Homotopy Arithmeticy

by Sven Nilsen, 2025

In this paper we introduce a simple model for Homotopy Theory.

Consider the sets of functions of type `bool → nat & (< n)` for some `n`.

Naturally, a function `f : bool → nat` corresponds to a tuple `(a, b)` where:

f(false) = a f(true) = b

A groupoid structure on some set `**S**` has the following constraint:

$f \in \mathbf{S} \quad \rightarrow \quad f^{-1} \in \mathbf{S}$

Where `$f^{-1}$` is defined as following:

$x : bool \quad \rightarrow \quad f^{-1}(x) := f(!x)$

For example:

$(1, 2) \in \mathbf{S} \quad \rightarrow \quad (2, 1) \in \mathbf{S}$

Since a groupoid is a category, one also has transitivity:

$(a, b) \in \mathbf{S} \ \& \ (b, c) \in \mathbf{S} \quad \rightarrow \quad (a, c) \in \mathbf{S}$

Transitivity and symmetry is the same as a partial equivalence.

When `$(a, a) \in \mathbf{S}$`, one can write this as `$a \in \mathbf{S}$`.
This should not be confused with identity morphisms,
because the objects in the groupoid are tuples.

To create a model of Homotopy Theory, we want to extend the notion of groupoids to n-groupoids.
The natural way to do this is by allowing substitution in the following way in sub-expressions:

a <=> (a, a)

For example, for a tuple `(a, b)`, one can get `((a, a), b)`.

This can be thought of as extending and contracting functions `f : bool → nat, f' : $bool^2$ → nat`.

f <=> f'

From the perspective of theorem proving, this is like adding or removing a parameter.

By considering all possible inputs, one can add a normalization procedure which collects all leaves in sub-expressions and constructs an ordered set.

For example, `((a, b), (c, a))` gets normalized to `(a, (b, c))`.

The normalization algorithm corresponds to treating functions as equal by their codomains.

Remember that the objects in this n-groupoid are tuples, so although one can normalize to `a`, this does not mean that the n-groupoid contain all possible objects `a, b, c, d, ...`. We need to add every object explicitly to the n-groupoid. The semantics of `a` will always be a tuple `(a, a)`.

This means that the structure of such n-groupoids is partial equivalence,
because there is symmetry and transitivity, but no reflexivity.

Now, we have a model of Homotopy Theory with proof semantics.
Yet, we can also move on to do Path Semantics in this model.

Consider a set `**S**` of such n-groupoids which do not share any object among themselves:

$$f \in \mathbf{G_n} \;\&\; g \in \mathbf{G_m} \;\&\; (f == g) \quad \rightarrow \quad \mathbf{G_n} == \mathbf{G_m} \quad \text{for all } \mathbf{G_n}, \mathbf{G_m} \in \mathbf{S}$$

For every `f : bool^(n : nat & (> 0)) → nat`, we can figure out which n-groupoid in `**S**` it belongs to.
Often, it is sufficent to know that `f` belongs to some n-groupoid, using path semantical qubit `~`:

$$f \in \mathbf{G_n} \quad <=> \quad \sim f \quad \text{for all } \mathbf{G_n} \in \mathbf{S}, \text{ where } <=> \text{ is a bit “magical”}$$

Translating the property of `**S**` to using path semantical qubit `~`:

$$\sim f \;\&\; \sim g \;\&\; (f == g) \quad \rightarrow \quad \mathbf{G_n} == \mathbf{G_m} \quad \text{for all } \mathbf{G_n}, \mathbf{G_m} \in \mathbf{S}$$

Or, using the definition of path semantical quality `~~` and adding recursion on higher sets:

$$f \sim\sim g \quad \rightarrow \quad \mathbf{G_n} \sim\sim \mathbf{G_m} \quad \text{for all } f \in \mathbf{G_n}, g \in \mathbf{G_m}, \mathbf{G_n}, \mathbf{G_m} \in \mathbf{S}$$

Path semantical quality `~~` preserves the partial equivalence structure in these n-groupoids.

Now, there is a particular interesting class of such sets `**S**`, which has the following property:

$$a \mathrel{!}= b \quad \rightarrow \quad \sim(a, b) \quad \text{for all } a, b, \text{ where } (a, b) \in \mathbf{G_n}, \mathbf{G_n} \in \mathbf{S}$$

This means, whenever `a` and `b` are distinct, the tuple `(a, b)` belongs to some n-groupoid in `**S**`.
By symmetry and transitivity, if `(a, b)` belongs to some n-groupoid in `**S**`, then `(a, a)` and `(b, b)` belongs to the same n-groupoid in `**S**`. I will get to the issue of totally defined inequality later.

The path semantical structure of `**S**` captures the notion that each n-groupoid consists of a number of pieces. Each piece only belongs to one particular n-groupoid and none of the others. When one consider a topology on such pieces, ignoring the numeric values, the n-groupoids become the classifying spaces of topology.

For example, if `!~(a, a)`, then it can not belong to any n-groupoid,
plus that no tuples can contain `a` in any of the n-groupoids, anywhere in `**S**`.
With other words `!~(a, a)` means a hole.

A hole does not belong to any particular n-groupoid in `**S**`. Furthermore, we are not stating our notion of continuity explicitly here. The definition of topology might vary between pieces, even within the same n-groupoid. We only know that whatever topology we use, holes tell us the global constraints on these topologies.

For example, when some topology on a piece is a dougnut (volume) or torus (surface), a dougnut as a hole in the middle and a torus has two holes: One in the middle and one in the interior. This means, you can not assign a dougnut to the same set `**S**` as to a torus where the torus is the surface of the doughnut. Why not? Because when the torus is the surface of the dougnut, they share the hole in the middle, but the dougnut connects the hole in the interior, which is not allowed.

Similarly, you can not use some topology where a ball (volume) and sphere (surface) in the same set `**S**`, where the sphere is the surface of the ball. However, you are allowed to do it when the sphere is the surface of another ball. Therefore, when we have a ball and a sphere in the same set `**S**`, we know that they belong to two different topologies.

This way of reasoning makes it possible to deduce properties of the set `**S**` with some added topology per piece in its n-groupoids, without knowing which choice of topology we have made.

In physics, a “hole” can be e.g. a wall that no object can penetrate through. We do not know, nor care, about the specific topology that these objects have, but we know for sure that they can not pass through that specific wall. If we had to define the topology for every object each time, then life would become very hard for Path Semanticists. However, Path Semantics usually do not worry about topology, but about the constraints and language biases of some mathematical language.

For example, you can use a container, e.g. a bucket and fill it with water. The bucket can be empty or filled with water, but it can not be both empty and filled at the same time. Here, the empty bucket functions like a “surface” on the filled bucket as “volume”. They can not both belong to the same set `**S**`. However, you might create a map of type `bool → **S**` to describe empty/filled buckets.

Remember that we only consider functions that maps to natural numbers up to some `n : nat`.

For `n = 2`, we have:

(0, 1) (1, 0)

Using symmetry and transitivity, one can show that there is only one n-groupoid that contains:

(0, 0) (0, 1) (1, 0) (1, 1)

In fact, this holds for any set `**S**` when we use totally defined inequality with the rule:

a != b → ~(a, b) for all `a, b`, where `(a, b) ∈ $\mathbf{G_n}$, $\mathbf{G_n} \in \mathbf{S}$`

When inequality is not total, we might have multiple n-groupoids in `**S**`.

Furthermore, since a total defined inequality operator spawns tuples for every pair, it means that everything in the single n-groupoid gets connected to everything else. It fills the space entirely.

Such spaces behave like propositions. Any member is as good as any other member, just like a proof of some proposition is just as good as any other proof of the same proposition.

Still, propositional spaces can contain a hole. When there is a hole, there is just one n-groupoid and it contains zero objects. This propositional space corresponds to `false` in logic. All functions belong to the same n-groupoid, but we did not say there could not be zero functions involved. In a such case, the inequality is not well defined for any input and can be ignored.

This means that there is an analogue of propositions in any choice of topology over such sets `**S**`.

When interpreting this physically, one can think about it as no matter how we interpret physical reality, there will be some concept that corresponds to truth, just by thinking about physical reality.

Now, by thinking of `true` and `false` as propositional spaces, they might be thought of as the extreme maximum and minimum spaces, which both contain only a single n-groupoid. Any other partially defined inequality puts distinct numbers in some amount of n-groupoids, but we do not know how many there are. All we know is that the numbers are paired, so all their connections belong to the same n-groupoid.

It is kind of like a person who searches for a soulmate among the other living beings on the same planet. One person might have multiple soulmates, but there exists at least one soulmate per person.

In biology, for every species that reproduces sexually, there has to be at least one individual per biological gender for the species to survive. Otherwise, the species will go extinct or is already extinct. Since reproduction happens sexually, the species needs instincts programmed into the DNA of individuals to seek out reproduction this way. While not all species raise children as couples, there must at least be some method a species uses to reproduce using instincts or behavior.

So, the idea of using inequality to spawn connections into n-groupoids, is very useful when interpreting the physical world. A lot of this mathematics makes sense as physical structure.

There is also an element of the unknown when inequality is partially defined. When a person is born, it is not certain from the perspective of the person which planet they live on. The person has to learn and explore to figure it out. In a sense, it could be possible in principle that all living beings existed only on a single planet. However, there can be also multiple planets. The undefined behavior of inequality prevents us from assuming that we know these things, when we in fact do not know them. We always have to investigate the particular constrains on topology to learn more.

For example, there could be a constraint in biology such that carbon based life forms are incompatible with silicon life forms and vice versa. This could produce an instable equilibrium where a planet tends to support carbon based life forms, but not silicon life forms, or supports silicon life forms, but not carbon based life forms. These two life forms might be compatible for a short period of time, but perhaps getting more and more unstable in the long run. We do not know. When stated in general, this has nothing to do with AI versus humans in particular, but for biology.

In constructive logic, we can not prevent anyone from adding new axioms, unless the language the constructive logic is implemented in does not support introducing axioms. This means, we can only prove theorems in the context of an unknown amount of n-groupoids, using this model of Homotopy Theory.

Notice that this model is much simpler than constructive logic. The utility of using a such simple model is to learn what Homotopy Theory means in the context of theorem proving. It can help build intuition about what language biases there are in constructive logic.

If two pieces within the same n-groupoid are separated, then one might move one piece to a second n-groupoid. Or, one might move two pieces from two n-groupoids into a one n-groupoid. There can be an infinite number of empty n-groupoids. Kind of like a diary that has a large number of blank pages. The meaning of a each n-groupoid is chosen arbitrarily. You can add your own meaning to the model. However, one piece can only be assigned to one n-groupoid. To work around this problem, you can use indices paired with pieces, e.g. `(0, a), (1, a), (2, a), …, (n, a)`.

While the limitation of keeping one n-groupoid per piece might seem overly restrictive at first, it is relatively easy to work around this limitation. Furthermore, when only looking at the pieces of the entire set `**S**`, by adding a time parameter, `time → **S**`, one might think of these pieces as moving around dynamically inside `**S**`. When there is no change in the definition of partial inequality, the pieces can move around but not be glued together or teared apart. This is a natural property of topological spaces. Therefore, `time → **S**` is a way to model topology easily. The rules that constrain how pieces move around between n-groupoids define the particular topology being used.

To describe an initial configuration with a topology `time → **S**`, one needs two pieces of information: The partial defined inequality operator, plus some map `init : nat$^2$ → nat` that tells which n-groupoid to put `(a, b)`, by calling `init(a, b)`. This returns the index of the n-groupoid to use. Now, this is a set `**S**`, so this technique does not tell you how to look up the n-groupoid. You can use a list of n-groupoids instead, which gives a natural way to initialize a state.

A rule can be thought of as a map: `rule : **S**$^2$ → bool`. It tells whether first state, first argument, can change into a second state, second argument. If you want to do probabilistic simulations using Markov chains, then you can return a probability instead of boolean: `rule : **S**$^2$ → prob`.

In many applications, e.g. video games, there is no reason to glue or tear apart objects. This means, the number of pieces can remain constant over time. Some game engines simply hide objects that are not visible on the screen and perform the same logic on both dead and alive entities, since the cost of CPU branching per entity can be more expensive than simulating a fixed array of entities. This technique also gives better predictability regarding performance, since the worst case of the algorithm gets easier to reason about.

Programmers rarely care about the topology of entities in their video games. Since the topology in this model is undefined, but also one can reason about holes, there is no need to add a specific topology but concentrate of what matters for the particular task at hand. This is usually reasoning about holes, instead of reasoning about how objects are connected.

In addition, when doing computer graphics, one rarely need to be concerned about volume vs surface at the same time. For example, it is common to do volumetric and surface ray tracing separately and combine the result afterwards into final rendering. The intuition that you can not put both the volume and the surface of the same geometric object into the same set `**S**`, helps to develop intuition why we tend to separate those two concerns in practical programming.

A geometric piece does not need to have specific coordinates, nor do we need to specify in full how we address the piece. This model cares very little about such implementation details.

Nor do you need to code this specific model to make programming easier. It helps to use the model mentally, as a language tool to focus on your project from a perspective of Homotopy Theory. Still, this is only to make reasoning easier. The purpose is to get stuff done, not over-think your design.

Homotopy Theory often focuses on higher morphisms, but in this model, this is just more complex sub-expressions. We usually do not care that much about what particular complex sub-expressions the model contains. Instead, we tend to write code that works for all cases. Many people feel the need to learn about these higher levels of abstractions and connections between things, but in practice, it is about reasoning about the number of pieces and how they relate to global constraints that is most useful.

Finally, all connections to a single point, e.g. `a` exists only within a single n-groupoid. Therefore, it can be useful to think of the point as belonging to that n-groupoid. Not just the piece, but its points.