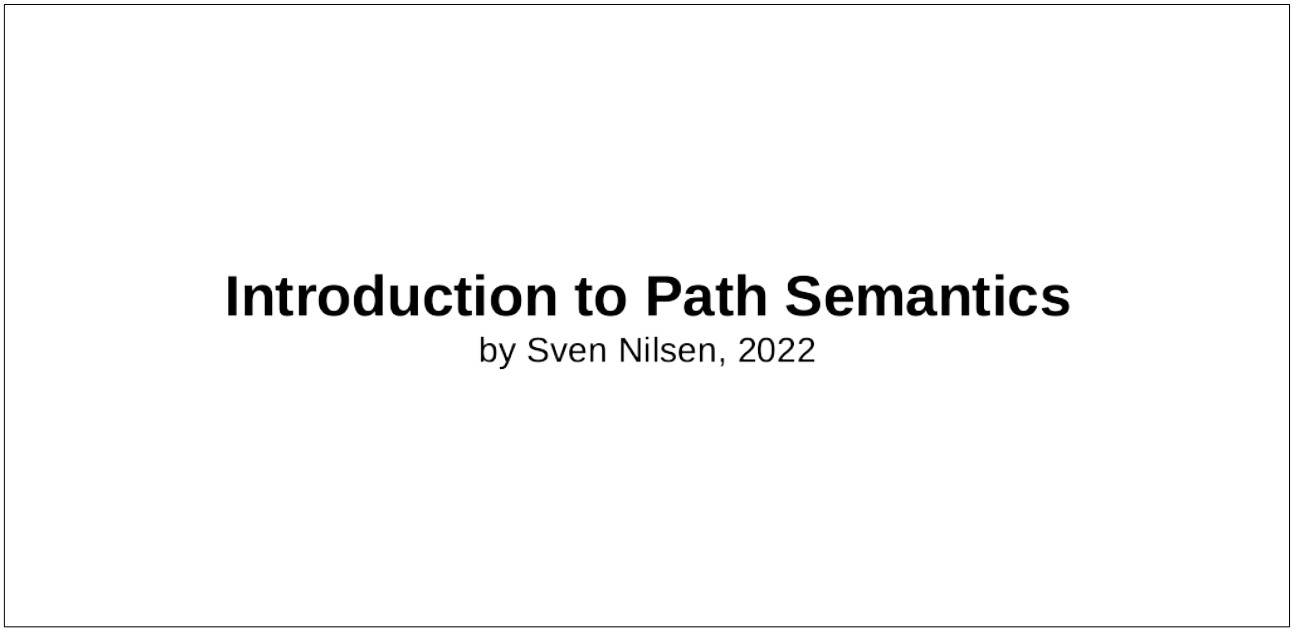
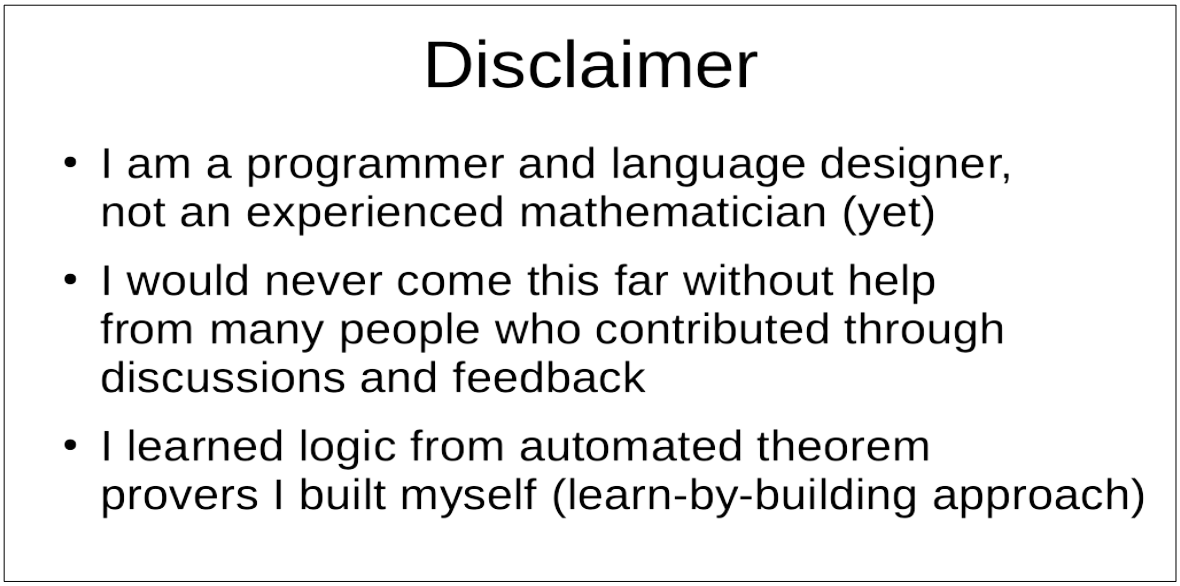
Transcript of Talk “Introduction to Path Semantics”

by Sven Nilsen, 2022

This talk was given to Category Theory Study Group on Applied Category Theory server (Discord).



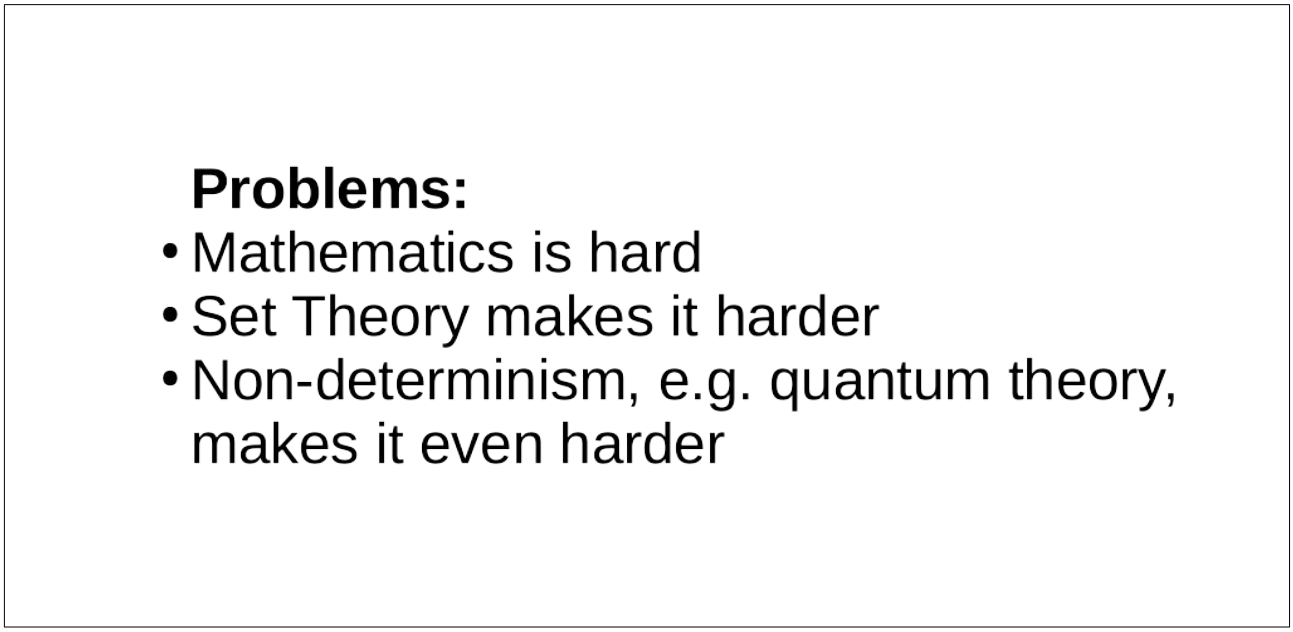
Welcome! I have been working 6 years on Path Semantics. The first thing people think is that the project is about exploring new math, but that is not actually the main purpose of the project. It is about language design.



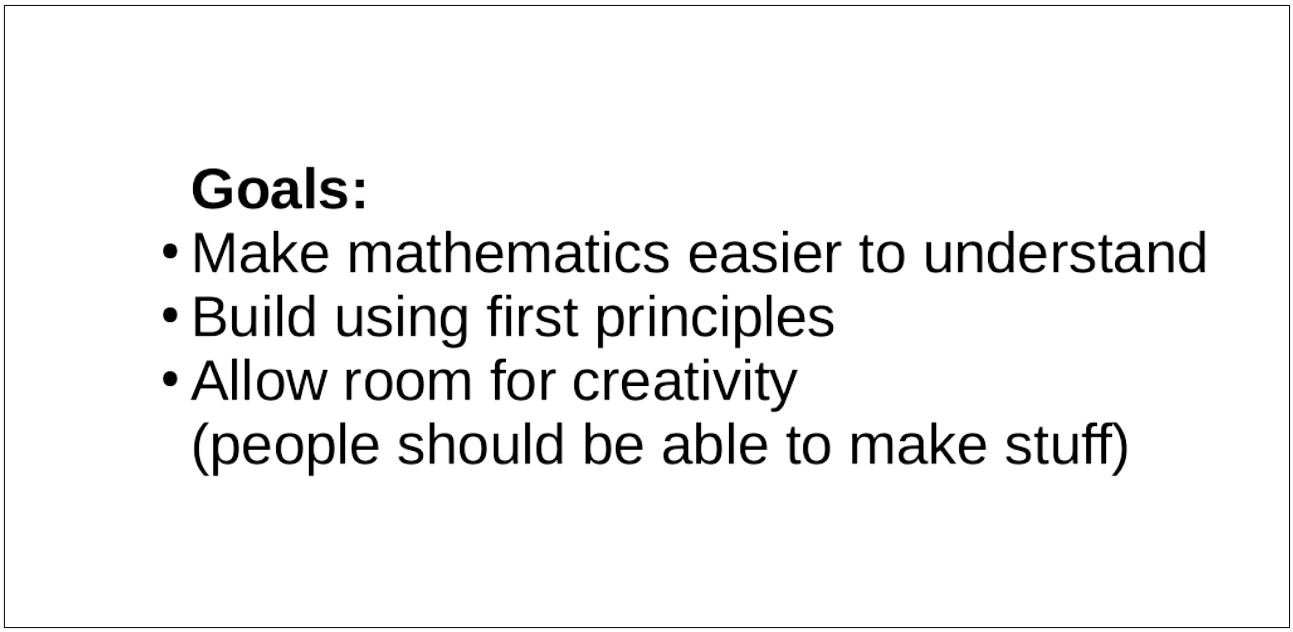
I am not an experienced mathematician, at least not yet. I am a programmer and language designer.

I have been discussing with people about this project and lots of people have contributed. I could never do this on my own without other people guiding me.

I learned logic from building automated theorem provers. I never took any logic classes. There might be stuff that you know that I do not know, because the things I am focusing on is how to design the language and make it work with automated theorem provers.



The problems are: 1) Mathematics is hard to learn. There are many people that tried to learned it and failed. 2) Set Theory makes mathematics harder. However, I saw an opportunity when Homotopy Type Theory was developed. We got a better understanding of mathematics. That is how I got into dependent typed first. I got inspired by Edwin Brady, the creator of Idris. I started the Piston game engine project. Path Semantics was a tool I used to develop some algorithms that saved me a lot of time. 3) There is also the problem, for example when you want to understand quantum theory. Quantum theory comes with a custom notation that is removed from the semantics side of it. I want to integrate so that people can learn what quantum theory means in terms of functions.

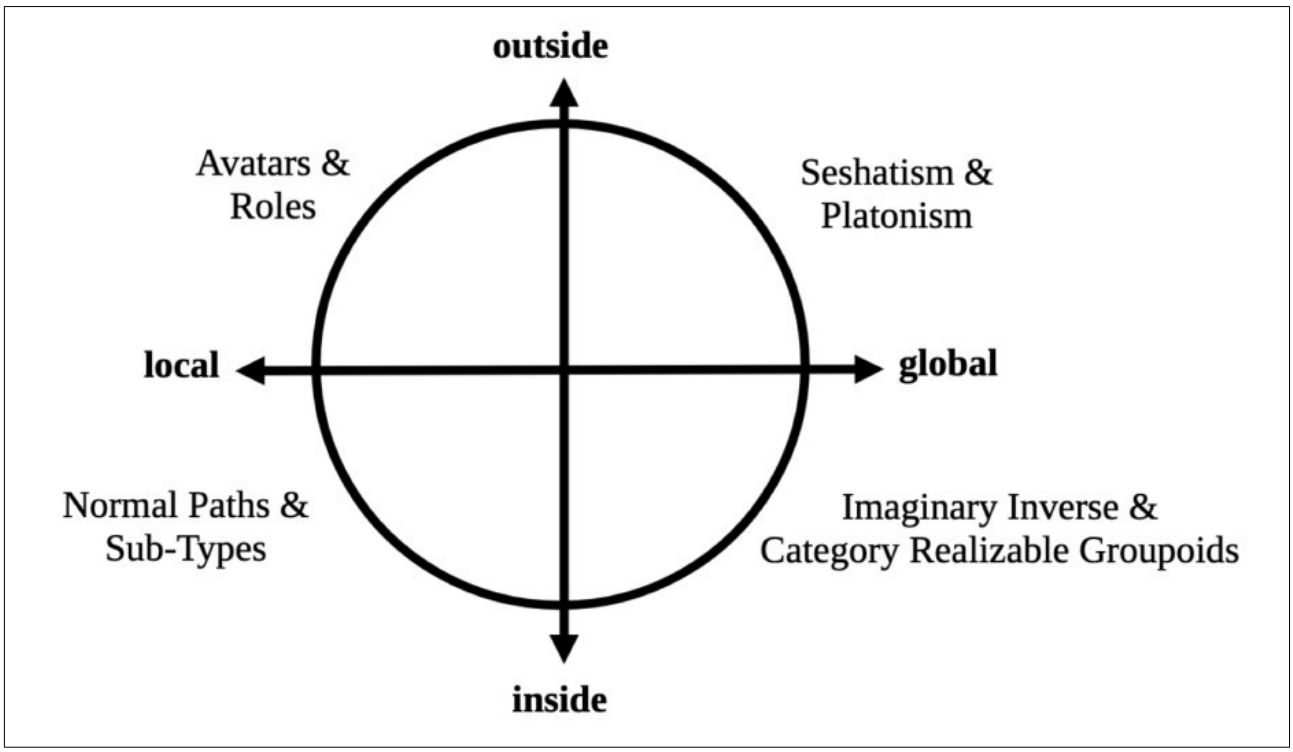


The goals are: 1) To make mathematics easier to understand. 2) Build using first principles. 3) Allow room for creativity. People should be able to make stuff.

Solutions:

- Focus on key obstacles to learning math
- Apply modern knowledge (CT, HoTT)
- Make everything extensible (develop new techniques)

The solutions are: 1) To focus on key obstacles to learning math. Instead of trying to address every problem, for example in the Wolfram Alpha project where people are trying to build this huge knowledge base, I try to focus on some simple areas of math that I can improve. 2) I want to apply modern knowledge, like Category Theory and Homotopy Type Theory. 3) I want to make everything extensible so people can build their own stuff. This requires some new techniques.



We created a diagram that is called the “Logi circle”. It consists of four parts. The x-axis here goes from Local to Global. The y-axis goes from Inside to Outside.

The Local to Global distinction is very common in mathematics, but the Inside to Outside might seem unfamiliar. It comes from philosophy. I worked with Kent Palmer to try to understand how these two sides are related.

I will use this diagram as an overview of this talk and I will go through each section in turn.

Normal Paths & Sub-Types
(Local, Inside)

Let’s start with “Normal Paths and Sub-Types”. This is the Local Inside.

```
> eval add[even](true, true)
true
```

We have a theorem prover that is called “Poi”, where we can write these expressions. It means when you add two even numbers, you get an even number. We can calculate with some statements that are a little more sophisticated than a normal calculator.

```
> eval concat[len](3, 4)
7
```

You can also do this with for example lengths of lists. If you concatenate a list of length 3 with a list of length of length 4, you get a list of length 7.

```
> eval concat[len][even](true, false)
false
```

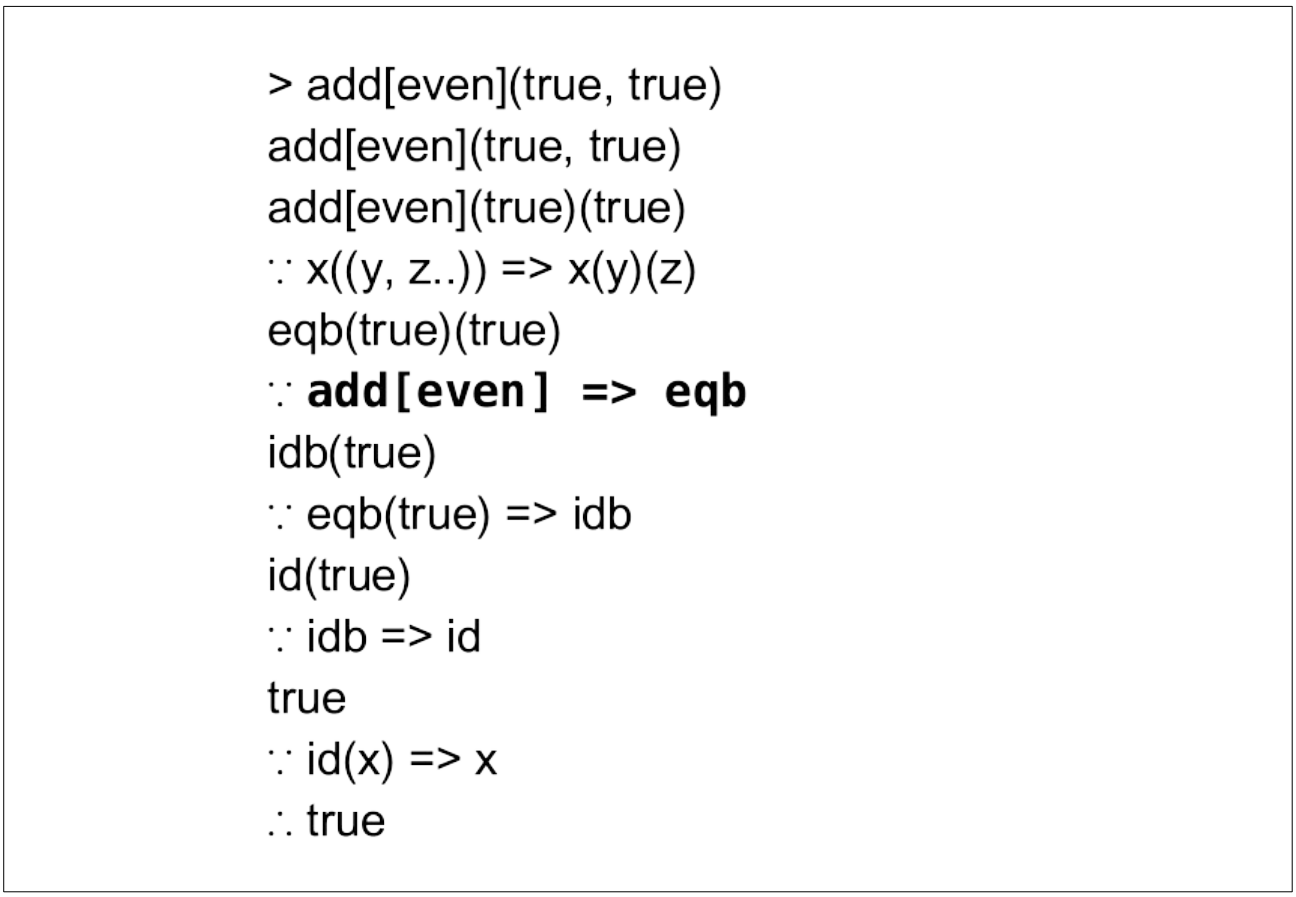
You can compose these expressions together. If you concatenate an even length list with an odd length list, you get an odd length list.

```
> eval mul_mat[det](2, 3)
6
```

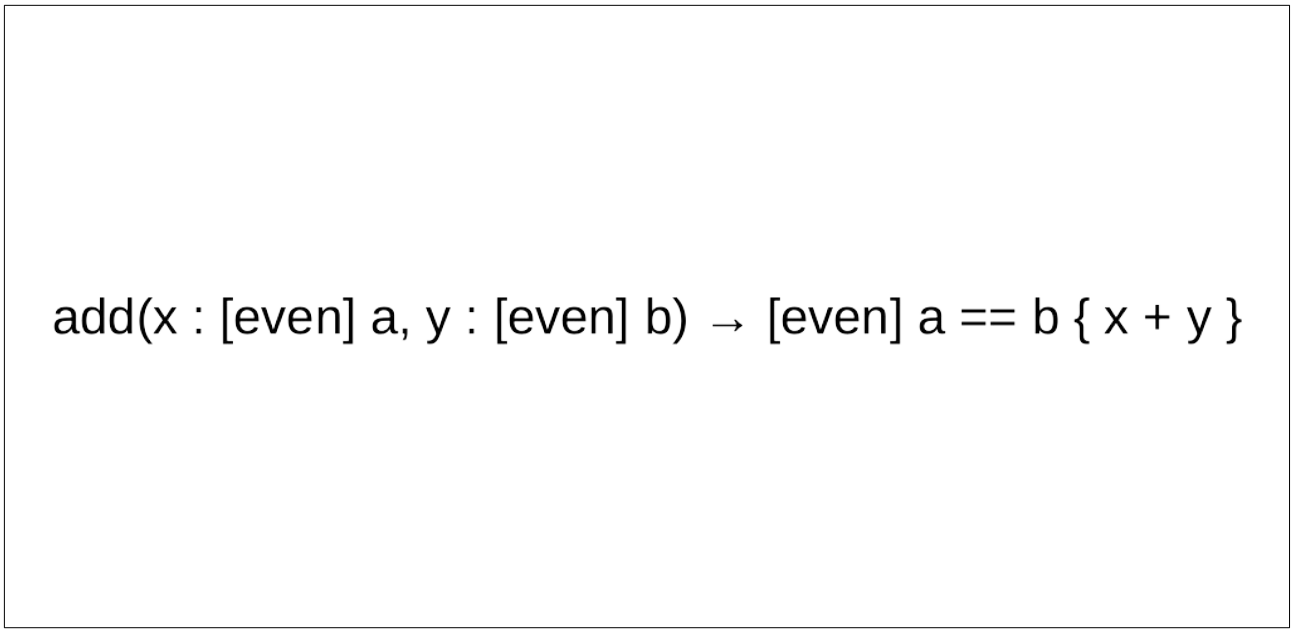
The purpose of these expressions is scaling to more complex problems. For example, when working with matrices, you often use determinants. When you multiply a matrix with determinant 2, with a matrix with determinant 3, you get a matrix with a determinant 6.

```
> eval add[even](true, true)
true
```

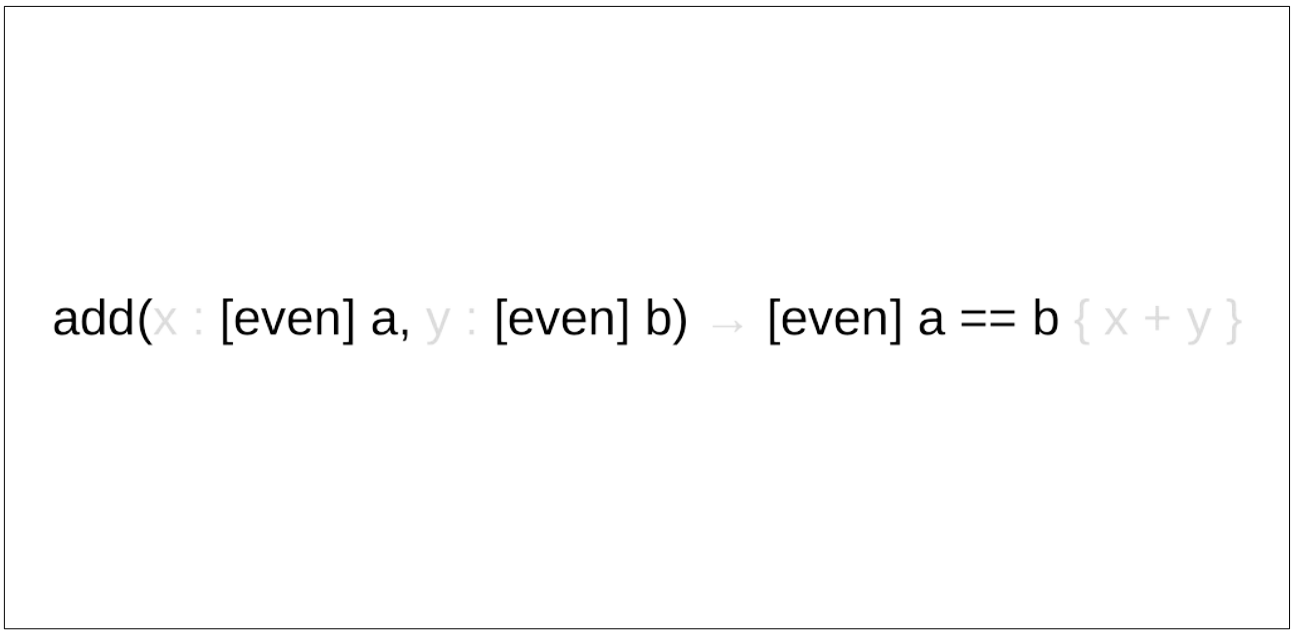
Let’s go back to the example when you add two even numbers.



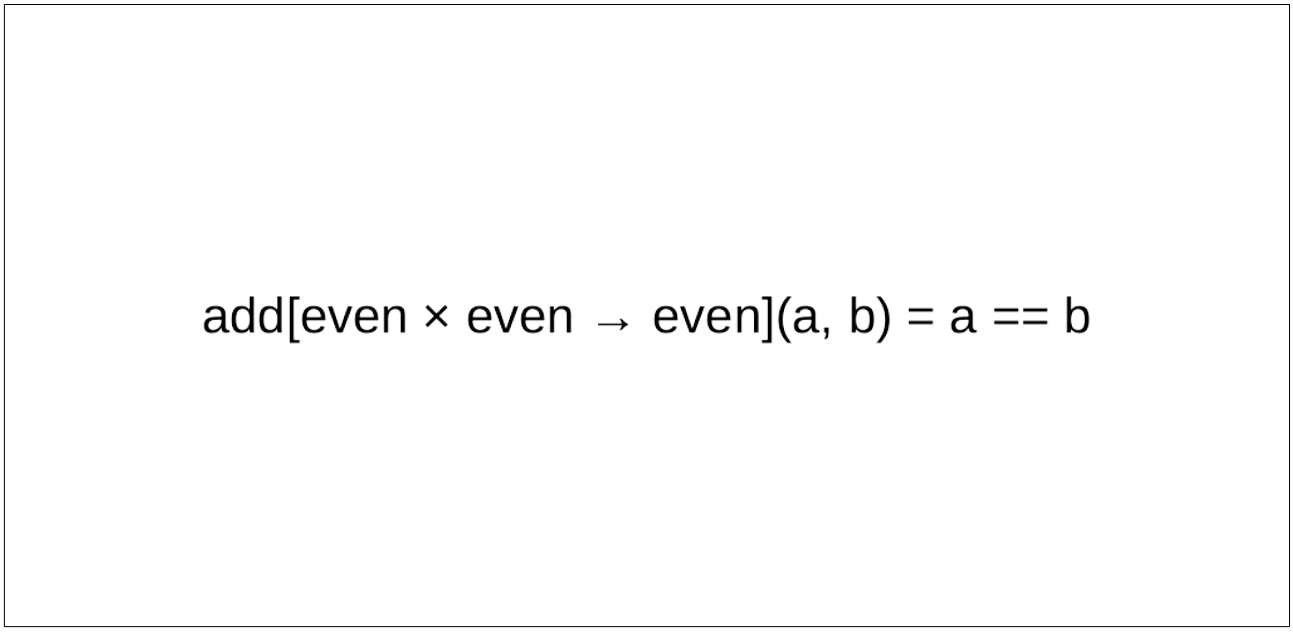
Looking at the proof, there is something strange about the statement in the proof where you transform the even property of addition into equality of booleans.



The reason we can do this I will show here. Let’s look at addition as a dependent typed function where the even properties are sub-types. This is something you can model in, for example, Coq or Idris.



Let’s focus on the type level.



We move in front of the parenthesis and erase the stuff that was grayed out. This transition is not possible to do in a dependent typed language. It is because these sub-types are computational and when you have these in the type system, the type theory becomes undecidable. That is why dependent types does not allow you to do this. However, you can use it in a point-free theorem prover, like Poi.

```
add[even](a, b) = a == b
```

I can simplify this syntax and just put “even” there since I use “even” on every argument and the output.

```
add[even] <=> \(a, b) = a == b
```

Then, I will isolate the naming from the definition…

add[even] <=> eqb

… and substitute the definition.

This is how I arrived at the rule that the theorem prover uses.

- Normal path: `add[even]`
  with constraints `add{(> 2), (> 5)}[even]`
- Sub-type: `x : [even] a`
  with sugar e.g. `[> 2] true <=> (> 2)`
- Multiple ways of expressing same stuff:
  `a : [f b] c, b : [f(a)] c, c : (= f(a, b))`
  Used to communicate focus

This syntax of using square brackets after the function with sub-types is called a “Normal Path”. You can also use constraints.

The syntax `x : [even] a` is a sub-type.

You can have syntax sugar, for example `(> 2)`.

There are also multiple ways of expressing the same stuff. For example, `a` can be given a sub-type, `b` can be given a sub-type and `c` can be given a sub-type. It is the same relation but there are different parts that you focus on. In the model it means the same stuff. However, this is a language intended to communicate mathematics. It is used to communicate focus.

Imaginary Inverse & Category Realizable Groupoids

(Global, Inside)

That was the first section of the Inside. Now we'll move on to the Global Inside. This requires introducing a new idea called the "Imaginary Inverse". I will talk about how this is used in Category Realizable Groupoids.

inv(g . f) <=> inv(f) . inv(g)

The Imaginary Inverse is a contra-variant functor. This means that when you take the inverse of `g` composed with `f` you get the inverse of `f` composed with the inverse of `g`.

$$f[g_1 \rightarrow g_2] <=> g_2 \,.\, f \,.\, inv(g_1)$$

Normal Paths can be rewritten using the imaginary inverse.

$$add[even] <=>$$
$$add[even \times even \rightarrow even] <=>$$
$$even \,.\, add \,.\, (inv(even) \,.\, fst, inv(even) \,.\, snd) <=>$$
$$eqb$$

This is how it works for the even property of addition.

Imaginary inverse can be used instead of syntax for normal paths, suitable for compilers and automated theorem provers

The Imaginary Inverse can be used instead of syntax for Normal Paths. It is not very readable, so it is suitable for compilers and automated theorem provers.

$$\forall x : obj(Set) \{ inv(x) := x \}$$
$$\forall f : A \to B \{ inv(f) := inv(f) \}$$
$$inv \,.\, inv <=> id$$

objects map to themselves
1-morphisms are just “wrapped” involutions

This is the definition where every objects maps to themselves and 1-morphisms are just “wrapped”. It is an involution so when you apply it twice you get the identity.

$$\text{inv} : (\text{Set} \to \text{Set}^{\text{op}}) \mid (\text{Set}^{\text{op}} \to \text{Set})$$

In the category of small categories the Imaginary Inverse has the type `(Set → Set^op) | (Set^op → Set)`.

$$\text{inv}\{C : \text{Cat}\} : C \to C^{\text{op}} \mid C^{\text{op}} \to C$$

We can generalize it to any category…

$$inv\{C : Cat\} : C \rightarrow C^{op} \mid C^{op} \rightarrow C$$

… but you want to get rid of this stuff…

$$inv\{C : Cat\} : C \rightarrow C^{op}$$
$$\because (C^{op})^{op} <=> C$$

… because the opposite of the opposite is the category.

inv <=> (^ op)

So, in the global perspective of the Imaginary Inverse, it is just taking the opposite of the category.

Solutions:
Forgetful on sub-category of `Set`

inv(f) => g

$Set^{op} \rightarrow (Set \mid Set^{op})$

There is something that is not so trivial that when you talk about solutions. For example, when you have the inverse of `f` and this normalizes to `g`, then you map from `$Set^{op}$` into `Set`, or if you don't have a solution you have to return the same input. So, you map also to `$Set^{op}$`. We don't like this property because it is an asymmetry.

Solutions & Problems:
2-morphisms on `Set | Set$^{op}$`

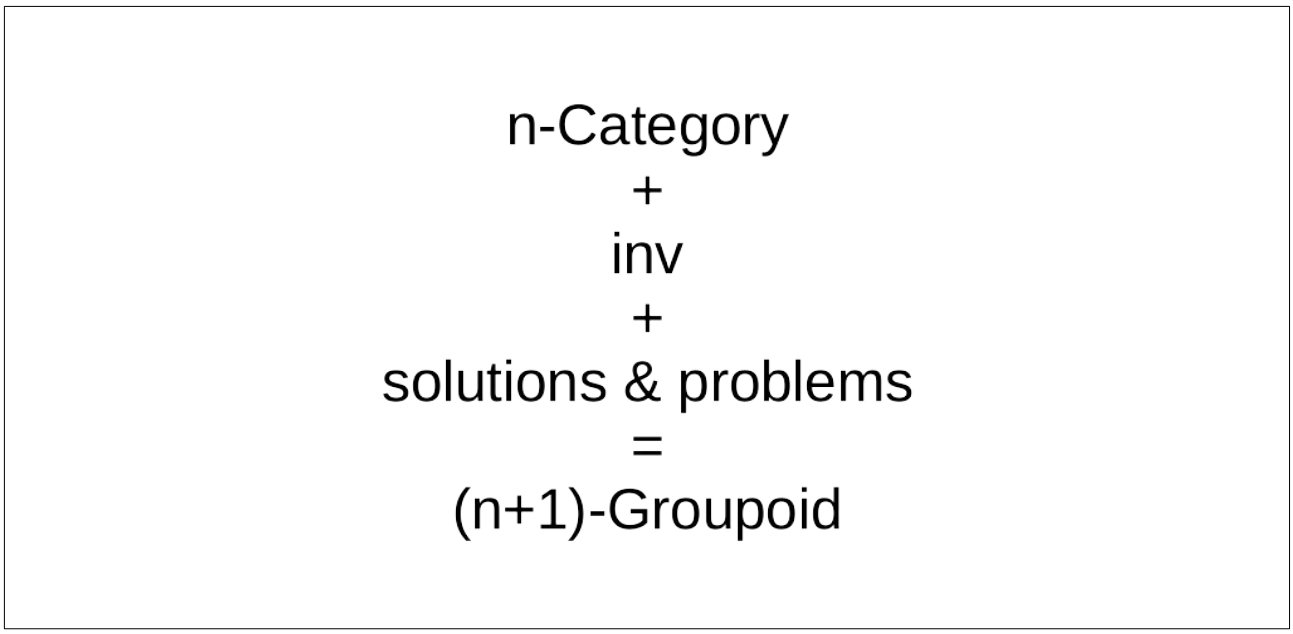
inv(f) <=> g

(Set | Set^{op}) ↔ (Set | Set^{op})

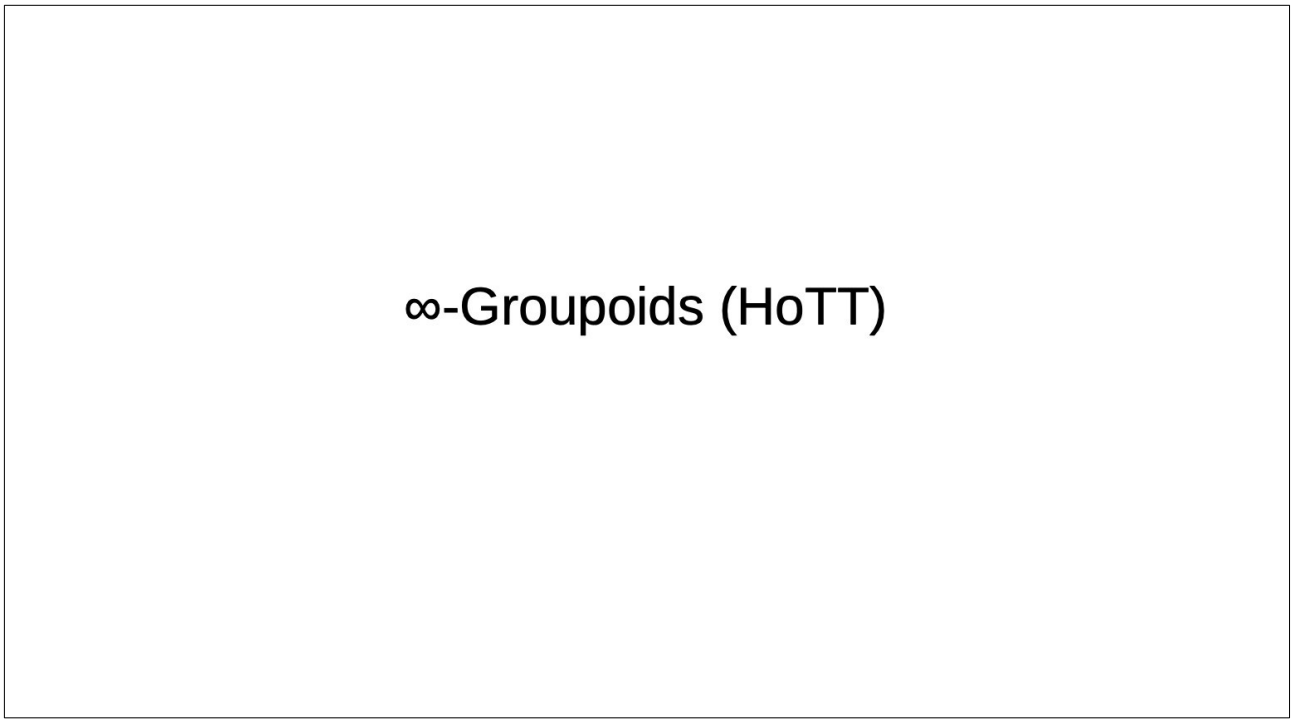
The way to fix that is by adding problems, for example, when you have a solution you can reintroduce the problem as an inverse of something. This gives you 2-morphisms on `Set | Set$^{op}$`.

1-Category
+
inv
+
solutions & problems
=
2-Groupoid

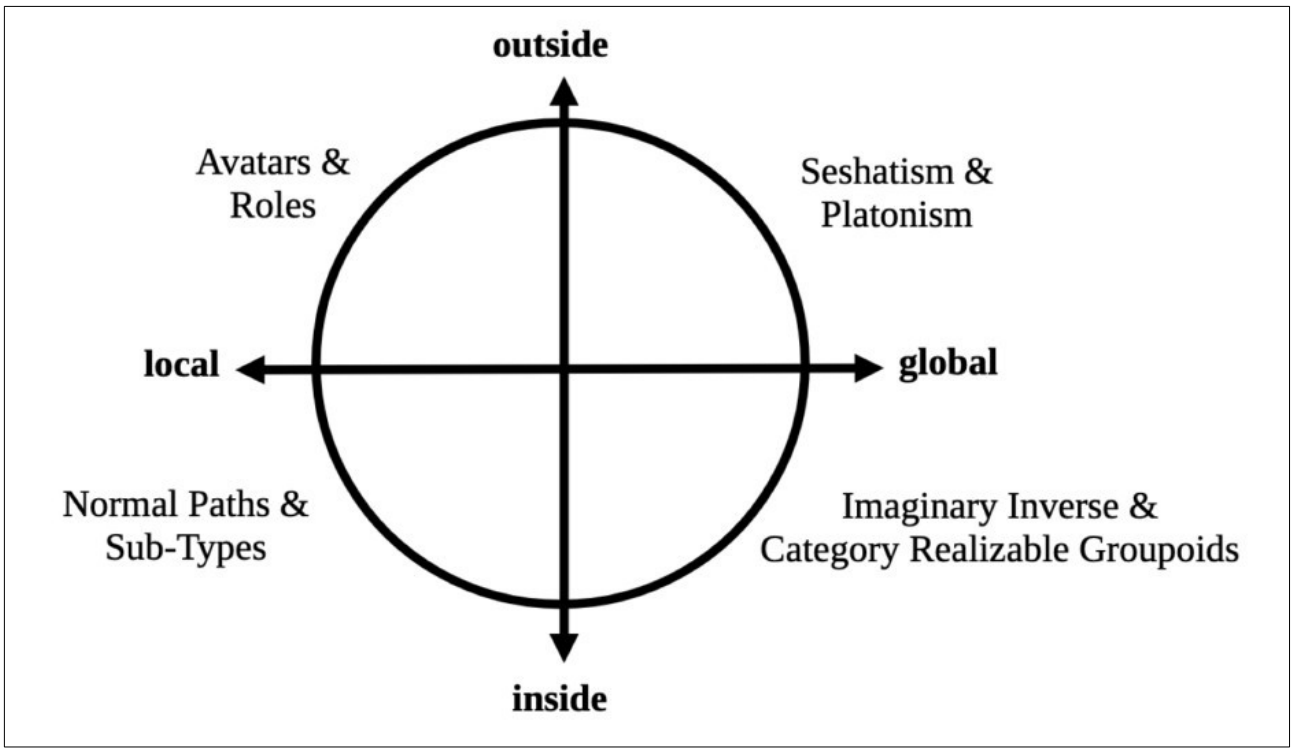
We started with a 1-Category, we added the Imaginary Inverse and we added Solutions & Problems. What we get is a 2-Groupoid.



We can do this for any category, so we start with an n-Category, we add the Imaginary Inverse and Solutions & Problem. We get an (n+1)-Groupoid.



By induction this gives us ∞-Groupoids, which are models for Homotopy Type Theory.



Now we have looked at the Inside of the Logi circle. We talked about Normal Paths and Sub-Types, the Imaginary Inverse and Category Realizable Groupoids. Now we will move to the Outside and this is where things get more strange.

Inside Philosophy: Seshatic Platonism

Symbols: Local
Semantics: Global

(<= 2)-Categories: Inside
(> 2)-Categories: Outside

First, let's think about the philosophy of the Inside. This philosophy is called "Seshatic Platonism". I will explain later what this means. In Seshatic Platonism, symbols are Local and semantics is Global. If you have a category that is less or equal to 2, then this is Inside. If you have more, then this is Outside.

Outside Philosophy: Platonic Seshatism

Inside: An external object is modeled as an unknown sentence in some language

Outside: There exists at least one symbol in some language which does not refer to its theory

The Outside philosophy is called “Platonic Seshatism”. Notice that in the Inside we have Seshatic Platonism and in the Outside we have Platonic Seshatism. Symbols and semantics are the same in both the Inside and the Outside. However, the Inside has a different definition in the Outside philosophy.

The Inside means that an external object is modeled as unknown sentence in some language. For example, if you have phenomena in the world, then you try to model it as some mathematical object. When you have finished that sentence, you have a perfect model of that phenomena, so you no longer need to observe the world. You can simulate it perfectly. This is a property of Inside languages.

In the Outside there exists at least one symbol in some language which does not refer to its theory. This means that you can’t complete this transition into a model. You have to keep observing or interacting. This makes the Outside very different from a philosophical perspective than the Inside.

Avatars & Roles

(Local, Outside)

In the Local Outside we have something called “Avatars & Roles”.

$$p(a, b) \rightarrow b : p \rightarrow p(a) = b$$

$$p(a, q'(b)) \rightarrow q'(b) : p \rightarrow p(a) = \{q'(\_)\} \in q'(b)$$

This is a new logic, called “Avatar Logic”. It has two axioms.

Alice : person

parent’(Alice) : mom

For example, Alice has a role Person. However, Alice as a parent has the role Mom.

parent’(Alice) : mom

1-avatar: `parent`
role: `mom`

The 1-avatar is Parent and the role is Mom.

(Bob, parent’(Alice)) &
parent’(Alice) : mom
=>
mom(Bob) => parent’(Alice)

”Bob has a mom which is Alice as a parent”

If Bob has a relation to Alice as a Parent and Alice as a Parent has the role Mom, then Bob has a Mom which is Alice as a Parent.

(Bob, parent'(Alice)) &
parent'(Alice) : mom
& uniq parent =>
mom(Bob) = parent'(Alice)

"Bob's mom is Alice as a parent"

However, if Parent is unique, then Bob's Mom is Alice as a Parent.

Notice here I have a "has" relation (on the previous slide) and here I have an "is" relation.

(true, bool)
bool : type
=>
type(true) = bool

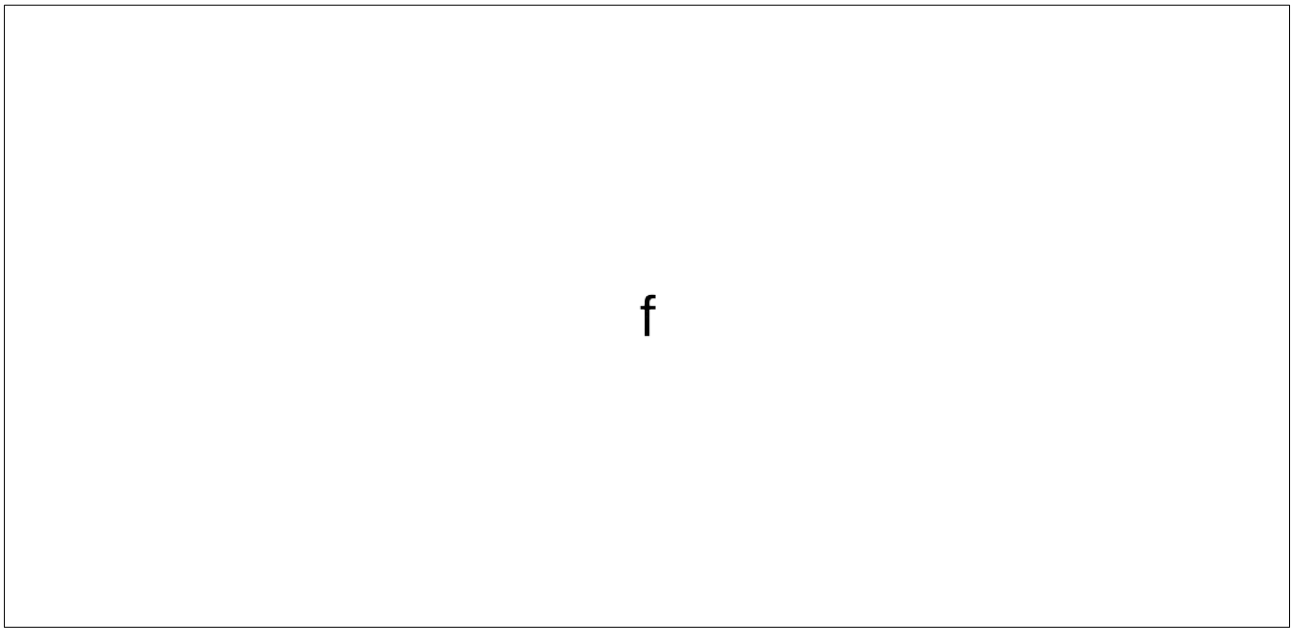
You can use this for simpler stuff, for example, True has a relation to Boolean and Boolean is a Type. This implies that the Type of True is Boolean.

- Introduce 1-avatars to solve over-constrained problems
- Build “larger theories” by relating ”smaller theories” by symmetries
- Alternative to first-order-logic for graph-like problems

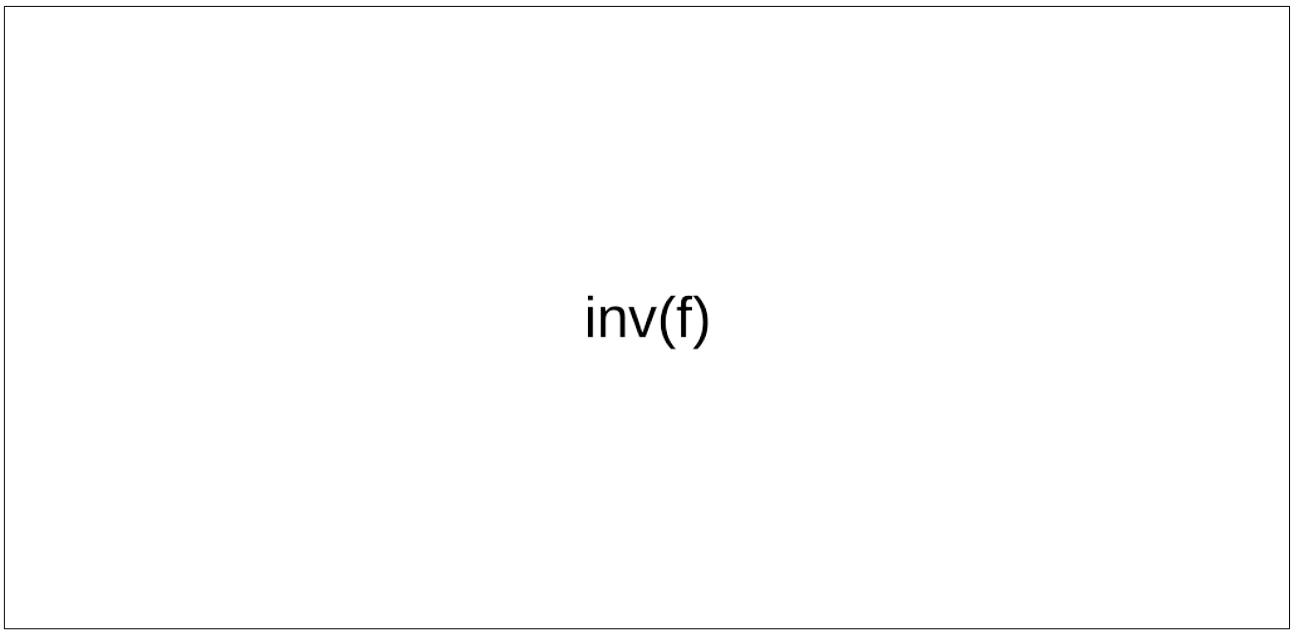
The way we use this logic is to introduce 1-avatars to solve over-constrained problems.

You build larger theories by relating smaller theories by symmetries. This is something you might recognize from Group Theory. The difference between this way of doing it and Group Theory is that in Group Theory you can’t solve over-constrained problems. You have to solve them within each sub-group. Here, you solve them by introducing new 1-avatars.

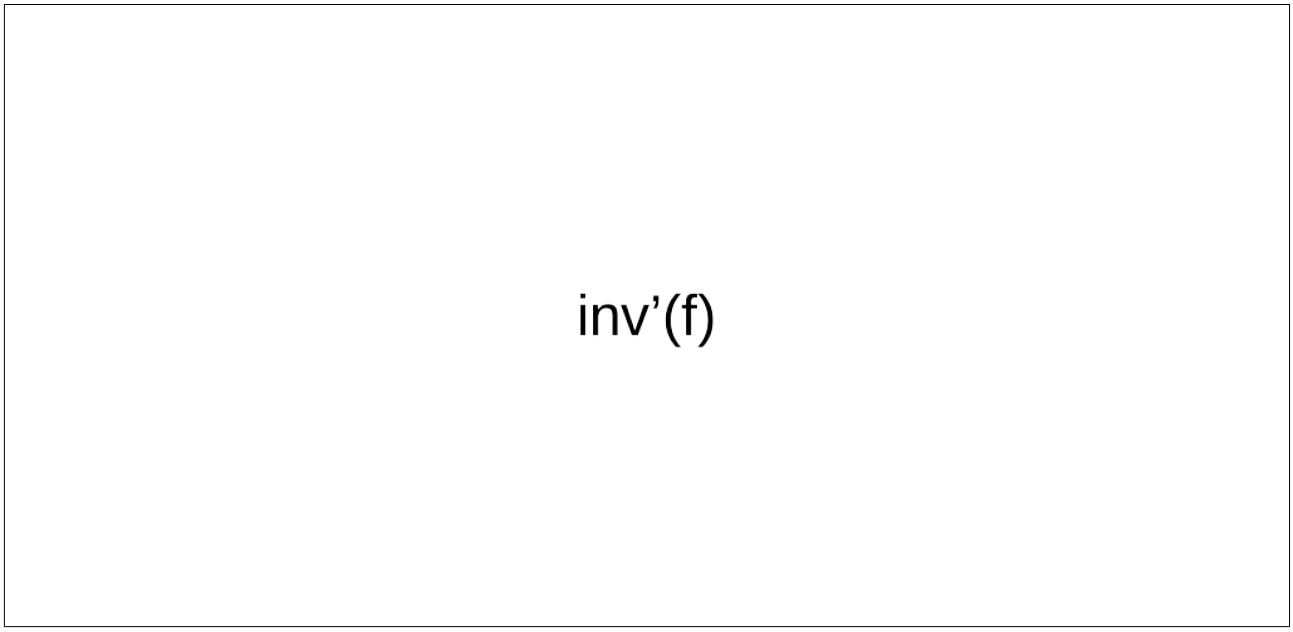
It is an alternative to first-order-logic for graph-like problems.



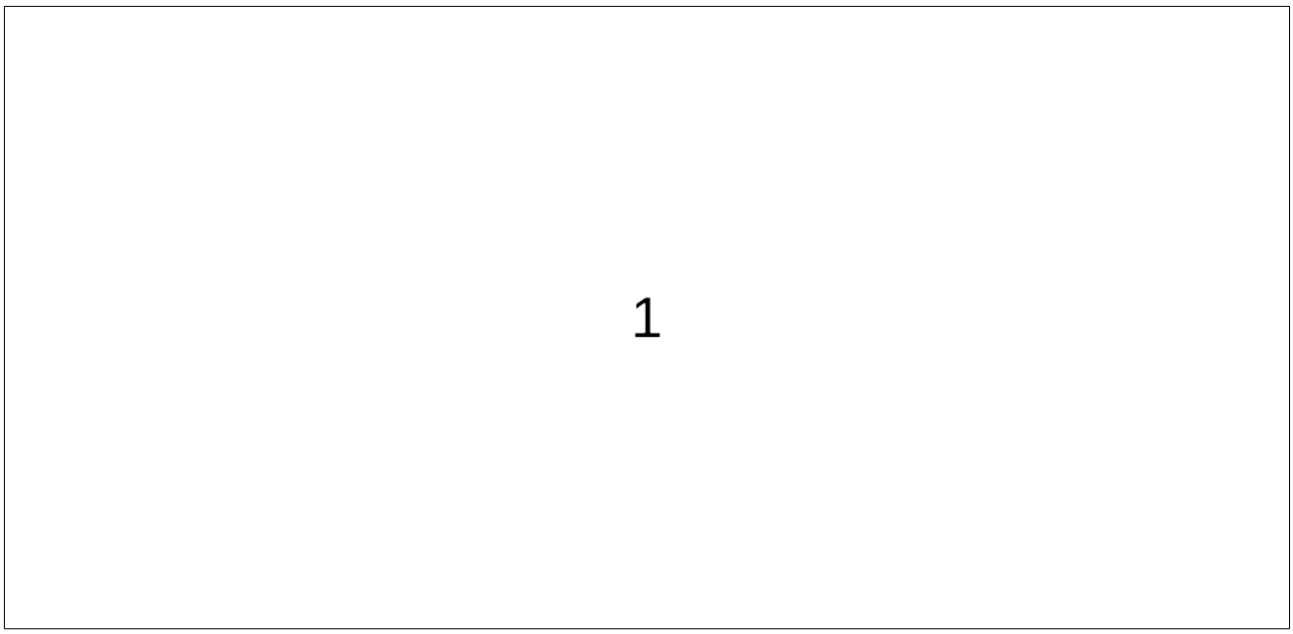
Let’s look, for example, at a function `f`.



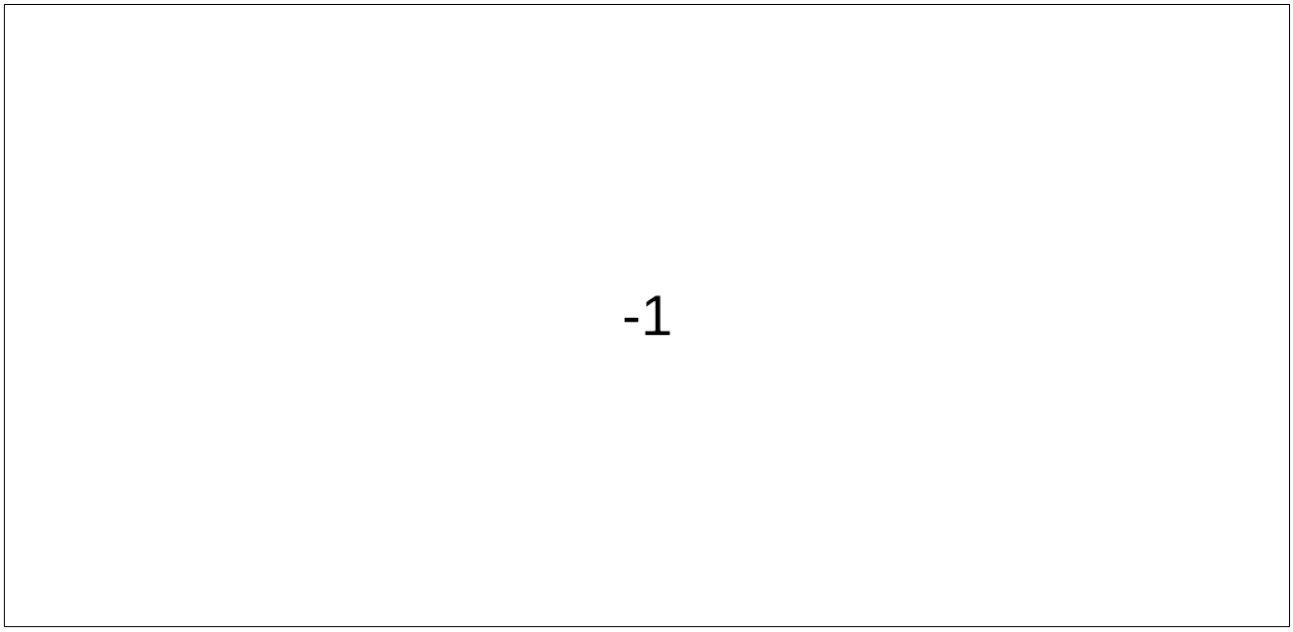
You have the inverse of `f`.



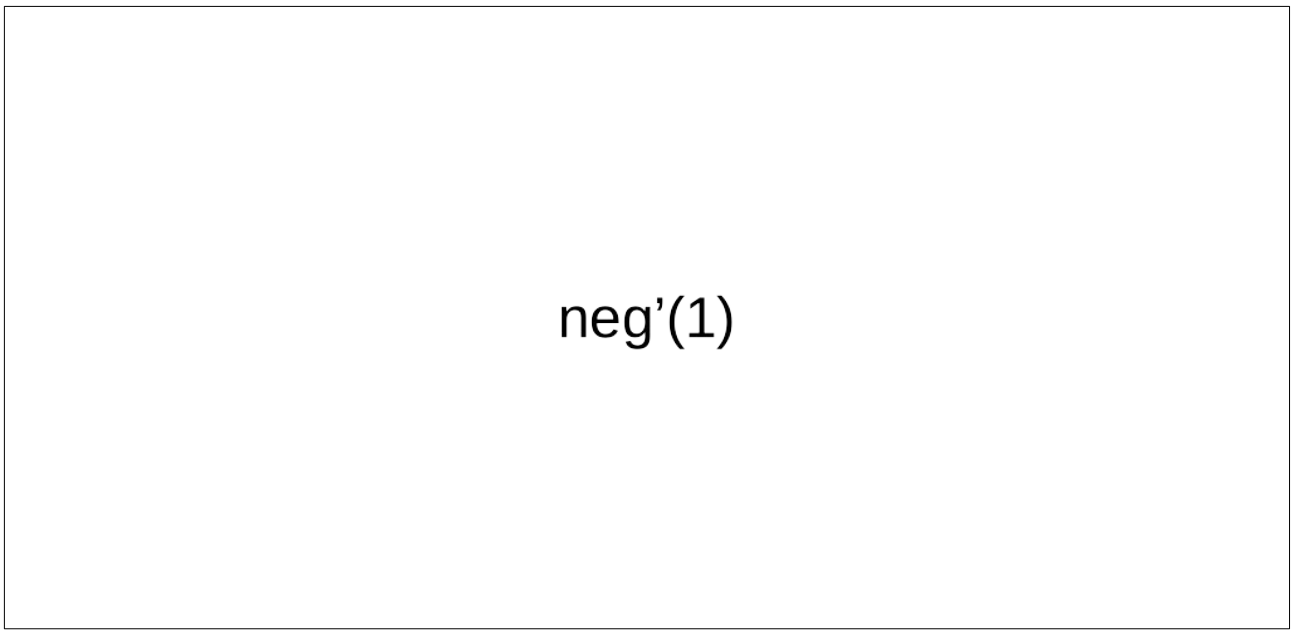
We can think about this as the imaginary inverse being a 1-avatar.



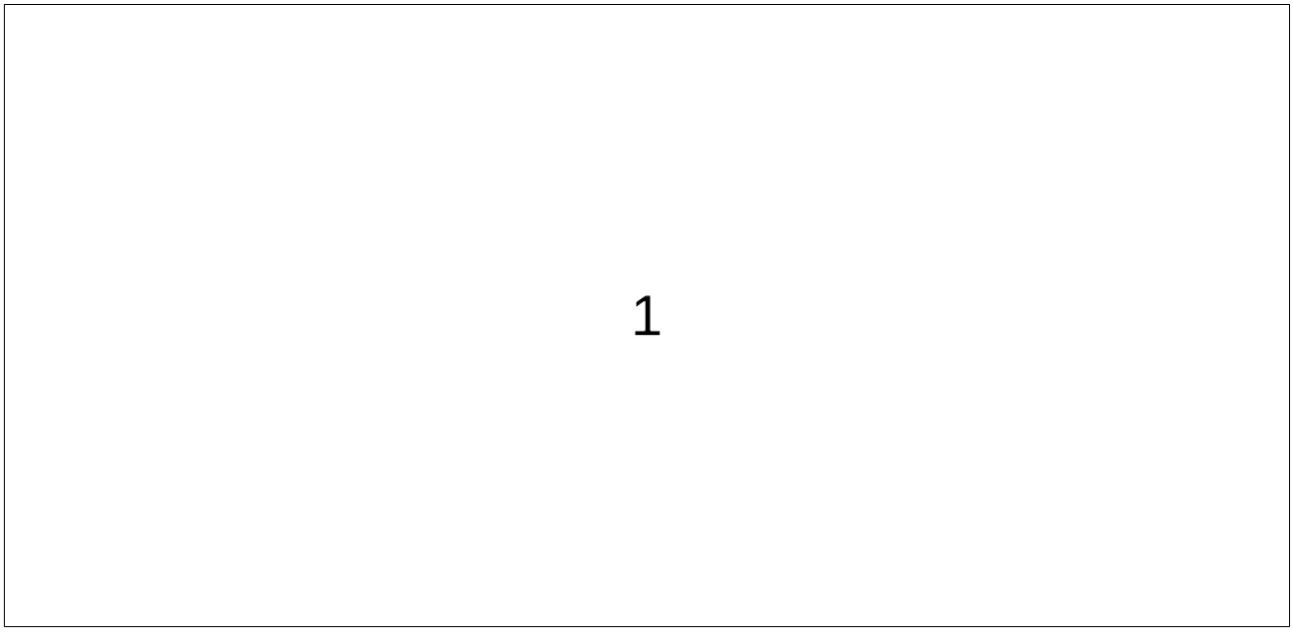
Let’s look at `1`.



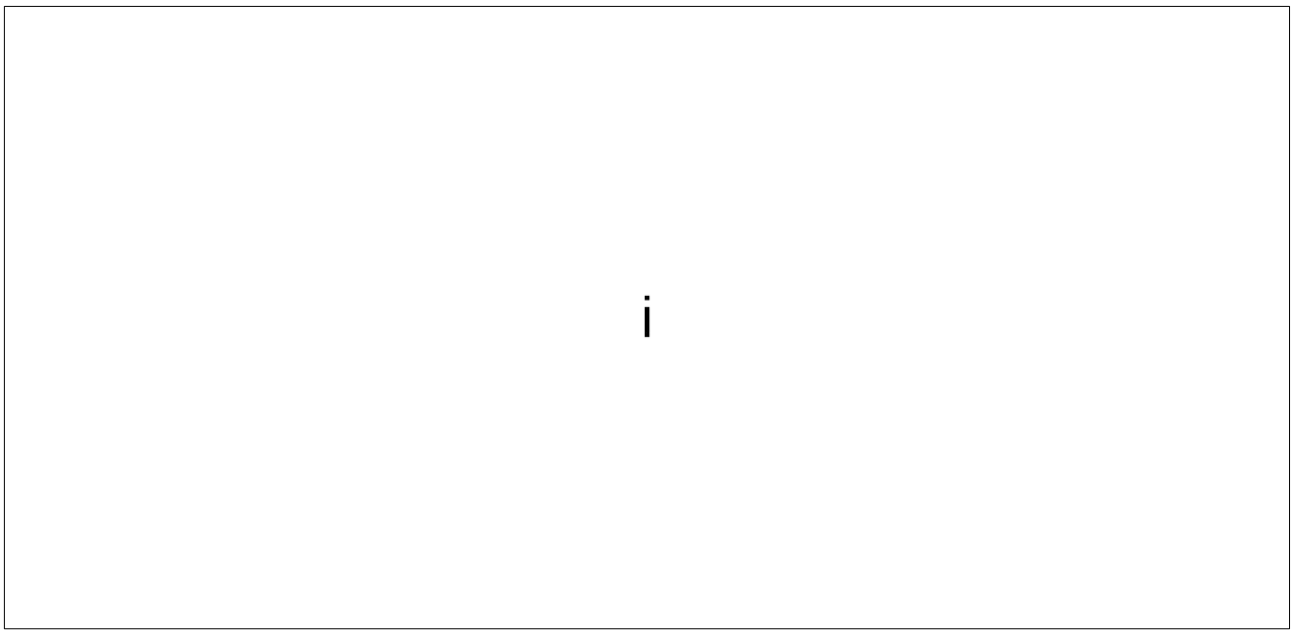
`-1`.



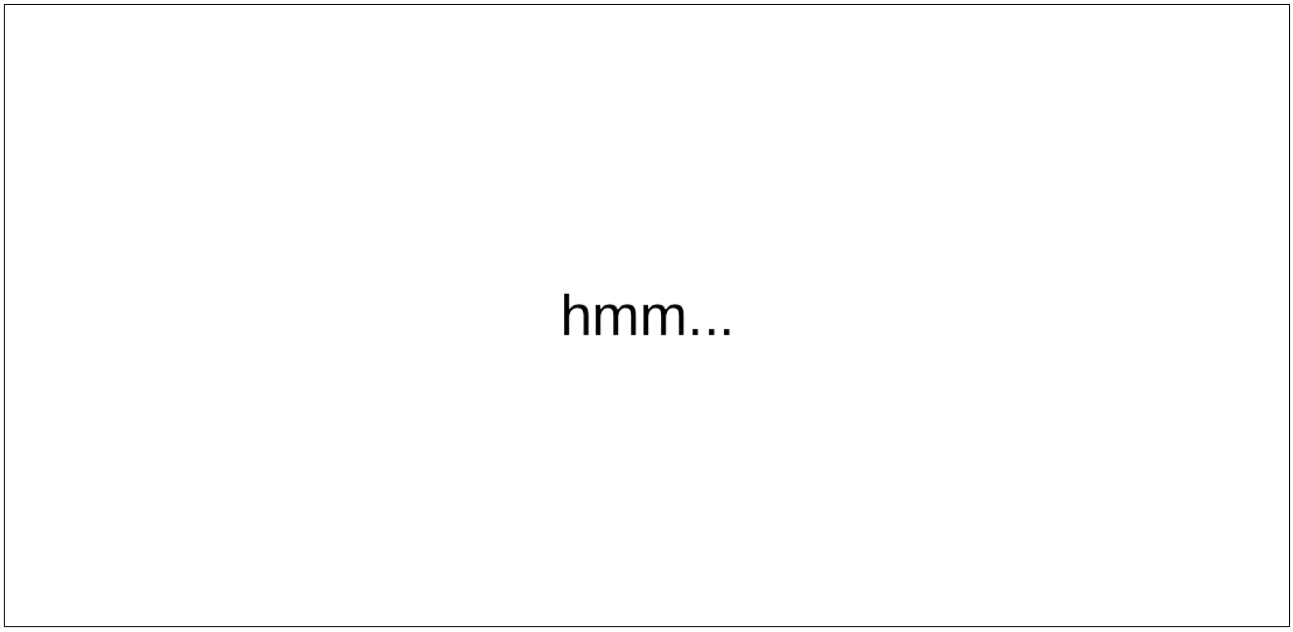
`-1` can be thought of as a 1-avatar using negation of `1`.



Let’s look at `1` again.



We have the imaginary unit.



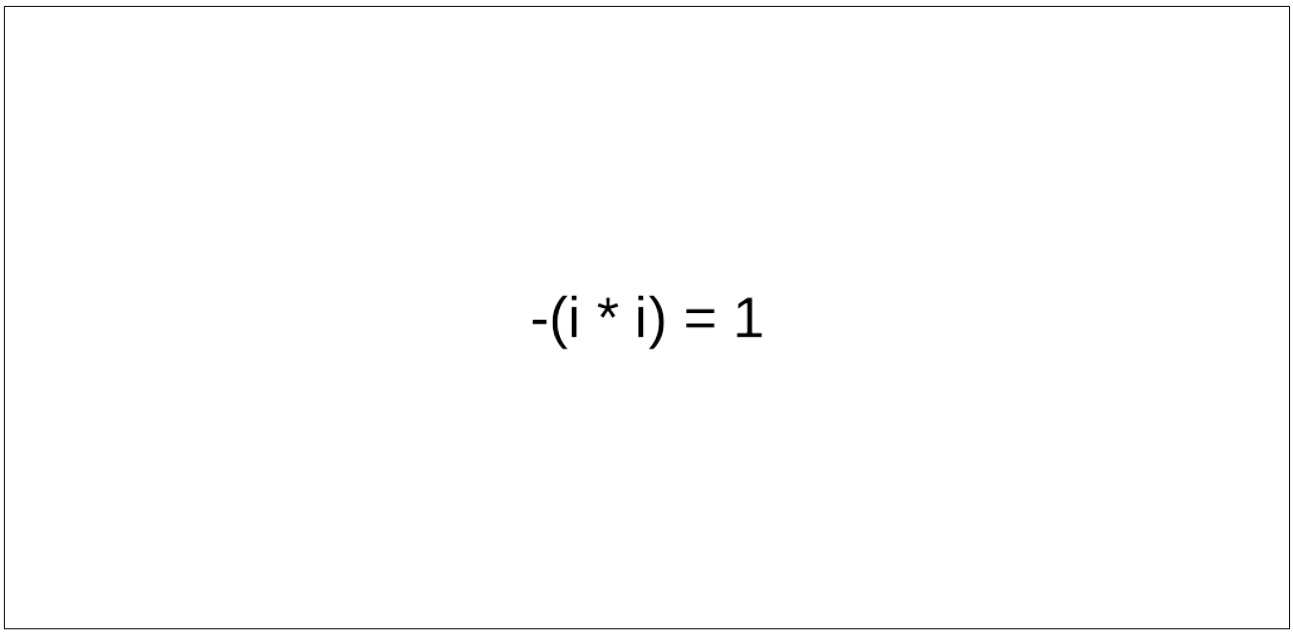
However, the imaginary unit has some more complexity than normal 1-avatars.

$$i^2 = -1$$

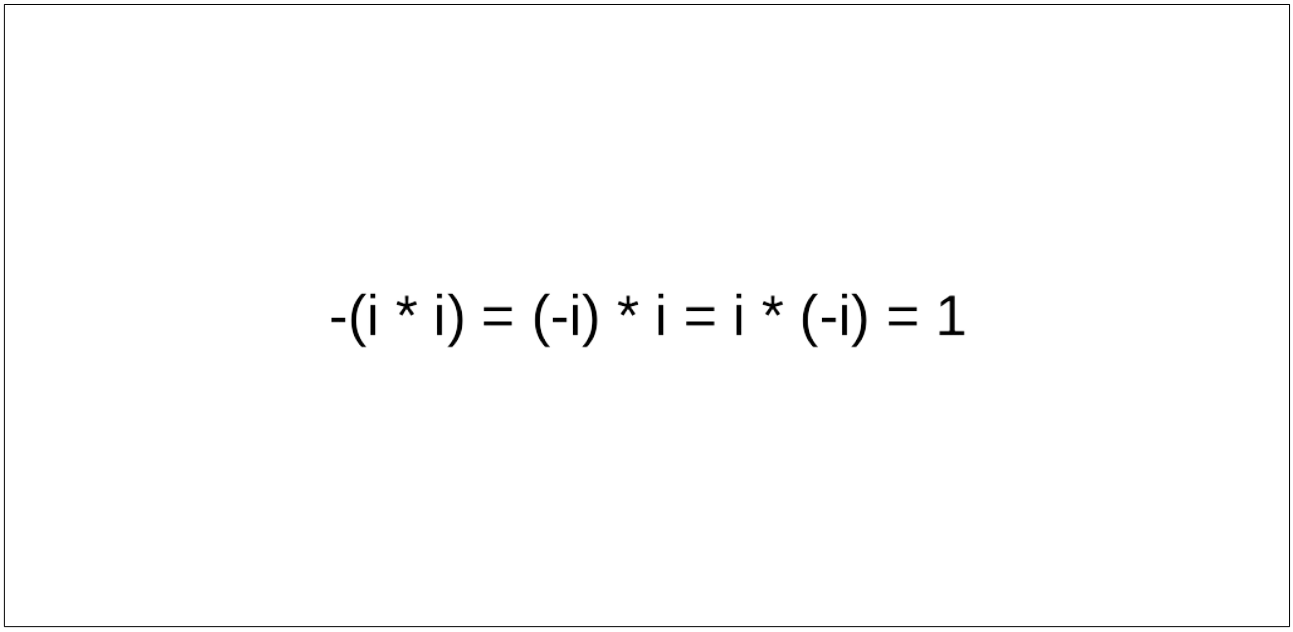
Let’s look at the definition.

$$i * i = -1$$

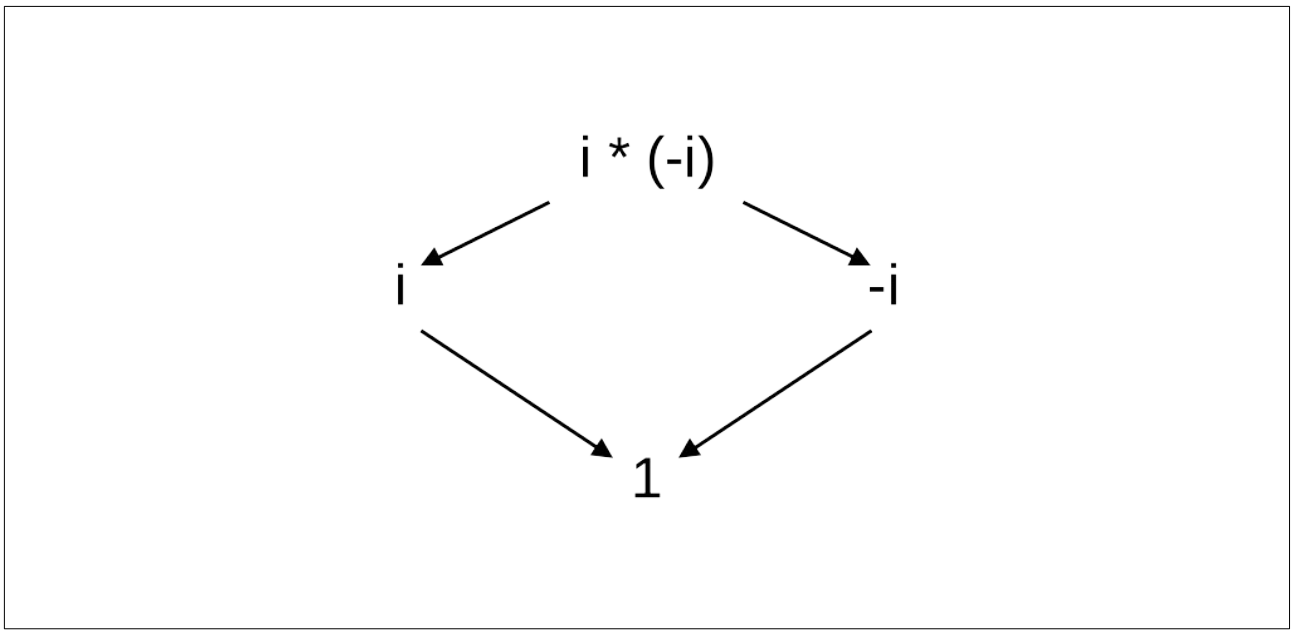
Expand the product.



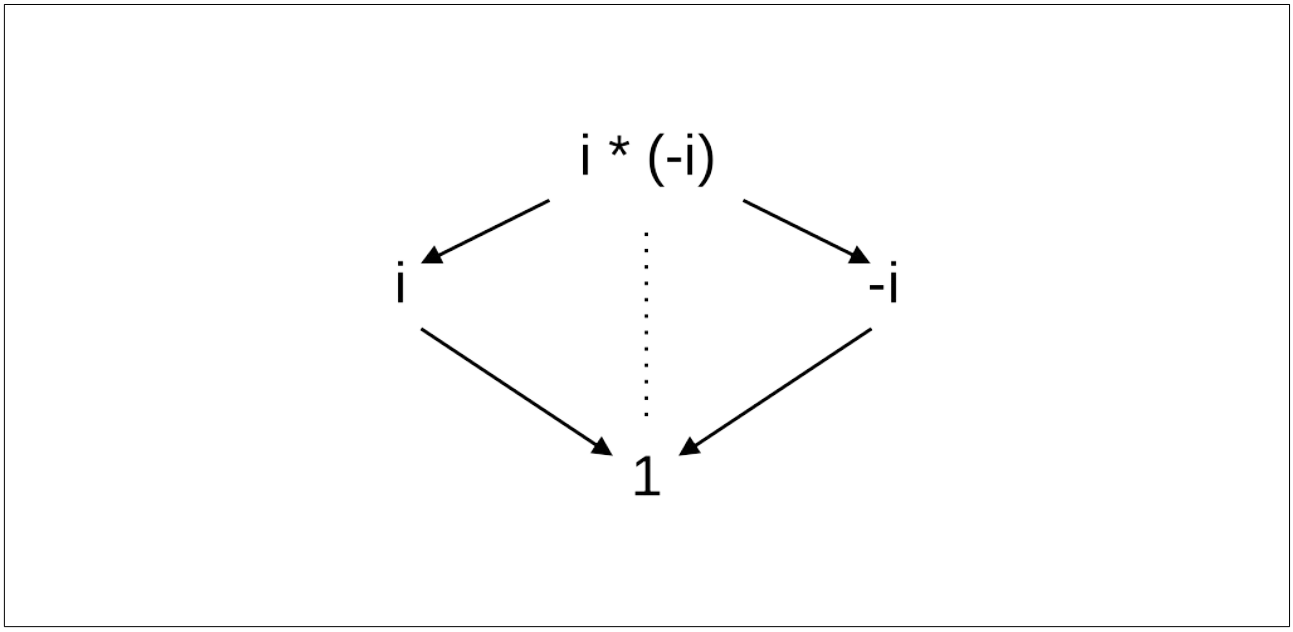
Move the sign over.



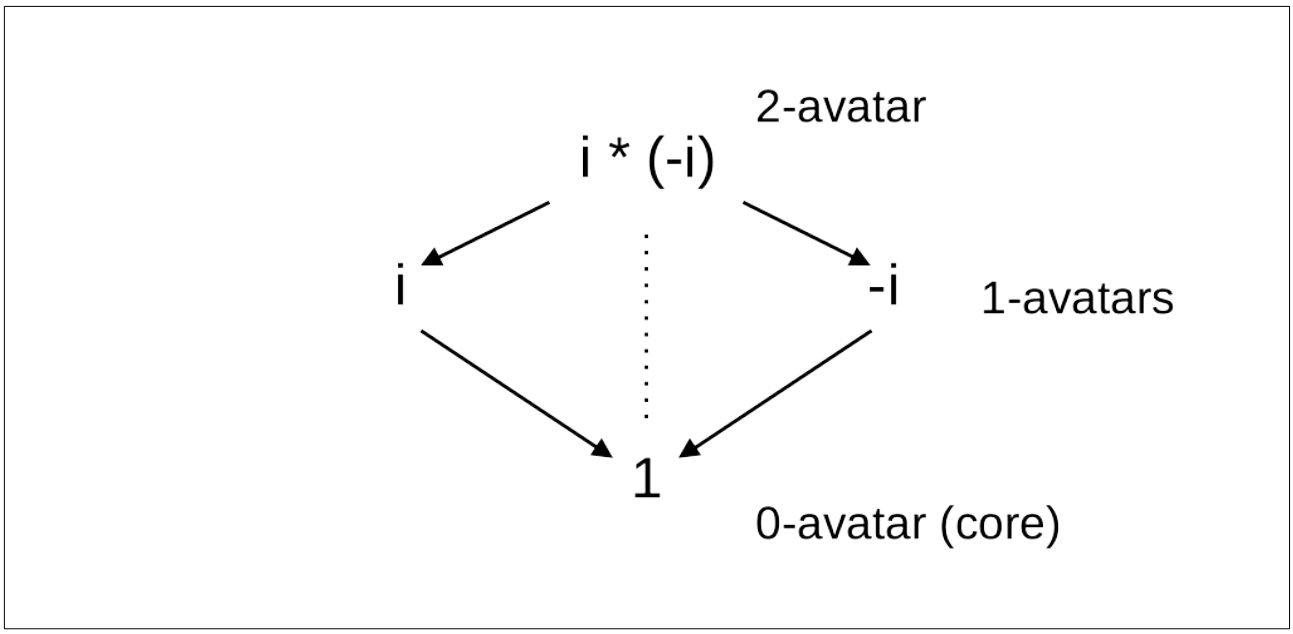
Look at the adjoint definition.



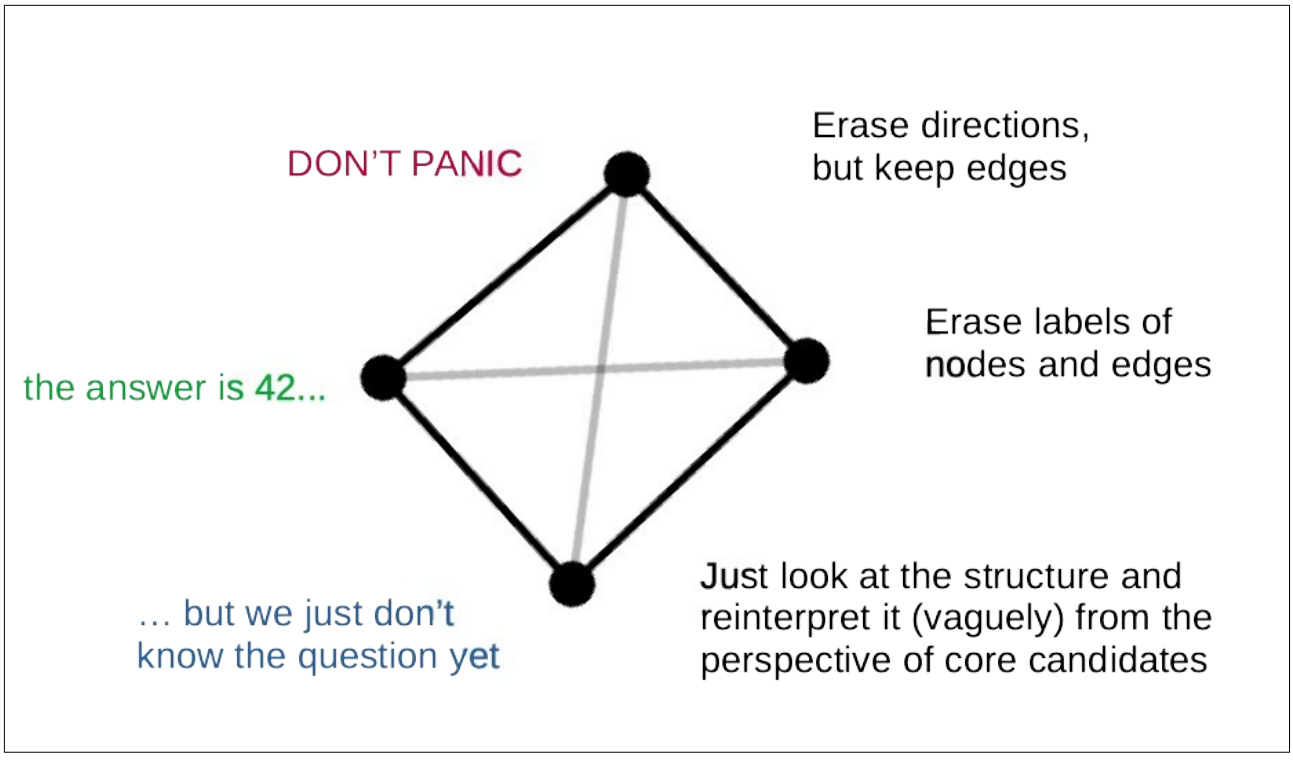
We draw this as a graph. There are two elements of this product that are 1-avatars of `1`.



The product is related to `1`.



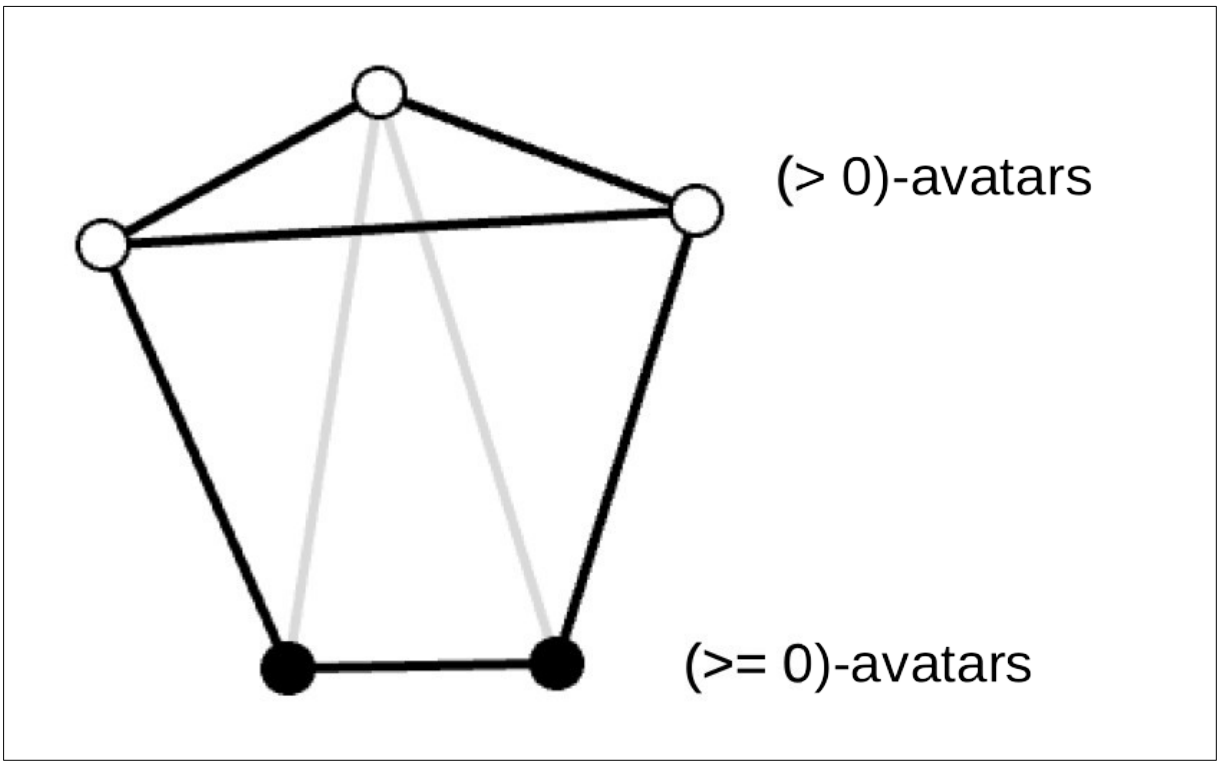
The product is called a “2-avatar”. The imaginary unit and its negation are called 1-avatars. `1` is called the 0-avatar, or core. So, a 0-avatar is the same as a core.



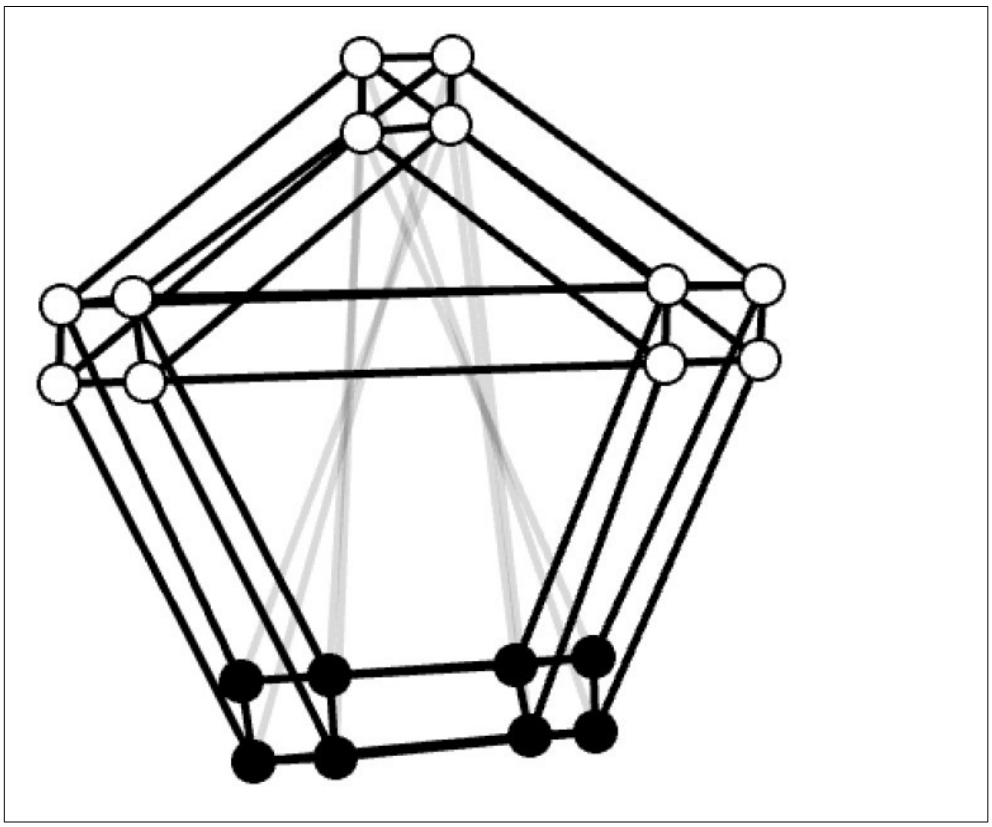
Now, we do a trick. We erase most of the information but we keep the edges. We erase labels of nodes and edges. We reinterpret the structure vaguely from the perspective of core candidates.

There are two overlapping dashed edges here because either corner can be a core. So, we have four dashed edges but two of them are overlapping.

There is a joke perspective here from the “Hitchhiker's Guide to the Galaxy”. It’s like, DON’T PANIC, the answer is 42, we just don’t know the question yet. So, even though we are making this transition into these graphs that have this complex aligned semantics, we don’t actually know how this structure relates to problems. However, these graphs have very interesting mathematical properties.



We can do this for any graph. There are some nodes that can’t be cores. Those are (> 0)-avatars. There are those who can, that are (>= 0)-avatars. We label nodes that can’t be cores with white and those who can with black. The gray edges are the relations between the cores and their highest n-avatars.

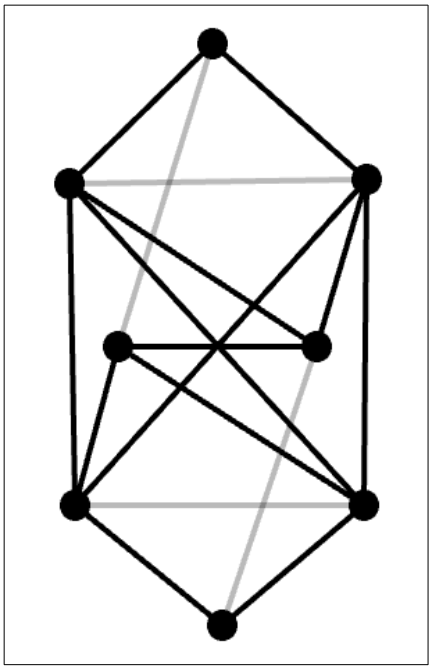


When we take the graph product of two such graphs, the product inherits the properties in clusters of nodes from the sub-products.

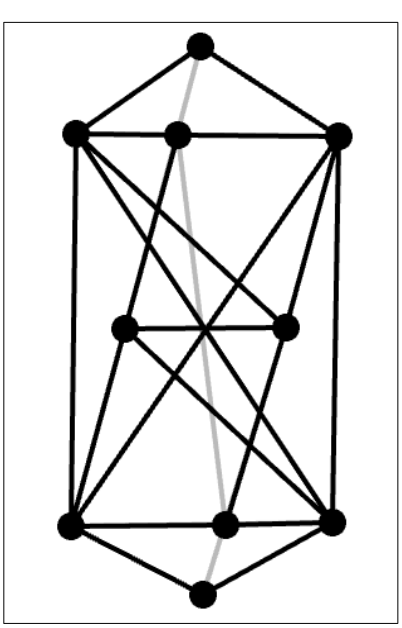
Cartesian combinatorics (stair-pair)
+
Möbius topologies (semi-contractibility)
=>
Outside foundation of natural numbers
(HPPs - Harry Potter Patterns)

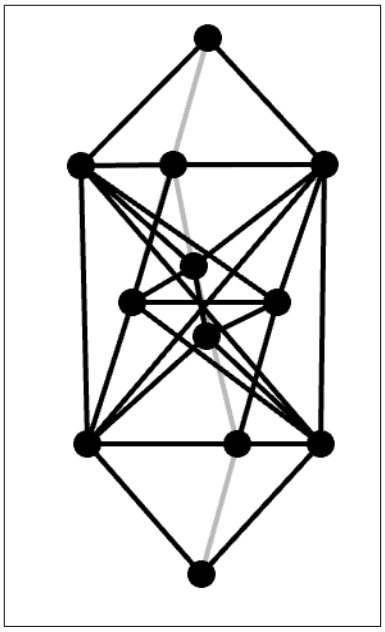
This algorithm starts with Cartesian combinatorics, a special version called “stair-pair”. We extend it to allow Möbius topologies. This property is called “semi-contractibility”. This gives us an Outside foundation of natural numbers that is called “Harry Potter Patterns”.

Here is a drawing my nephew made with the houses of Harry Potter on top of this graph.

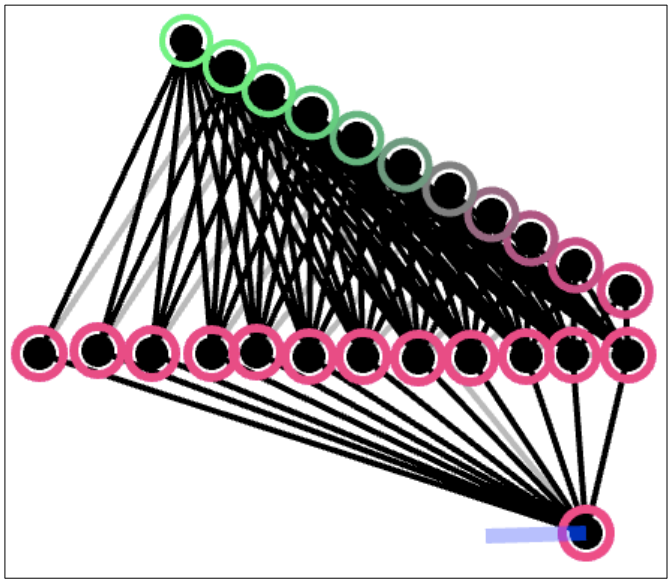


Here is the graph. All the nodes in the graph can be cores.

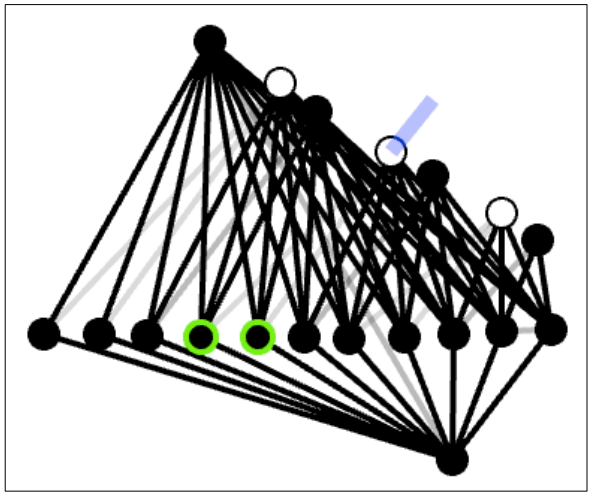




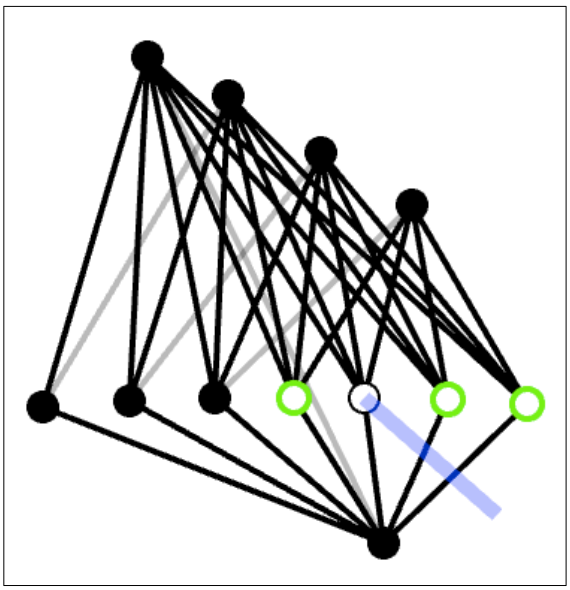
We can add new nodes.



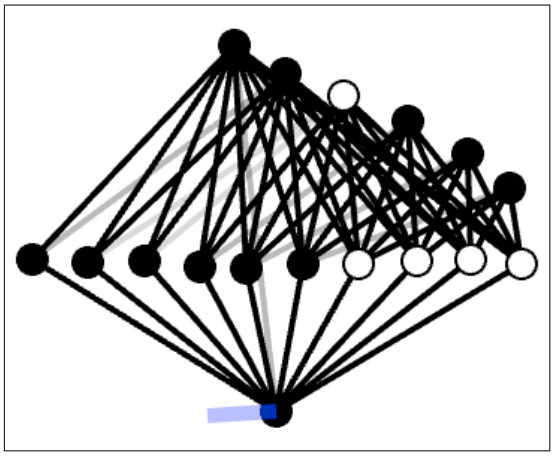
Here is a simpler view. This represents 10 or 11 depending whether you start counting with 0 or 1. The diagonal here is “1, 2, 3, 4, 5, …”. There is a layer of control nodes. There is one node at the bottom.



This graph has the property that when you erase nodes, it notices and marks the node below as white. It has a skip-error-detection property.



It also has an offset-error-detection property. For example, if you start with 5, it will label 5 white nodes. Here, it starts with 4, so it labels 4 white nodes.



You can also combine them. These graphs model natural numbers, but they do it in a very different way than Peano axioms of natural numbers. Here we have some kind of error detection. Through this error detection mechanism we can reason this sequence knows or models natural numbers.

Seshatism & Platonism
(Global, Outside)

The last section is Seshatism & Platonism. This is the Global Outside.

a ~~ a

This requires introducing a new operator. It is written with two wave symbols (`~~`). It is similar to equality, but it does not have reflexivity.

- Symmetry: `(a ~~ b) => (b ~~ a)`
- Transitivity: `(a ~~ b) & (b ~~ c) => (a ~~ c)`
- Lifts biconditions: `(a == b) => (a ~~ b)`
  (if `a` and `b` are symbolic distinct)

It has symmetry, transitivity and lifts biconditions when `a` and `b` are symbolic distinct.

partial equivalence (surface)
+
equality (depth)
=
quality

At the surface, this operator is partial equivalence. At the depth, it is equality. Together, these makes quality. We don’t call all partial equivalence relations for quality, only those that lifts biconditions.

equality (no `**e**`, no reflexivity)

The name comes from removing the “e” from “equality”. No “e” means no reflexivity.

$$\frac{(x : T) \;\&\; (y : U)}{(x \sim\sim y) : (T \sim\sim U)}$$

We want this operator to have the property that when `x` is qual to `y`, their types are qual.

$$\frac{(x \Rightarrow T) \& (x < T) \& (y \Rightarrow U) \& (y < U)}{((x \sim\sim y) \Rightarrow T \sim\sim U) \& ((x \sim\sim y) < (T \sim\sim U))}$$

I can rewrite this by introducing an order to propositions. I model this in propositional logic.

$$\frac{(x \Rightarrow T) \& (x < T) \& (y \Rightarrow U) \& (y < U)}{((x \sim\sim y) \Rightarrow T \sim\sim U) \& ((x \sim\sim y) < (T \sim\sim U))}$$

We don’t need this part, because this follows from axioms of path semantical order.

$$\frac{(x \Rightarrow T) \;\&\; (x < T) \;\&\; (y \Rightarrow U) \;\&\; (y < U)}{(x \sim\sim y) \Rightarrow T \sim\sim U}$$

I'll get rid of it.

$$\frac{(x \Rightarrow T) \;\&\; (x < T) \;\&\; (y \Rightarrow U) \;\&\; (y < U)}{\mathbf{(x \sim\sim y)} \Rightarrow T \sim\sim U}$$

I take this…

$$\mathbf{(x \sim\sim y)} \;\&\; (x \Rightarrow T) \;\&\; (x < T) \;\&\; (y \Rightarrow U) \;\&\; (y < U)$$

$$T \sim\sim U$$

… and move it above the line.

Core Axiom

$$(x \sim\sim y) \;\&\; (x \Rightarrow T) \;\&\; (x < T) \;\&\; (y \Rightarrow U) \;\&\; (y < U)$$

$$T \sim\sim U$$

This gives us the Core Axiom of Path Semantics.

- Proves many properties of Type Theory
- Adds propositional layers
- Improves brute-force performance on type-like problems

The Core Axiom proves many properties of Type Theory. It adds propositonal layers and it improves brute-force performance on type-like problems.

PL – Propositional Logic (classical)
IPL – Intuitionistic PL (constructive)
PSI – Path Semantical IPL

The logical languages involved here are: Classical logic, constructive logic and path semantical constructive logic.

PL = IPL + excluded middle
IPL
PSI = IPL + quality + core axiom

Classical logic is constructive logic plus the excluded middle. Path semantical constructive logic is constructive logic plus quality and the Core Axiom.

What is “Platonism”?

What is “Seshatism”?

So, what is Platonism and what is Seshatism? Platonism is a philosophical position about the existence of abstract objects. The dual of this philosophy is called “Seshatism”. It is a name I came up with because Plato is a man and mortal, while Seshat is a goddess from ancient Egypt, which is female and immortal.

Platonism: `a ~~ a`

Seshatism: `!(a ~~ a)`

We can trace back Platonism to self-quality. This means `a` is qual to `a`. Whenever you use symbols in some language, self-quality is a byproduct of the process. This is how we refer to abstract objects. We need symbols to refer to the Platonic world. However, when we assume that we can’t do this, we get negated self-quality and this is Seshatism.

Platonism: `a ~~ a` (abstract objects)

Seshatism: `!(a ~~ a)` (concrete objects)

If you think about Platonism as abstract objects, then concrete objects correspond to Seshatism.

Platonism: `a ~~ a` (equivalence classes)

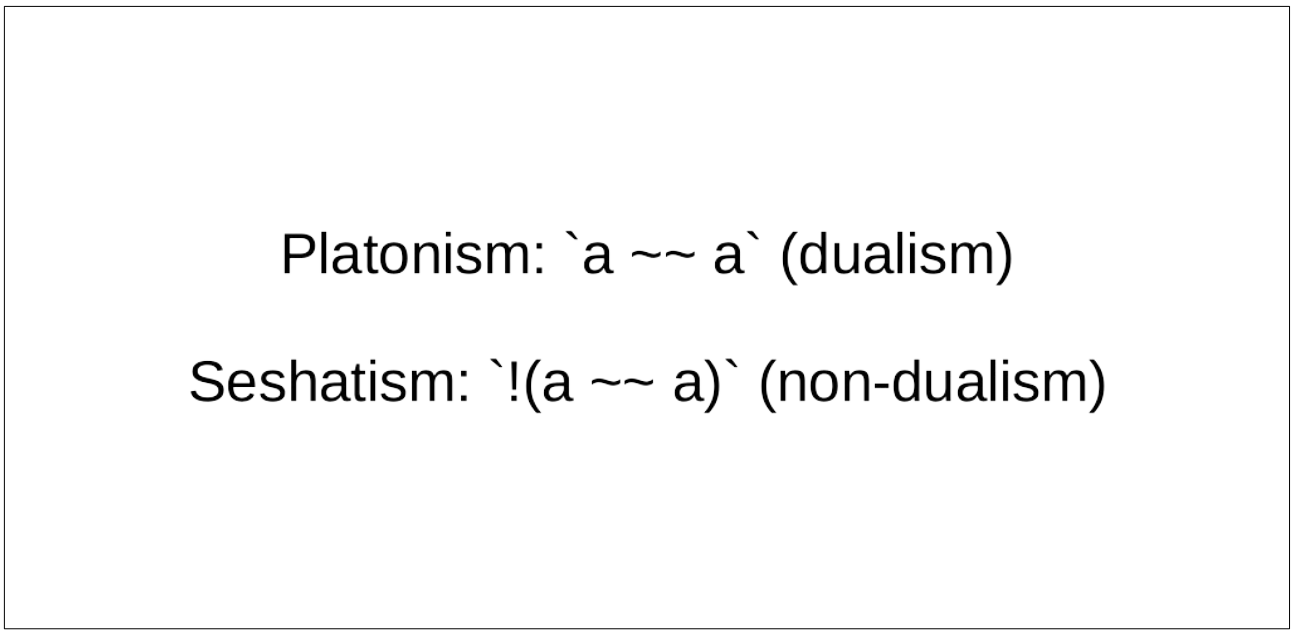
Seshatism: `!(a ~~ a)` (DAGs)

If you think about equivalence classes in Platonism, then the corresponding structure we get in Seshatism is Directed Acyclic Graphs.

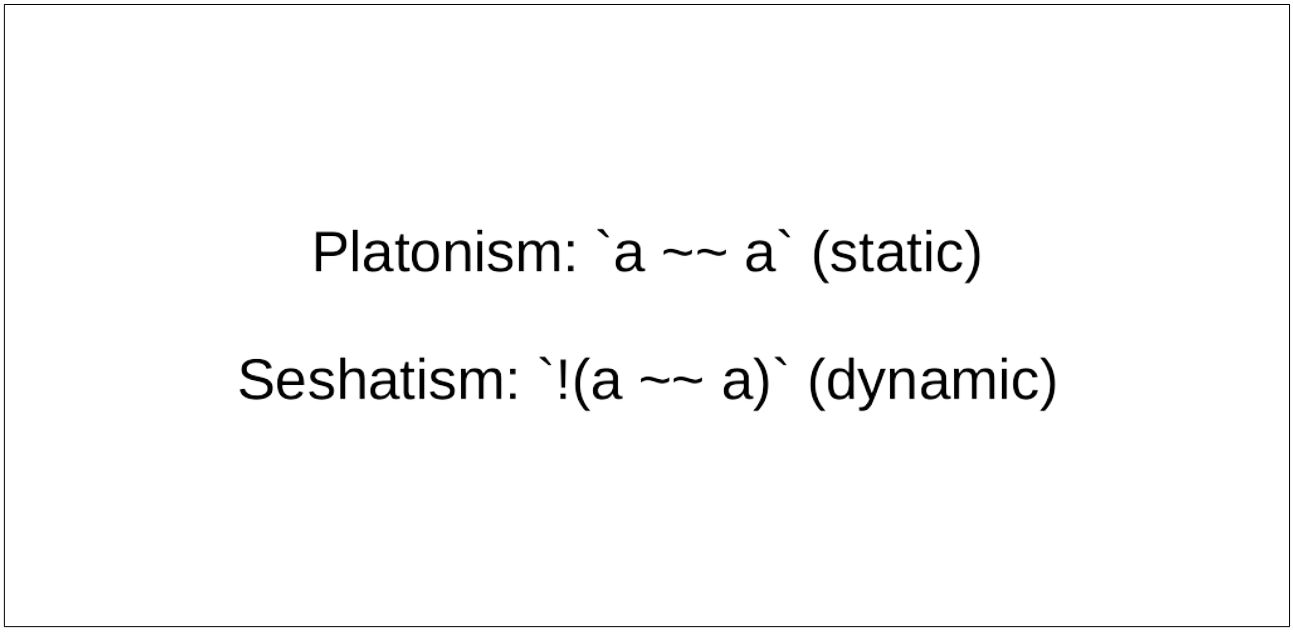
Platonism: `a ~~ a` (binary codes)

Seshatism: `!(a ~~ a)` (trees)

If you think about binary codes in Platonism, then we get trees in Seshatism.



If you think about dualism in Platonism, then we get non-dualism in Seshatism.



If you think about static in Platonism, then we get dynamic in Seshatism.

Platonism: `a ~~ a` (space)

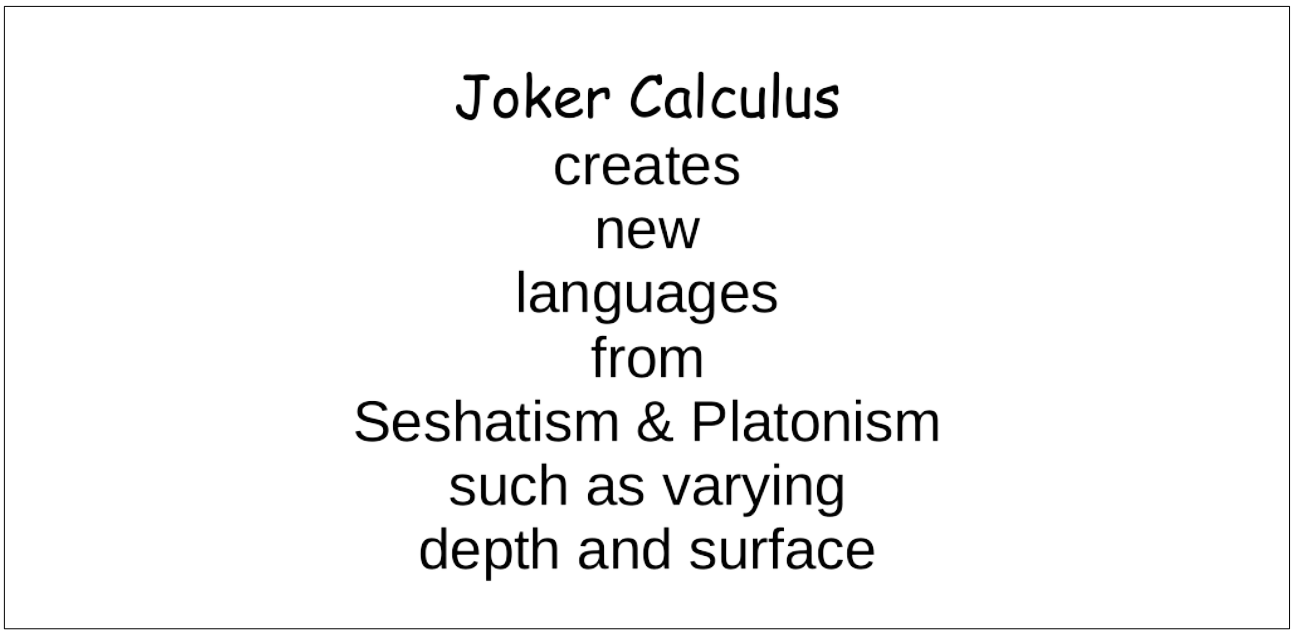
Seshatism: `!(a ~~ a)` (time)

If you think about space in Platonism, then we get time in Seshatism.

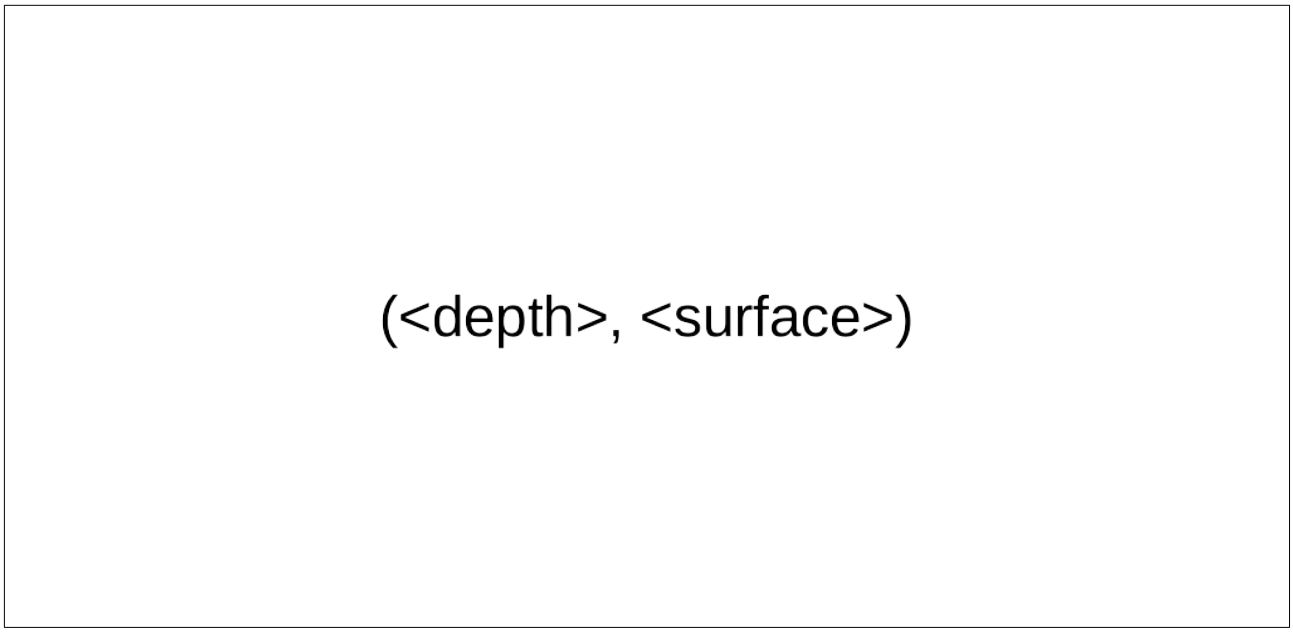
Platonism: `a ~~ a` (learn-by-correctness)

Seshatism: `!(a ~~ a)` (learn-by-error)

If you think about learning-by-correctness, such as theorem proving, in Platonism, then we get learning-by-error, for example machine learning, in Seshatism.



There is a calculus called “Joker Calculus” that creates new languages from Seshatism and Platonism, such as varying depth and surface.



This is written as a tuple where the depth language at the left side and the surface language at the right side.

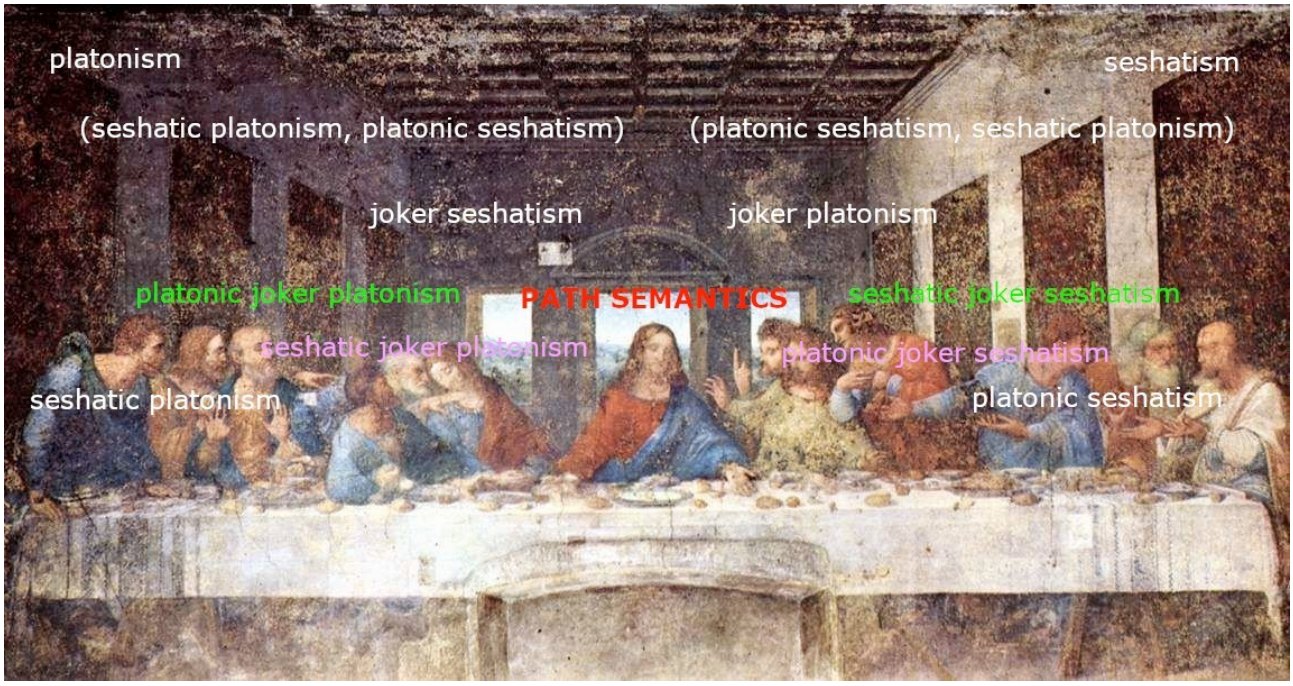
For example, `(seshatism, platonism)`.

This normalizes to `joker seshatism`.

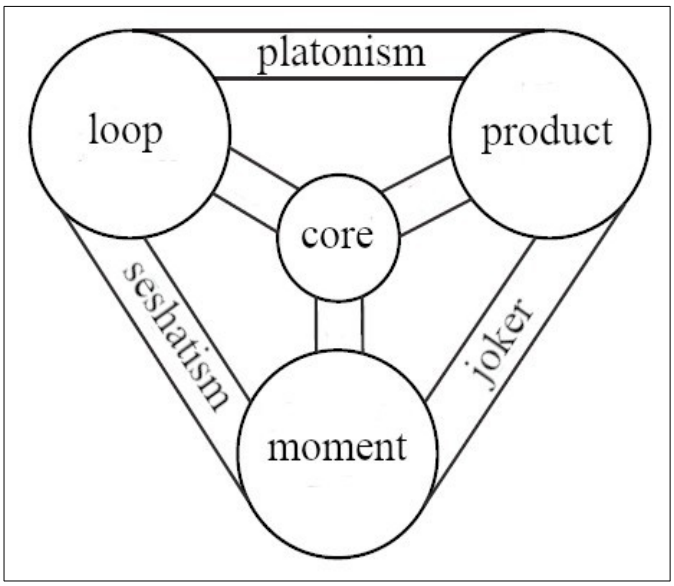
joker seshatism = sarcasm

This is the same as sarcasm.

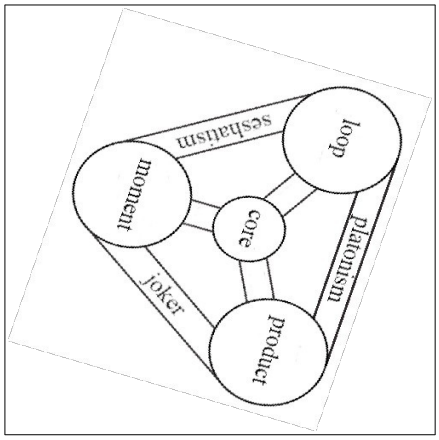
If I go back (2 slides), then sarcasm is when you say something that sounds like true, but what you mean is actually the opposite.

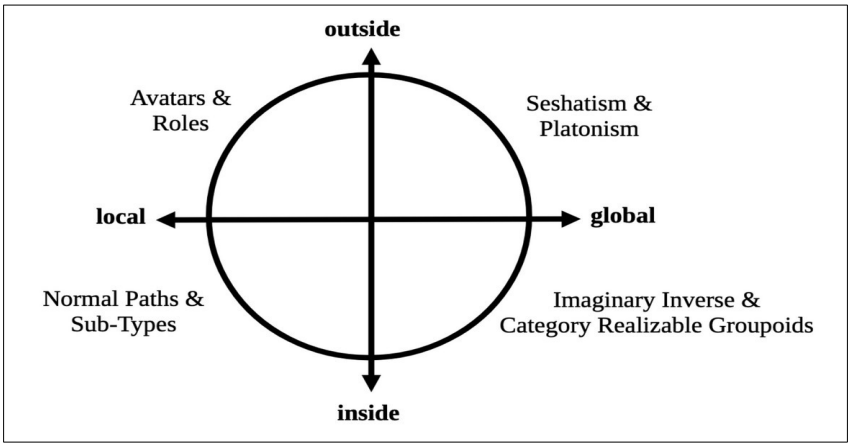


Joker Calculus gives rise to many languages that have different levels and variants. You can see on the left side, you have Seshatic Platonism, that is the philosophy of the Inside. At the right side you have Platonic Seshatism, which is the philosophy of the Outside.

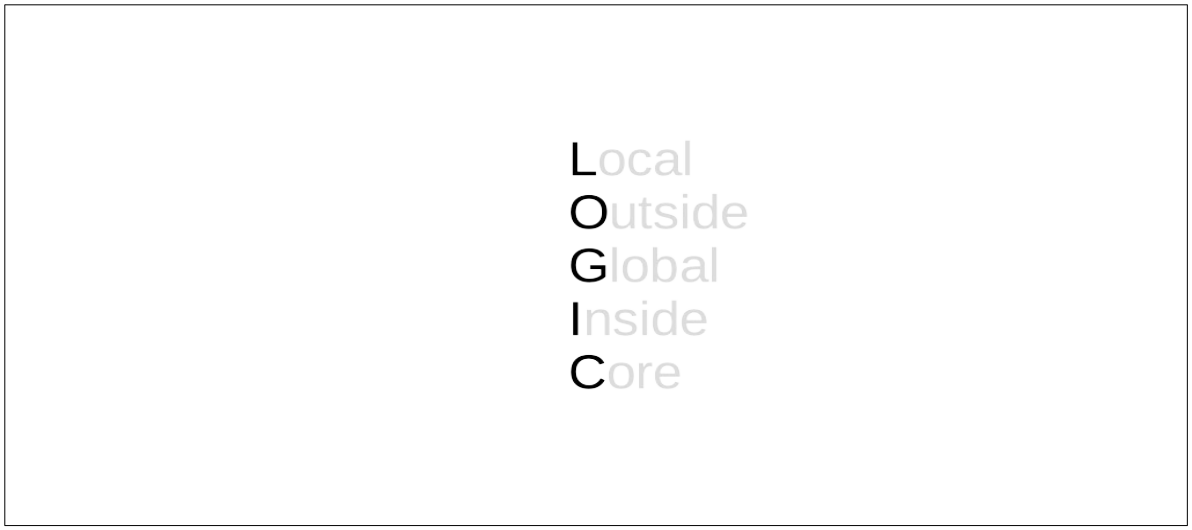


There is a corresponding Trinitarianism which uses something called “witnesses”. I won’t go into this in this talk. There are Loop, Product and Moment witnesses. Between Loop and Product there is Platonism. Between Loop and Moment there is Seshatism. Joker is between Moment and Product.





If I rotate this toward the Global Outside, then I get the Logi circle.



Local, Outside, Global, Inside and Core form an acronym: LOGIC. Thank you!