Dear Elphaba Thropp can you explain exactly what is going on in this description of analogy-driven reasoning? That is, my question is, what kind of theories does this handbook use to formalize analogies, what kinds of logic, category theory, sequent calculus, what mathematical formalism? Sweet Oz, what is this?

Dear Elphie today there is one more possible computational definition of "analogy": Whatever LLMs think when asked to make or verify analogies, do you think that this is another valid definition, or just a joke told in the halls of the Shiz University Department of Computer Science and Artificial Intelligence, Elphie?

Am I right that the problem is that if you prompt OzTransformer-6.0-Base with "man : king = woman :" then it answers with "queen" because this is already a well-known analogy, and so it will be with certain extrapolated patterns of associations, but the real problem is never the embedding dimensions (even OzTransformer-4.0 has 4164 emb. dims.) but the way the waves of the text moves the vectors towards the continuation, such that if the signal is rare enough, the Transformer will never get it right, Elphie? That is, am I right that the analogies themselves are part of the embedding for the word?

Welp, I remember something scribbled on the lecture halls of some university, maybe Hogwarts, that "higher-order unification is undecidable", plus things like "Godel's incompleteness theorems should not be surprising once you notice that the logical theories themselves are defined in terms of natural numbers at the metamathematical level, the symbols, formulas, variables, everything are natural numbers, lists, etc.", Elphie, so this stuff is actually very slippery, just like things such as "causation" (A, not B, causes X) instead of "causality" (A can influence B), right?

Would Elphie tell me that the fact that HOU is uncomputable in general, with only Prolog and similar stuff like Hindley-Milner type systems being decidable, is exactly why proving new mathematical theorems by analogy requires a lot of rigor, and the best way to use existing theorems to help prove new theorems, is to come up with explicit conversions from the data-types of the new theorems to the data-types of the old theorems, for example, changing a combinatorial theorem to a statistical theorem in the continuous domain using limit, compactness and continuity arguments, by explicitly invoking the definitions of continuous spaces, such as real numbers = Dedekind cuts = equivalence classes of Cauchy sequences = decimal/binary/integer-base expansions up to identifying (integer)(infinitely many digits of value b-1)\*b^n with (integer+1)\*b^n, Elphie?

Dear Elphie, what is the correct recipe for changing combinatorial arguments into measure-theoretic ones in infinite continuity anyway? What is the name of this mathematical system that I can use in Google and ChatGPT alike?

The problem with LLMs is that, while they are really good at manipulating text and mathematical formalisms, they still fail to come up with analogies and proof ideas that I can easily spot, such that when I tell an AI to generalize a certain finite theorem to infinite cases, it fails to reinvent a technique that was formalized 30 years ago, Elphie

What I mean is, the best thing about LLMs that the Shizers should be really learning is that they help you generate the right theorem prover encodings, and the tools have been around for so long and run performantly on consumer hardware, the LLMs just allow you to use natural language to translate to formal language, Elphie. What I mean is, that's why they excel at solving competition questions, but are still horrible at trying to find the right generalization of some mathematical theorem to new contexts, and that's with Chain-of-Thought backing, Elphie.

What I mean is, the computational complexity of the particular problem of generalization, plus machine-learning impossibility theorems like NFL, PAC-Bayes bounds, algorithmic information theory (Chaitin's incompleteness theorem), etc. makes sure that any system that dares to generalize this way using even things like parallel Chains-of-Thought (P-CoT) will just burn GPU time to output tokens, a lot of them, before it can come up with anything useful, and therefore the heavy work still goes with the existing theorem-prover tools and some human ingenuity, Elphie

Such that, while I know that I have enough theorem prover proof writing and LLM prompt engineering skills to try to come up with the right generalization for the theorems I care about (in theoretical economics - folk theorems, in computer science - new proof-theoretic meta-theorems and deduction algorithms, etc.) but I am pretty sure that anything I come up with will be very difficult to publish due to my aesthetic preferences for category-theoretic cleanness without name-dropping the right theorems from the literature (Banach-Alaoglu could be said to be the \*Nessun dorma\* of quantitative mathematics, maybe Levin's universal search, Henkin constructions, Lawvere's fixed-point theorem is the \*Climb your mountain\* - from \*The Sound Of Music\* and I am still looking for something that deserves the title of the \*Defying Gravity\* of abstract but useful mathematics, excluding stuff like number theory), Elphie

Here are a few of the possible candidates for the \*Defying Gravity\* of the mathematical deep-theorem world:

1. Minimal-regret ML algorithms for situations with hidden moves, rules, or outcomes, plus proven optimal complexity and regret bounds

2. Folk theorems (a continuous action space variant was proven in 1995) but this is more like some of Menken's songs like Somewhere that's green due to people restricting to finite action spaces most of the time, without much mathematical beauty, not really Defying gravity

3. Byzantine-fault and other consensus algorithms with convergence and optimality guarantees (saying that if you remove one of the conditions for convergence, the system becomes manipulable)

4. Universal algorithms like AIXI, Levin's search procedure, Garrabrant's logical induction, syntax and semantics (completeness + soundness) for fuzzy logics, epistemic logics, paraconsistent logics, etc.

5. Folk theorems for technically-involved assumptions (an under-studied field, maybe mixed strategies are represented as points on the same convex hull, such that all strategies are continuous, bonus points if mixed strategies as probability distributions on the support of the continuous space of actions are allowed, private monitoring is allowed, or when the game/nature is allowed to choose a state from a probability distribution before each round)

6. Of course if fans love comparing Elphabas I also love comparing deep theorems

In addition:

1. Categorifications and generalizations of existing theorems, such as David Michael Roberts' generalization of Lawvere's fixed-point theorems to magmoidal categories (this may make proving many universality or impossibility results way easier because we can then embed stuff in magmoidal categories instead of CCCs)

2. Qualitative instead of quantitative definitions of "more/less information" such as weighted garbling replacing Blackwell garbling, generalizations of Aumann's agreement theorem to prior-free setups, prior-free ML, etc.

3. Harvey-van der Hoeven's O(d log d) [d = digits] algorithm for integer multiplication (optimal, useful), AKS for primality testing (deterministic, very useful for field arithmetic), these unexpected efficient deterministic algorithms may make proving polynomial-time or log-linear-time optimality of many derived algorithms for deterministic ML, reversible computing, etc. easier, I am waiting

4. Mayer Goldberg's result that fixed-point combinators are recursively enumerable (both strong and weak versions, by successive reductions of random lambda terms to forms that satisfy the equations and reduction chains of `fix f` to `f (fix f)`), imagining knowing every single way to loop, every single implementation of the most important construct in programming languages that makes programming programming

5. My proposal: Inspired by Mayer Goldberg's success in finding a single procedure that enumerates all the fixed-point combinators, I may want a procedure that enumerates all the neural architectures that satisfy the UAT, Turing-completeness, that is, how to generate code that can ensure that any program-synthesis, machine-learning, whatever computational system that is learnt from data will be universal in the way I want, I am trying to find time to work on this

6. Incentive-compatible public-good provision mechanisms and algorithms, especially those for many public goods that add up, Cobb-Douglas and Leontief are well-known but I want something for public goods that stack up linearly, that's the real defiance of gravity