Motivations of Path Semantics

by Sven Nilsen, 2024

In this paper I show that Path Semantics arises from an initial semantic catastrophe, starting with the motivation to unify logic and types in a single coherent language that enables efficient brute force theorem proving. While there is no "perfect" language satisfying the original motivation, through a dialectical process the reader is guided by proofs to the solution of the core axiom and explaining how it satisfies other informal motivations of doing Path Semantics.

I consider myself a Wittgensteinean, despite the belief that Wittgenstein was mostly wrong. However, he was one of the few philosophers who went deep enough for people to say he was wrong. I am a Wittgensteinean, not because I follow him dogmatically, but because I believe in trying to work on the same problem he worked on: The synthesis between logic and language, which in my perspective must ultimately be profoundly human, no matter which direction we choose to go, because we are above all other forms of being limited and enriched by the human experience. For now.

When I think about why people should learn Path Semantics, given they are motivated, I can list seven ways to approach it which is not grounded in anything beyond the choice of motivation itself:

1. Improving efficiency of brute force theorem proving
2. Exploiting symmetry in type-like problems
3. Logical semantics of layers/frames/moments
4. Theorem proving about randomness
5. Deriving correct programs
6. Study of language bias and trade-offs in design
7. Having fun

In my view, people should do something out of their own interest, not because I think Path Semantics is some sort of answer. For those seeking answers, they will be disappointed because there are many more questions yet to be asked than answers to be found. For those seeking new questions, to explore math beyond any limited though of previous generations, Path Semantics provides the tools to build the vehicles needed for long time exploration into the unknown.

In our society where people often claim to know logic, there are very few people who actually do logic, so to save yourself some time, here is what logic is, in its most naive form:

```
f <=> \true
```

A proof \`f\` returns \`true\` for every possible input. That is all.

A logician might be thought of as a person who produces various programs which outputs \`true\` for every possible input. Now, this does not sound very exciting, because the output is the same every time. However, the trick is to combine existing programs with varying outputs in new and interesting ways. So, a logician is like an artistic performer, a magician, impressing the audience which as reward for reading the proofs of the logician get their ways of thinking altered.

This means that logic is simultaneously both very formal and very informal. The mechanistic evaluation of a proof is boring as it gets, but the interpretation of a proof is as creative as it gets.

Most people believe that logic is boring because they project a certain mindset onto others. Psychologists know today that people in general suffer from something called “optimism bias”. This bias is caused by people believing themselves to have better information than other people. Under no circumstance scientists were able to remove this bias, despite teaching people about it in advance of the experiment, except the case when the subject was depressed. So, each person thinks that their own life is much more interesting and meaningful than the lives of other people who they interpret as more boring and meaningless regardless of what other people do. This way, when people believe that logic is boring, they project their own mindset which often prevents them from learning that logic is not boring, a self-fulfilling prophecy.

Keep in mind that people who think logic is boring will read the previous paragraph and still think it is boring, due to optimism bias.

Logicians are in an eternal existential battle, where on one side they are expected to fulfill an extreme notion of rigid discipline, but on another side they are expected to baffle and awe their audience. Failing to so, in one way of another, might mean the end of their career. So, this is an almost impossible task to do, like the most difficult video game ever designed, yet logicians still happen to succeed from time to time, enough to keep logic going forward.

However, logic as a discipline is threatened by extinction, due to the rise of programming. Initially, programming was dismissed as menial labour performed by women to make the huge and slow computers perform the operations they were supposed to do. Men concentrated on the hardware design of the computer. After a while, when women started to change the world, men discovered that programming was actually fun and started taking over, a common social pattern in history. Today, programming is still male dominated. Logic lost its high status due to men changing their ways of thinking about programming.

In programming, while still using logic, there is no longer need to fully emphasize the interpretation as in logic, but the useful side effects of running a program. This means that the extreme difficulty of logic as an art branched out into many disciplines of programming where the richness and difficulty can vary. The world today is fully dominated by the art of programming, completely replacing politicians and rulers as the central powers in society, which merely serve symbolic roles today, providing rituals to convince people how to coordinate. People might still associate these symbols with power, but every programmer knows what actually runs the world in practice.

In Path Semantics, we say that logic is Platonic biased, while programming is Seshatic biased. So, the rise of logic is connected historically to a general trend where Platonism increases in influence. The rise of programming is connected historically to a general trend where Seshatism increases in influence. There is no clear boundary between humans and their tools for thinking, so taking a perspective of language bias in history is more useful.

Now, by understanding clearer what role logic has in society, one can turn to Path Semantics to talk about the axiomatic foundation of the language biases Seshatism vs Platonism:

!~a	Seshatism for some proposition \`a\`
~a	Platonism for some proposition \`a\`

The \`~\` operator is known as a “qubit”. It is tautological congruent, unlike most operators in math which are normal congruent. In classical logic, this generates a random truth table using the bit vector of \`a\` as seed. You will find an implementation of \`~\` in the Pocket-Prover library.

Somebody who has never been exposed to Path Semantics before might find this mysterious.

Some immediate questions one might ask:

- How can Seshatism vs Platonism exist both inside and outside of logic?
- Why can Path Semanticists claim that such language biases must necessarily exist?
- Is this some sort of trickery that violates the rigor of logic as a discipline?

These 3 questions will be answered, but they will probably not be answered in a satisfactory way because they lead to new questions, which lead to new questions and so on, ad infinitum.

Seshatism vs Platonism evolved from research on the logical foundation of Path Semantics. It turns out that in the field of Continental Philosophy, these ideas were already established, but outside the analytic tradition. This means, Path Semantics introduces the analytic foundation of these language biases. Path Semanticists only claim that there are certain motivations that lead up to this analytic foundation. The rest is interpretation of logical proofs and philosophical history. The rigor comes from the seemingly paradox that that there is no way to satisfy the original motivations with a formal foundation, but that Path Semantics arises from working around these problems.

With other words, Path Semantics comes from dealing with a semantic catastrophe.

To understand this semantic catastrophe, it is useful to first talk about what logicians mean by propositions. In the Western analytic tradition, a proposition was thought to be a statement that was either true or false without making any assumptions, or formally expressed:

(a | !a)^true   for all \`a\`

This limits language to classical logic, which was later replaced by constructive logic as the most important language for mathematical foundations. In constructive logic, logicians think of propositions as a mathematical object in homotopy level 1. These objects have the property that every proof \`$p_0$\` and \`$p_1$\` of a propositional type means the proofs are equal (in some sense):

$$p_0 : A \qquad \& \qquad p_1 : A \qquad \& \qquad \mathrm{IsProp}(A) \qquad \rightarrow \qquad p_0 == p_1$$

Instead of just being true or false, a proposition in modern sense can mean anything that has the same “sense” in every context, e.g. a color, or a number, or any kind of symbol.

So, in order to get to propositions, logicians must first think about what propositions can mean. They find that it can mean infinitely many things, not just true and false. This change of perspective is significant to understand how logic as a discipline changed from the classical era to modernity.

However, despite the meaning of propositions being so fundamental for logic and therefore also for all of mathematics, there is an even more fundamental property that must be assumed before anything else in order to get things going, which is the following:

$$p_0 : A_0 \qquad \& \qquad p_1 : A_1 \qquad \& \qquad p_0 == p_1 \qquad \rightarrow \qquad A_0 == A_1$$

This means when two proofs are the same, their types must be the same.

This property is how logicians think about types in general. The idea of types was invented to solve Russell’s paradox in Set Theory. This idea was so useful that it ended up in most mainstream programming languages today. Types are not something that logicians can ignore. However, the problem with types is that they do not have any formal basis in logic itself! In fact, most languages with types treat them as special judgements that exist outside the theory, or object-language.

Path Semanticists look at types and see mathematicians making a choice. However, what does this choice mean? Can we claim to understand the foundations of mathematics without studying the most fundamental property of types? So, the task of a Path Semanticist is to translate the language of types into some form that enables us to talk about it in terms of:

f <=> \true

The basic idea is to introduce "levels" of propositions. These levels have some kind of symmetry between themselves, satisfying the fundamental property of types. Within each level, logicians can use normal theorem proving, but between the levels the language is constrained to the language of types. This way, the language of logic and the language of types can be combined.

As a useful notation we separate the propositions by levels using parentheses, starting with level 0 and then level 1, etc. The operator `:` is replaced by `=>`:

$(p_0\ p_1) \qquad (A_0\ A_1)$:
$p_0 \Rightarrow A_0 \quad \& \quad p_1 \Rightarrow A_1 \quad \& \quad p_0 == p_1 \quad \rightarrow \quad A_0 == A_1$

Here, there is no hidden symmetry between the levels. The notation is just used this way to illustrate where the propositions are located by level, without attributing them any special meaning beyond making it easier to read. The language is still ordinary logic (normal and boring).

Now, this looks all just fine. Until one discovers that negative proofs as propositions have no meaning in the language of types, yet they are needed in the normal language of logic. If the only problem was that such statements did not have any meaning in the language of types, then this would be OK, as long we still could use normal logic. However, the problem is much, much worse! In fact, it is not possible to "think negative thoughts" in one level without "polluting" the next level:

$(p) \qquad (A_0\ A_1)$:
$!p \quad \rightarrow \quad A_0 == A_1$

This is because `!p` implies `p => $A_0$` and `p => $A_1$` and `p == p`.

The semantic catastrophe happens by logicians being unable to use language of normal logic and language of types at once. A logician can use such layers to prove theorems about type-like problems. Or, a logician can use such layers in isolation to prove normal theorems. However, it is not possible to integrate the two different use cases. One must choose one or the other.

OK, one might think, what is the big deal?

The big deal is that if one can unite the two use cases, then it is possible to improve the tools of logic since most practical problems are type-like in the real world, yet we want the tools of logic to reason about them, outside the context of how a specific implementation works.

For example, in brute force theorem proving, by exploiting the symmetry between layers with built-in semantics of types, one gets exponential speedup! A modern laptop can handle 40 propositions in a minute on single CPU, but by formulating a problem using type-like symmetry the same laptop can handle up to 60 propositions (e.g. 30 in two layers). However, the same system can not use normal theorem proving except for a single layer at a time, wasting the other layers. While this improvement of 40 to 60 might not seem that significant, it is in practice because one can pack a huge amount of math in this extra space and use them as building blocks for more advanced math.

In programming competitions, e.g. the demo scene, a program of size 4kB can be vastly more complex than a program of 2kB. While these small sizes are mostly rules for participants, the innovation of packing complexity into small spaces and times drives programming as a discipline forward. The art of programming can be somewhat measured using concrete limits and constraints.

Inspired by this idea, Path Semanticists want to measure the progress of logic as a discipline. Brute force theorem proving enables people to not bother about individual axioms and rules, but about logic seen from the boundary of language in terms of resources.

So, in Path Semantics the big question is whether there exist a "perfect" language within some notion of constraints that satisfies the combined use cases of normal logic and types.

The answer? No.

This has vast philosophical consequences. When mathematicians introduce types in their notation, they are not basing this idea on any logical foundation, at all. Despite the fundamental property of types being agreed upon by all mathematicians, there is no underlying model of this property, even in principle, in reality. The language of normal logic and of types are forever separated by choices, which are made arbitrarily by individual mathematicians, depending on the practical need.

In summary, after more than hundred years of mathematical research on foundations, the conclusion is that the most universally agreed upon property that comes prior to all other assumptions, before even talking about propositions, is merely fiction. Let that sink in.

In an extreme sense, it is not wrong to say that mathematics is a failed project, at the very start. Yet, in practice, mathematics seems to work more or less, often more than not, flawlessly. All our technology is thanks to math. How is it even possible that the most basic idea of it all is fiction?

To understand this, I will take you on a journey, showing step by step, by a series of incredibly creative usage of logic, that this inevitably leads to Path Semantics and hence all the weird ideas surrounding it, such as language bias of Seshatism vs Platonism. The inescapable conclusion is that mathematics is essentially based on ideas, that individually, might get me fired from any logics department today, had they been given the opportunity. Luckily, I approach this problem as a programmer and we are going to hack our way together through the jungle of epistemic uncertainty.

First, I am going to ignore the original motivation for solving this problem,
by introducing an `sd` operator that represents symbolic distinction:

$$\begin{array}{ll} (p_0\ p_1) & (A_0\ A_1): \\ \mathrm{sd}(p_0, p_1)\ \ \&\ \ p_0 \Rightarrow A_0\ \ \&\ \ p_1 \Rightarrow A_1\ \ \&\ \ p_0 == p_1 & \to \quad A_0 == A_1 \end{array}$$

Since `p` is symbolic identical to `p`, this undermines the idea that
if `p : $A_0$` and `p : $A_1$`, then `$A_0 == A_1$`.

However, if there is anything that mathematicians universally agree upon, then it is this property. Why would I choose to undermine the most fundamental idea in math?

The basic idea is that since this idea is so fundamental but unsolvable, perhaps one can talk about it indirectly instead? Perhaps this would suffice for most applications? Let us go with this idea.

It turns out that while `sd` is not possible to express in normal logic, it is in the language of brute force theorem proving. However, to explain why, it helps understanding how Pocket-Prover works.

The Pocket-Prover library exploits the property of logic that no operators branch on the CPU, so instead of processing single `bool` types, it packs 64 bits together in an `u64` type and does them all in parallel using operators on `u64`. This improves the performance 64 times, since modern hardware has 64 bit instructions. Now, this is only possible because logic works the way it does, regardless of what is expressible within logic or not.

It is not allowed for normal logic to “peek” sideways on other bits that are processed in parallel, without erasing this knowledge somehow. One can think of normal logic as only concerned about single bits. The solver prepares truth tables such that every possible combination of truth values are tested. This is why it takes exponential amount of time to check a proof with respect to the number of argument propositions. A common trick is to use alternating square waves of fixed frequency, like when counting in a binary number system. Every expression that is evaluated corresponds to some square wave, but the signal can be arbitrary complex. In a finite proof, there is a finite number of argument propositions which can be used to construct other expressions. Since every proof of few argument propositions must be valid by extending the proof with new argument propositions, there is no “upper limit” to the complexity of the square wave. One can think about this as normal logic being agnostic to what kind of world exists out there. There are no imposed constraints on the complexity of the world. The more complex the world gets, the more combinations of expressions can be expressed and this complexity grows exponentially.

So, the brute force solver manipulates the input over a set of possible states. However, there is no such requirement that the solver must treat every proposition as independent of the others. The language of the brute force solver is more powerful than the language of normal logic. This is why it is possible for a solver to skip certain inputs such that, in the combination with the symmetry of type-like problems across levels, the symmetry appears to take into account symbolic distinction. It does not need to declare the `sd` operator explicitly, which is a good thing, since the `sd` operator is only well-defined for inputs that are symbolic distinct and otherwise undefined. Therefore, by skipping certain inputs, treating all propositions as part of one mathematical object, it is possible to construct a language that appears to understand symbolic distinction implicitly. The user is never allowed the freedom to express `sd`, anywhere, so it does not strictly violate normal logic.

This language is called “PSL” for Path Semantical Logic. It is a classical logic (excluded middle) with layers of propositions, using symbolic distinction and type-like symmetry between layers.

In PSL, it is impossible to prove:

$$\begin{array}{lllll} (p) & (A_0\ A_1): & & & \\ p \Rightarrow A_0 & \& & p_1 \Rightarrow A_1 & \rightarrow & A_0 == A_1 \end{array}$$

Notice that here, the layers contain the hidden symmetry of type-like problems.

What PSL can not prove seems to be desirable from one perspective of mathematics, but when reformulated using negative `p` one can see why a such theorem should not be provable:

$$\begin{array}{lll} (p) & & (A_0\ A_1): \\ !p & \rightarrow & A_0 == A_1 \end{array}$$

A such case is a semantic catastrophe that collapses propositions in level 1.

PSL does not actually fix the problems except removing one case of the semantic catastrophe. The language allows theorem proving on type-like problems, or allows normal theorem proving in a single layer, but not both at the same time. This is due to the notoriously absurd Creation Theorem.

The Creation Theorem is as following in PSL:

```
(a b)         (A B):
!a      &     b => B        →     A => B
```

Similar to the reason symbolic distinction was introduced,
the problem lies with expressing negative statements in level 0.

PSL allows normal theorem proving within each level, as long there is no communication between levels. With other words, each layer is a kind of parallel universe in normal logic. In another perspective, but incompatible with the parallel worlds, it can be used to reason about type-like problems across levels. It improves upon the situation where normal theorem proving in a single layer causes problems in higher layers. However, it still does not allow combining type-like problems and normal logic. When some proposition is stated negatively in level 0, it causes seemingly “divine magic” in level 1, through connections, but those connections seem unrelated to what caused the event. Hence the “creation” event.

Even for logicians, the Creation Theorem is so unbelievable and unintuitive that one feels compelled to check this for oneself before trusting that this is an actual theorem and it is not caused by any bugs in the brute force solver. It is the logical analogue of demonstrating that magic is real.

With symbolic distinction undermining the basic motivation for the most fundamental and agreed upon property of types, prior to all other ideas, it is even more incredible that the Creation Theorem happens in addition to it. If it was not the case that PSL can be used on type-like problems, as a hidden language inside normal logic with extended levels, then there is good reason to be skeptical. From one perspective, I would be immediately fired from the logics department, but luckily the use case of practical theorem proving saves the day and it does so extraordinarily: Exponential speedup.

Now, a Path Semanticist is not satisfied with a such demonstration of real magic. It is time to really get into the weird world beyond what normal people consider logic. Path Semanticists take a look at the situation and says: Let us fix this problem with noise!

If there is anything that logicians do not like, then it is noise. This is because noise destroys all the certainty that one has grown accustomed to in logic. From a programming perspective, one can not exploit noise in the same way when using brute force solvers. It means to give up the layers of propositions with hidden symmetry. So, why do it? Why move further away from the initial motivation of studying type-like logic? Should not the whole point be to make progress towards it?

Even in the hypothetical scenario where logicians get used to PSL, I am going to get fired again, because I am going to break the rules once more, beyond the already weird situation we are in.

Welcome the qubit operator `~`!

The qubit `~a` takes the bit vector `a` and uses it to generate as a seed for a random truth table, a new bit vector which is non-deterministic. Each time one evaluates the proof, one might get a different answer. However, the solver can sample the proof and minimize the energy, such that if in any case the proof is `false`, then it is `false` forever. The downside is that one can never be sure that a proof is `true`. This makes logic more like a scientific activity than a deterministic discipline.

In addition, the qubit operator checks one fixed bit in the bit vector to decide whether to flip all bits or not. This provides the property `!~a == ~!a`. So, if I have not already been fired, then this would be the second time to fire me for blatantly violating things you are supposed to do in logic.

And it works.

The qubit operator `~`, despite being the most hackish thing ever, works seamlessly in normal logic. Not only does it work, it corresponds to the analogue of introducing new propositions!

It literally blows the mind of logicians. Literally. It produces an extensible semantics which blows out the limited universe of finite propositions, making logic infinite-valued. Not even the ancient Indian Catuṣkoṭi, a four-valued logic and its generalized versions, had the imagination of the qubit operator. This new universe is so large and complex, that the number of binary operators grows so fast, applying the qubit operator up to 18 times, would collapse the physical universe into a black hole simply by listing them all. It requires a 4TB hard drive to store the number of binary operators in level 18. Each nested level applying it once more, is so complex compared to the previous level as colors on a computer screen is to black and white. It is literally the very essence of blowing the mind, formulated.

Normal classical logic (excluded middle) with the qubit operator `~` is called “PSQ” for Path Semantical Quantum propositional logic.

I would have been fired twice for this, even after PSL, had it not been for the introduction of new propositions. However, I am going to get fired again, as no logical department is safe for me.

Grudgingly, logicians can check that the qubit operator works, even for single bits. The trick with using the bit vector improves performance, but it does not violate the basic principles of logic. For any size of the bit vector, there exists an associated algorithm for the qubit operator.

However, I am not done.

Instead of `sd($p_0$, $p_1$)`, I put `~$p_0$ & ~$p_1$` into the property of types:

$$\begin{array}{ll} (p_0\ p_1) & (A_0\ A_1): \\ \sim p_0\ \&\ \sim p_1\ \ \&\ \ p_0 \Rightarrow A_0\ \ \&\ \ p_1 \Rightarrow A_1\ \ \&\ \ p_0 == p_1 & \rightarrow\quad A_0 == A_1 \end{array}$$

Here, I do not use PSL, so there is no hidden symmetry between layers.

This is the most horrible choice imaginable, because what would be a seemingly saner choice:

$$\begin{array}{ll} (p_0\ p_1) & (A_0\ A_1): \\ \sim p_0 == \sim p_1\ \ \&\ \ p_0 \Rightarrow A_0\ \ \&\ \ p_1 \Rightarrow A_1\ \ \&\ \ p_0 == p_1 & \rightarrow\quad A_0 == A_1 \end{array}$$

At least this would not bias the language toward some yet-to-be-introduced propositions.

However, this would enable proving `$A_0 == A_1$` from `p : $A_0$` and `p : $A_1$`. Remember, I had already given up on trying to formalize the most agreed upon idea of types. What I want is something that works in practice for most applications. There is no “perfect” language that solves all problems.

To enable backwards compatibility with PSL from PSQ, I add the following axiom:

$$sd(p_0, p_1)\quad \&\quad p_0 == p_1\quad \rightarrow\quad \sim p_0\ \&\ \sim p_1$$

This is because `sd` is undefined for `sd(p, p)`, so the qubit operator `~` allows an escape hatch. Now, if I want to enable the type-like property for `p`, I can just state `~p`. Boom!

Like magic, PSQ solves the problem of expressing the most fundamental idea of math,
which is the basic property of types. It just requires… faith.

What is faith? It is believing in something that is not directly accessible through reality. With other words, a projected proposition, a new proposition on its own, kind of like `~a` in relation to `a`.

So, math is basically made out of faith? Let that sink in.

Surely I would have been fired by now, but PSL gets in the way. The exponential speedup is so convenient that even introducing faith into logic is acceptable as long it builds a bridge back from PSQ to PSL. Logicians have to specify axioms for when type-like problems occur, which is burdensome, but it is not that bad in the context that one can fall back to PSL when needed.

It is more or less a miracle that somebody could in principle defend me to keeping my job, given that I was given one in the first place. However, that is not the kind of world we live in.

This means, in the most horrible way imaginable, PSQ allows us to think about the synthesis between logic and types, using faith of all things, which leads to the language bias of Seshatism vs Platonism. You see, the bias toward `~p` is an internal difference that is not detectable between speakers of logic, but only within the same language of logic. This is a recent idea in philosophy.

Instead of getting fired, the logic department, pride of its analytic tradition, have to pick up books by philosophers in the past century. They were absolutely convinced that these philosophers were talking gibberish, but now the stuff in Continental Philosophy somehow comes back to bite them.

By ignoring my original motivation, I took a detour to pick up the qubit operator which turns out to be the key to solving the problem. However, it adds some scary implications, a sense of doom, to the very foundation of mathematics. We are not going to escape language bias?

Anyway, these ideas must be discussed another time. For now, let us finish the work.

It would be nicer if we had the following:

$$\begin{array}{ll}(p_0\ p_1) & (A_0\ A_1): \\ \sim p_0 == \sim p_1\ \ \&\ \ p_0 \Rightarrow A_0\ \ \&\ \ p_1 \Rightarrow A_1\ \ \&\ \ p_0 == p_1 & \to \quad \sim A_0\ \ \&\ \ \sim A_1\ \ \&\ \ A_0 == A_1\end{array}$$

This allows the same process to happen over and over, from one layer to the next. Each layer might be thought of as a “moment” in time, causing new moments, ad infinitum.

To make this simpler, I introduce a “quality” operator `~~`:

$$p_0 \sim\sim p_1 \qquad \Leftrightarrow \qquad \sim p_0\ \ \&\ \ \sim p_1\ \ \&\ p_0 == p_1$$

Now, we have reached the core axiom of Path Semantics:

$$\begin{array}{ll}(p_0\ p_1) & (A_0\ A_1): \\ p_0 \sim\sim p_1\ \ \&\ \ p_0 \Rightarrow A_0\ \ \&\ \ p_1 \Rightarrow A_1 & \to \quad A_0 \sim\sim A_1\end{array}$$

At every step, I have used various tricks to make progress. However, these tricks were largely unintuitive. Yet, there seems to be no other possible way. In attempting to unify the language of logic with types, one gets Path Semantics, like it or not. It does not help that talking about path semantical quality seems to invoke feelings of something “beyond”, as in consciousness.

At this moment in time, humanity has not yet managed to solve the secret of consciousness. Every attempt so far has failed. This is because consciousness inherently is based on some notion of subjectivity, where a person makes language boundaries between a personhood and the world.

However, these language boundaries are not fixed. They allow different language biases. Since one can talk about these language biases in Path Semantics, it makes it relevant when talking about consciousness. Path Semantics is much more powerful than just a tool in this particular area, but it opens up a can of worm, or a new wonderful world of meaning, depending on whether you like it at the moment or dislike it. There is no rule that you have to like Path Semantics all the time.

The quality operator `~~` is called “quality” because it is a partial equivalence relation, no reflexivity, hence the “e” in “equality” vanishes. It happens that this word also overlaps in semantics when we attribute some quality to an object in the world. In physics, a quality is not part of the object itself, but a projected property.

For example, when a quality of food is that it tastes good, then this is not something about the food itself, even though it appears to be, because food tastes good only when it is structured in a certain way. However, the judgement that the food tastes good is an interaction with a person who eats it.

This means, a quality is not something that is part of an object itself unless there is an “other”.

Actually, this idea gets more complicated the more we study the world through physics, because objects do not seem to have well defined properties in many ways, unless they are observed.

In Platonism, it was also thought that a quality of an object was a relation between the object and its abstract form. So, a person had the quality of a person through the idealization of the idea of a person, a “perfect person” existing “out there” in the Platonic world.

Naturally, due to ancient Greece lacking women, as only 1 in 10 were women, this idea resulted in oppressing the rights of women across the Hellenistic world, thanks to Alexander the Great. This was because women were perceived as less perfect versions of people than men.

When Plato wrote about the invention of writing, he told a story about the god Thoth from ancient Egypt. This story was perhaps thousand years old at the time he was writing. Yet, Plato did not know that Thoth had a female counter-part, Seshat, that represented writing in a completely different aspect. Seshat sacrificed her originality of writing such that people could be original creators. When a book was finished, Seshat copied it into her divine library. This idea of originality, is the dual of the abstract “copying” of Platonic forms. The discovery of the dual in the core axiom of Path Semantics inspired the idea of “Seshatism”, a dual philosophy of Platonism.

Seshatism might be thought of having existed as a developed philosophy for thousands of years in ancient Egypt. The way ancient Egyptians thought about writing, was not merely as a tool to record history, but also as a way to bring something desirable into existence, into the cycle of time. This is reflected in the idea that Thoth could create five new days in the calendar. Women had much better rights for thousands of years in ancient Egypt, even compared to many places today. Property was inherited from mother to daughter. If you were a female slave, then you got payed, but not if you were male, etc. The female and male aspects of deities were balanced through an idea “ma’at” which was the balancing principle when pharaoh weighted his heart on a scale in the demon world. This philosophy was lost over time, buried under sand in the slowly corrupting and descending process of the ancient Egyptian empire. In modern times, artists take this theme up again, inspiring philosophers to look back to the same sources. The rise of Seshatism has merely begun.

There is no source of language bias toward Platonism beyond the core axiom of Path Semantics. Mathematicians have a usually a feeling that mathematics “ought to be” abstract, yet there is no formal underpinning model of that choice, except one that is done arbitrarily. Math is entirely symmetric in the language bias of Seshatism vs Platonism. This means, for every abstract mathematical idea we produce, there is a corresponding “outside” idea. In Path Semantics, one can even define this distinction between mathematical languages as Inside vs Outside theories. An Inside theory is a language where an unknown is some sentence within the language. An Outside theory has at least one symbol which is outside the theory of the language. The natural numbers have been constructed in a pattern of filled Avatar graphs, providing an Outside theory of the understanding of the same objects. However, it is currently unknown how to prove that this language is the same as the Peano axioms, because it works by completely different principles. In this language, one can only “know” what a natural number is by what errors are produced when modifying the pattern. It is like an error-correction mechanism. Similar ideas are also found in representations of supersymmetric models in physics, called “Adinkra diagrams”.

There seems to be no escape from language bias, even in mathematics itself, at the most fundamental level of how logicians understand it. This can be a shocking discovery to some, because it triggers asking a lot of questions about how blind people have been behaving throughout history. For example, when treating each other badly, it is often due to language bias.

One example of the dual properties of Seshatism vs Platonism, is the following:

a ~~ b	→	~a	Platonism
!~a	→	!(a ~~ b)	Seshatism

These properties are interpreted in relation to originality vs copy-ability. When stating `a ~~ b`, one can think about it as copying the essence of `a` into `b` and vice versa. A property of the quality operator `~~` is that `a ~~ a` is the same as `~a`. However, when stating `!~a`, this copying process is not possible, for any proposition `b`. So, there is no Platonic form out there that can be used to judge that a person is not like a “perfect” idea of a person.

In Seshatism, an object is “original” in the sense that it can not be copied. Originality is not a property that follows from the process of cloning. Even in the case when one has two perfect clones which have equal claims to be original, there is some historic original. This idea overlaps with identity metaphysics. The use of the idea of originality is important in a civilization, because it decides who gets to make money from some intellectual work. However, it also comes with a negative side: Over the course of history, male philosophers were often credited ideas invented or discovered by female philosophers.

In principle, one can choose a core axiom that has the same bias toward Seshatism as the one biased towards Platonism. However, this axiom is essentially the same! The difference between Seshatism vs Platonism is an internal difference, undetectable between speakers of logic. There is something strange going on here, where a person can detect the distinction within their own minds, yet it fails to manifest as a concrete process between objects.

When archeologists find people in ancient history, the feminine and masculine features were less visible than in the modern world. Through history, what is considered feminine and masculine has changed, fluctuated and sometimes reversed. So, while language bias plays a huge role in how people perceive gender roles, they can change. It is not something inflexibility and certain, like how most people think about logic. Now the time has come to think of logic itself through this lens.

The conclusion Path Semantics leads to is that there is no such thing as completely objective math.

All math is relative to languages we design as human beings. These languages can operate in the world through instructions given to a computer. However, there will always be language bias in every such system. There is no philosophical higher meaning of the correctness of such systems beyond serving the benefit of people. You are not allowed to think higher about yourself just because you can use logic. This is because logic itself is limited by language bias.

The synthesis of the dialectical Wittgensteinean problematics between logic and language seems to our level of understanding today, to be limited to profound human activity. In some sense, what it means to be human in this intellectual perspective is to find these irreconcilable languages which do not go together, requiring a person to be there to explain how it works. Your meaning of life is not depending on a specific theory of the world, but in the very lack of finding a satisfying solution. Yet, we have to make choices anyway. The future is shaped by the errors we make over time. There is no “perfect” world we can make, because there is no such proper judgement. Despite this struggle, humans try to survive and make the best out of it.

Only because math starts with a semantical catastrophe does not mean that everything is lost.
I think mathematics has done pretty well!

Now, look back at the motivations people have to learn Path Semantics:

1. Improving efficiency of brute force theorem proving
2. Exploiting symmetry in type-like problems
3. Logical semantics of layers/frames/moments
4. Theorem proving about randomness
5. Deriving correct programs
6. Study of language bias and trade-offs in design
7. Having fun

Did you think, when reading this the first time, that 7) was kind of childish? After learning about how deep this goes, how little we understand for now, have you changed your mind?

If you did, then I have already accomplished my intention of writing this paper. The point is not that one can apply one amazing trick after another to expand the ideas that cement in society. My point is that there is no further motivation needed of doing so than having some fun!