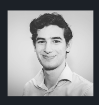
A Universal Language

One Ray to rule them all, One Ray to find them,
 One Ray to bring them all, and in the darkness bind them.

An implementation of Rays
[31]

[32]
: A Universal Language.

OrbitMines Research



30 June 2024 DRAFT: POSSIBLY IMPRACTICALLY VAGUE

Introduction

This thing is, in essence, a language to understand inconsistencies. A conceptual framework to make sense of ambiguity: A story of how destructively confusing languages can be. Though to me, most importantly, it is here as infrastructure. Infrastructure for the design and implementation of a .

A simple way of phrasing this, is that the concept of a (hyper-/) 'Vertex', (hyper-/) 'Edge', (hyper-/) 'Graph', (hyper-/) 'Rule', (hyper-/) 'Tactic', (hyper-/) ..., (hyper-/) 'Rewrite' are merged into one thing: a Ray. It handles surrounding context, ignorances, equivalences, ..., differentiation (And if it cannot, then it offers a way of implementing it for all of the above).

Though quite importantly, even if those previous words are complete nonsense to you: Either this, or projects following from this, will aid in your understanding. This is the start of a story which will provide infrastructure for communication between all sciences, (programming) languages, compilers, interfaces, ..., videogames.

Let me show you how.

Arc: Where to start...

Admittedly, this starts with a simple concession: that this is a generalization of many ideas whose details I don't fully understand. Though crucially, this is not important for my purposes here.

A list which undoubtedly falls short, would contain: , , , , Covariant computation [6] [7], , , , ..., Infrageometry [11] [12]. A more complete set of ideas and a partial history of me becoming aware of them can be found in my archive [13].

I suspect that a large numbers of problems arise from a rather simple conceptual mistake. And perhaps calling it a mistake - is itself a mistake. For we are always forced to first find practical tools, before we can find better ones. But therein lies to me the possibility of that mistake: .

Perhaps you could consider this as my attempt to provide proper infrastructure for that exploration. Perhaps not just exploration, perhaps better communication.

Though, in my ignorance, only recently did I become properly aware of the scope of this project [15]. And so, even though I still need to learn more about his history at some point, allow me to take on ' naming: A Universal Language.

Arc: Core Ideas

A few steps back

Let's first take a few steps back; this will be necessary. First, you must throw out any kind of assumptions you're bringing to the table. Just like we'll do now for Rays: Anything we'd like to make, should be phraseable in our universal language. It wouldn't be much of one if that wasn't the case.

That however, doesn't necessarily make it easy to phrase the things we would like to phrase. But let's start somewhere anyway:

Not much of a somewhere. But the basic premise becomes this: I don't know what things around me look like. Let's start by looking around me in some direction:

I don't yet know what this is or means, I just know I moved in some direction. You'll start to see the pattern of what we're doing here: I need to start traversing to find things around me - otherwise I can't know about them.

Alright now let's try to move backwards.

You'll see that there's no recollection of what we just did. In order to say even something as simple as that, we need some notion of memory. We need some way remember what we just did. Let's try it again with a notion of memory of where we've already been:

And a simple move back:

TODO: [...]

Essentially what any of this comes down to. Is things are entirely inferred from surrounding context. Yet your abstractions can be ignorant of how you're using them. Whether something is a function, number, geometry, topology, ..., structure becomes quite hard to say when you consider its surrounding context [REPHRASE]. More usefully what we're doing here, is saying: "Can you see a difference? And can you ignore it?"

What is a Ray?

Simply put, a Ray consists of two parts. One part [....]

References

Direction, arrows, ..., one-way connections

Similar to . The only way to show you a one-way connection, is to have access to some way in which it is not one-way. What do I mean by that? - Quite simply put it is this: If you can't remember that you forgot something, you wouldn't notice. [REPHRASE]

Or in terms of our Rays: If I didn't have access to this [SOMETHING]

I wouldn't be able to point it out, so it would just be this:

Superpositions

Traversal

Arbitrarily branching, stepwise, superposed, ..., partial traversal & equivalences

TODO: [This needs some restructure, good ordering here]

TODO: [?]

...TODO... This way, you can just draw a single line (or even arbitrary structure), and say: "What if I wanted to regard that as the same? What would happen?". The answer to those are far from obvious.

Breaking Recursion

Local self-references, constants, orbits & Self-referential operators

Note that whenever you have a self-reference through operators. Either we break the recursion there through some implementation. Or we simply decide to stop orbiting. And say it could be any of these things, it could be any of some superposition of things.

Superposing Languages

Simultaneously having 'different levels of abstraction', superposing operators, 'multiple abstraction implementations', ..., simulation

Almost always with any abstraction, you'll see the following simple pattern: (1) First one of something, (2) then more of things like it, (3) then some recursive construction of that thing. And noticing that is far from obvious.

Switching Perspectives

But this introduces a rather hard problem, namely that: .

Naming & Grouping superposed Languages

TODO: [Move elsewhere?]

A lot of this comes from the realization. That most differences come from the context in which they're applied. But this presents a problem of how one often uses languages: Specific names for specific perspectives. And that makes useful generalization quite hard. [REPHRASE]

Essentially the problem becomes. When do you decide that a particular kind of perspective, or switch in perspective should have a different name associated with it. Essentially what we're asking, is: Why is it so important to name this differently? Would it be harder to find if one didn't do that? [REPHRASE]

TODO: []

Modelling Unknowns

Part of any task then, becomes this:
. This should somewhere be quite intuitive: You can use tools without knowing how to make those tools. Essentially wanting to understand unknowns, might as well be called reverse engineering: How is it done? What aspects of it can be
replicated, decomposed, ..., understood
?

This lands us in the world of descriptions, definitions, lazy functions, unresolved pointers, non-committal ignorance, awaiting function execution, questions, ..., conjectures. Or: We can point to things to which we don't yet have any (or a definitive) answer. Take for instance an extreme of saying: "Whatever this direction is, you need to follow it completely":

This is essentially what it means to point to something you don't yet understand: I point in some direction without having to define what that something is.

Compression

Generalization of (perceived, ..., partial) geodesics

Now that we can superpose languages, and state with better clarity what having access to certain operators, ..., structure even means. We now fall into the world of compression. As this allows for a generalization of 'shorter paths' given our capabilities.

Note that with compression, we're always necessarily partially ignorant [20] of context and relying on some invariance [21]. Thus, any story about compression, becomes a story of rediscovery. Bringing with it an incredibly complicated world: You will have to deal with redundancy, ambiguity, forgetting, assumption violation, ..., inconsistencies. Where changes in resources, ..., capabilities will always play a role in how, ..., when one can compress. Or even better: This will always play a role in any (partial) translation. It's just often ignored as a problem.

But let's first do a dive into the implementation details, before we start attacking these problems.

Arc: Full Implementation

Ray.py

Python Implementation

Ray.py - Ray.ts

Crosscompilation of Python and TypeScript Implementation

Arc: Examples

Example: Dynamics

Some preliminary intuitions for physics

Some of these physics-related intuitions will have to be confirmed elsewhere. This is currently not my priority for understanding. I'll defer to Jonathan Gorard's [22] [23] (and other's) work for that for now.

But for program dynamics we can ignore those connections for now, as they are probably quite similar.

TODO: [Ordered to talk about usefully (assymetry), higher-arity cases more in line with ignoring/invariances of that order. Where an invariance is something like a branching ray as a cursor along every entry of some other ray. ("Also, interesting to note might be that Von Neumann and Birkhoff attempted to ground quantum mechanics using order theory (their attempt was not very successful at that)." @pr)]

TODO: [Local changes move larger structures.]

TODO: [Cannot have interaction without an idea similar to gravity?]

TODO: [Shoving causal history in some direction?]

TODO: [Something like: Constantly all the rays as functions are executing (in orbits), then if something causes something else's behavior to change, you get the inconsistencies.]

TODO: [Particles are seemingly temporally stable orbits/modular structures?]

TODO: ["Wrong dynamics" from a particular perspective, often probably still keep traversing - they still work. It's just that from the perspective you wanted, they don't.]

TODO: [Reprogrammability & inconsistencies as foundational?]

TODO: [Invariances at start hence a modular structure might be a necessity physically]

Example: Mathematics

Some preliminary intuitions for mathematics

You could probably phrase mathematics as have access to the `.self` equivalency Ray, and assuming one can traverse that structure arbitrarily, and ignoring how one has access to that. Basically: I'm saying all these things:

are the same. But I'm ignoring how I know about that.

Or in other words: I'm assuming their consistency - and that assumption has consequences, as alluded to here:

And this becomes problematic for mathematics if there is some way to traverse from `.self`, back to the current ray we're referencing. [NEEDS REPHRASING]

Which would be the moment you'd call it an inconsistency. But that as an argument, only holds up if you can indeed traverse arbitrarily. But it is likely exactly this property which allows for homoiconic foundations of mathematics [25]. [NEEDS EXAMPLE]

Similarly. Concepts like - absolute equality -, follow a similar pattern [26]. Namely in the sense that it is an admission of a difference and the ignorance of it.

Arc: Rendering Engine

A Reprogrammable (Visual) Interface

Open inputs, outputs, compute substrate, ..., interfaces

Arc: Universal Version Control

Concurrency, Dependency management, Causal Histories, ..., Theorem proving

Version control, causal histories, theorem proving, ..., a (programming) language are all rather similar. Always we consider some sort of persisting, surviving, ..., crafted items whose rediscoverability is not entirely obvious. In the case of version control, or reversibility, the gnawing question becomes: "What if I didn't know about something?"

nor could rely on these things I currently know about.

could I rediscover this other thing I'm interested in?"

TODO: [Essentially, version control comes hand-in-hand with compression. It becomes a story of redundancy, ..., recoverability.]

Arc: Exploration and Discovery

Wrapping up

Future inquiries

...

And to wrap up this "Wrapping up" arc, allow me to repeat a few things already alluded to in my 2023 thought excerpts [27]: & . Which by default should carry over to everything I write down. Though repeating it might be necessary.

In any case, even though I did not understand the scope of what I was making: We edge ever closer to the project I have been anticipating for several years now: . This is merely one of the first few steps of many more to come.

Footnotes & References

[6]

[7]

[11]

[12]

[13]

[15]

[20]

[21]

[22]

[23]

[25]

[26]

[27]

[6]

[7]

[11]

[12]

[13]

[15]

[20]

[21]

[22]

[23]

[25]

[26]

[27]