), \text{not}(b))\ \}$$

This particular law is one of the two known as De Morgan’s laws. The other follows from:

$$\text{or}[\text{not}] <=> \text{and}$$

In general, an asymmetric path of the form:

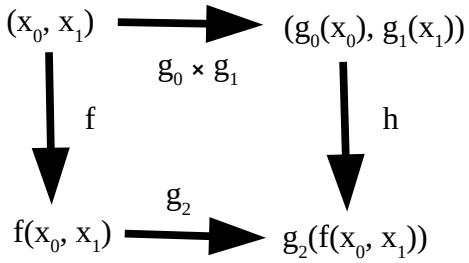
$$f[g_0 \times g_1 \to g_2] <=> h$$

Can be written as an equation:

$$\forall\ x_0, x_1\ \{\ g_2(f(x_0, x_1)) = h(g_0(x_0), g_1(x_1))\ \}$$

This means that normal paths are not weird, or “special”, seen from a mathematical perspective, but just ordinary expressions of a certain kind. It is a small sub-set of all equations.

One can also think about normal paths as a commutative square:



For all computer programs, all perfect predictions happen implicitly, without extra cost, as a mathematical parallel computation.

Programming languages uses types to help programmers avoid mistakes. A type is a perfectly predictable property of the data. Therefore, normal paths are a *generalization* of type theory. For this reason, normal paths are more powerful than dependently typed systems. Since dependently typed systems are undecidable, this also means that normal paths are not possible to implement fully on a computer.

So far I have talked about symmetric paths and asymmetric paths. A symmetric path is when we measure a property of some system in the same way. Asymmetric paths measure one property of a system from other properties. Now it is time to talk about unconstrained vs constrained normal paths.

The examples I used previously for symmetric and asymmetric paths were unconstrained. Usually, you do not have to worry much about constrained normal paths. They are there to fill out the theoretical gaps of normal paths, but are less used than unconstrained normal paths.

The rest of this paper is more technical, to explain experts how constrained normal paths work.

An unconstrained normal path is the following:

$$f\{true_1, true_1\}[g_0 \times g_1 \to g_2] <=> h$$

$$true_{1\,T}(x : T) = true$$

Here, `h` is a unique solution and `f` is a total function. All total functions have `$true_N$` as constraint.

Constrained normal paths are normal paths of partial functions, where `<=>` is changed into `=>`:

$$f\{c_0, c_1\}[g_0 \times g_1 \to g_2] => h$$

Here, `=>` means that the left side can be replaced by `h`, but it is not a unique solution. This means that we might make an error by going in the opposite direction. It is still the same right-to-left.

The equation of a constrained normal path of two arguments is the following:

$$\forall\ x_0 : c_0, x_1 : c_1\ \{\ g_2(f(x_0, x_1)) = h(g_0(x_0), g_1(x_1))\ \}$$

This means that `$x_0$` and `$x_1$` are filtered, or constrained, by `$c_0$` and `$c_1$` respectively.

In standard mathematics, `$c_0 \times c_1$` are the domain of the function `f`. However, many people think of the domain as something separate from the function itself, which might lead to unsoundness when doing theorem proving. In path semantics, the rules are more strict and requires that the domain is a part of the *identity* of the function. When the domain changes, the *identity* of the function changes.

Therefore, constrained normal paths are not the same as unconstrained normal paths. They have different meta-properties. However, with respect to computation under substitution only, all properties are the same. This means that constrained normal paths behave similar to unconstrained normal paths from right-to-left, but are different from left-to-right.

The *identity* of the function changes because in order for two symbols to be identical, the two collections of symbols associated with the two symbols must also be equivalent. The right side counts as an associated symbol.

In logic, this is called “Leibniz’ law” and holds for predicates, but in path semantics it is interpreted as a collection of associated symbols in general. Associated symbols can be in abstract sense, e.g. include the symbols that we use in future versions of path sematics.

To deal with the problem of identity change of functions, it is common to use something called “constrained functions” in path semantics. It overloads the classical way of thinking about functions.

In path semantics using constrained functions, the domain is called “trivial path” and written `∀f`:

f <=> f{∀f} Every constrained function has a trivial path

This means, that the trivial path is always there and part of the function identity.

Function identity is particularly important when generalizing normal paths to probabilistic paths. Unlike a normal path, a probabilistic path is always defined, but only for finite types. The problem of probabilistic paths is that they encode conditional probabilities implicitly, such that it satisfies probability theory if and only if the correct interpretation of function identity is used.

It is impossible to talk about functions without indirectly mean something about the domain.

Another place where function identity is important is for semantics of choices. Any program executing on a Turing machine is understood as a construction where the input of a function is concrete. With other words, it has a unique domain value. This process of constructing programs for Turing machine through making choices is called “elimination of choices”.

In theorem proving, to say that a problem is decidable means the existence of some process that eliminates all choices. From the perspective of choices and theory about informal theorem proving, decidable is a destructive process and not constructive. Informal theorem proving is considered more powerful than formal theorem proving. With other words, anything executing on a Turing machine is different than what we usually mean by functions in mathematics. Mathematical functions, deterministic or not, always include some element of choice, that is implicitly understood from the practical usage of programming languages. Elimination of choice means constraining functions gradually until a unique input is found, at which point the output of the executed program has some meaning.

Since Turing machines eliminate choices, it also means that the parallel computation performed by normal paths also has a concrete value. So, even if the identity of functions changes through executing them on a Turing machine, the identity of the normal path changes in a corresponding way such that the meta-properties that hold for the normal paths are true for the executed program.