NOTE: unfinished draft

Combining LLM and RL, and Logic Transformer

[redacted]

[redacted]

Abstract. This paper has two main ideas: The first is to explain two types of LLM (large language model) + RL (reinforcement learning) architectures, which are probably known on the internet but not explicitly stated. The second idea is developed based on what we call "Type L" architectures. Under this perspective, formal logic is essentially the same as natural language, and so LLMs (language models) are the same as logic processors. Thus we introduce the Logic Transformer which may have advantages over traditional Transformers.

Keywords: AGI · large language models · reinforcement learning · neural-symbolic integration

Part I. LLM + RL architectures

For **string diagrams** [2] there are usually two conventions: 1) data are nodes, functions are edges: $x \xrightarrow{f} y$ or alternatively 2) functions are nodes, data are edges: $\xrightarrow{x} f \xrightarrow{y}$. In the following, I make explicit nodes for both functions (grey) and data (white), whereas edges merely represent linkages.

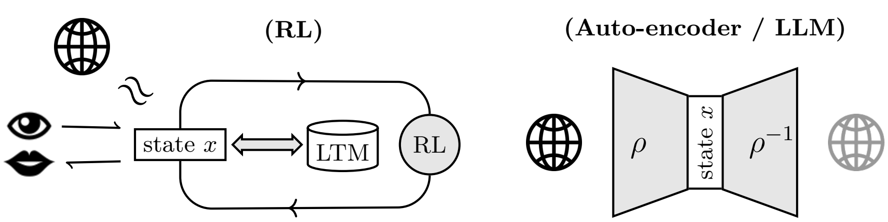


Fig. 1. The eye represents observations and the mouth (speech) actions. Because RL has to maximize rewards, its internal representation (the state) must eventually approach a good approximation of the world. LTM = long term memory, which works by associative recall and condensation, but will not concern us in this paper. The auto-encoder, of which LLMs are a special case, works by compressing world-data (via ρ) and de-compressing (via ρ^{-1}) to re-construct the data (grey world).

First let's recall the **fundamental forms** of RL (Fig.1 left) and auto-encoders (right). The question is how to combine them, for which we can ignore the LTM module. Both RL and LLM have the "state" x in common but there are subtle differences as to how to interpret these states (Fig.1). Initially my research favored **Type W**, where the LLM is interpreted as a **world model**. This approach suffers the problem that we don't really understand the internal state (marked "?") of LLMs, which consists of many layers of Transformers and their intermediate token outputs, so we don't know how to "merge" the RL state with the LLM state.

It seems that **Type L** would be easier to work with, and it is also the "mainstream" approach, along the direction of RLHF [**RLHF**]. Here, the LLM simply loops over itself to complete the RL cycle. LLM models the "spoken thoughts" of human thinking, as found in text corpora. These thoughts are interpreted as internal states of the RL. Ever since Richard Montague's success in converting a fragment of English into formal logic [**Montague**], the line between formal logic and natural language is blurred. We should not obsess with the idea that "logic" must be some kind of cryptic, undecipherable code. A new perspective: *the "representation" of human internal thought is just natural language* (or very close to it). Type L also has the advantage that we can directly examine the internal state of an AGI. What are currently called "prompts" is just **Working Memory**.

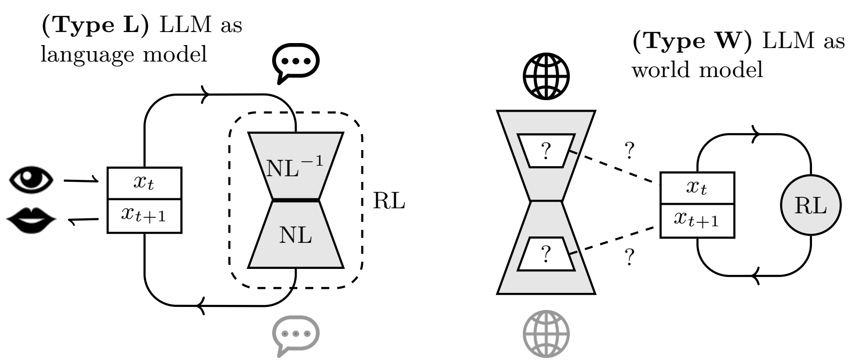
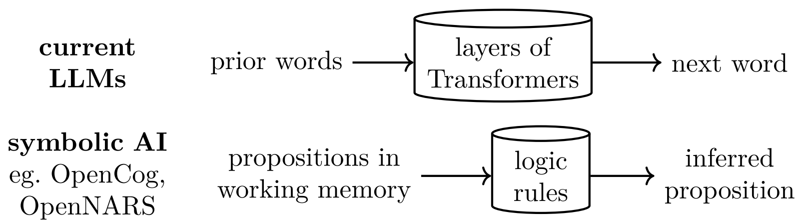


Fig. 2. Two types of RL + LLM architectures. NL^{-1} is a map that compresses natural language to a hidden representation (not shown explicitly).

Part II. Logic Transformer

As we know, LLMs predict the next word per each iteration. The number of possible natural-language sentences is an astronomical figure (if not ∞), so it is infeasible in practice to represent probability distributions over this set. However, as vocabulary size is finite, the conditional probabilities $\mathbb{P}$(next word | prior words, context) are manageable. This is partly why LLMs are so efficient – exploiting the decomposition of sentences into words.

For a long time I tried to analyze the Transformer (Self Attention) from a symbolic-logic point of view, but did not know how to interpret the meaning of **tokens** – do they correspond to logic propositions, or to sub-propositional symbols? The question is how to reconcile these two views:



Finally I had a break-through insight that a symbolic-logic Transformer should output **atomic propositions** that modify a linguistic tree **at one position** at a time (Fig.3). Tree structure simplifies the implementation of logic rules – fewer parameters needed; learns faster. An analogy is that humans try to parse sentences syntactically first (which requires only superficial information such as parts-of-speech) before deeper semantic understanding. In order words, grammar *simplifies* subsequent understanding.

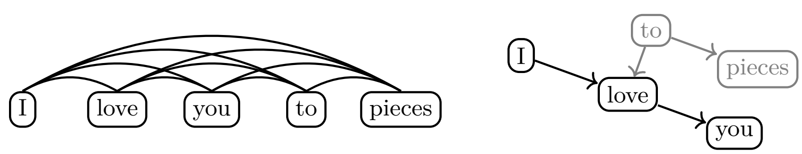


Fig. 3. Left: In traditional Self-Attention Transformer, each token is related to every other token, ie. fully connected graph. **Right:** Example of a linguistic tree, ie. a connected graph without loops or double edges, here also with directions added to form a DAG, directed acyclic graph. From a **Chomskyan** perspective, any NL sentence can fit into a tree structure. This example starts with a simple relation, I-love-you, and the "love" node is modified by a "to" clause (gray). Let's not dwell too deep into this example, as machine-learning will fill in such details.

0.1 Knowledge Representation

We want to endow the structure with

- Tree structure as inspired by Chomskyan linguistics
- Ontology such as "kingdom-phylum-species-genus" in biology or a library catalog system.
- The (probabilistic) strength of a thought as the length of the thought vector.

0.2 Rete

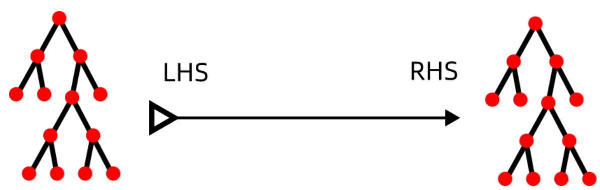
0.3 Graph-rewriting

In [1] [**Goertzel2021b**], Goertzel formulates intelligence as the re-writing of some metagraph or hypergraph structure, which is essentially the same as the author's idea of tree as Working Memory and which I now link to the Transformer. Rewriting is defined categorically as certain morphisms, requiring some diagrams to commute. This defines intelligence in **algebraic** terms, and it requires the discretizing functions above to achieve such algebraic rewriting.

Relation algebra [4] [5].

The hallucination problem. The significance of combining RL and LLM is such that the system can *think in loops*, eliminating thoughts that contradict accepted truths, thus achieving genuine, logically coherent understanding, instead of just parroting human speeches. Reward maximization in the RL loop forces better ideas to win, ie. those ideas that are logically coherent with the rest of the knowledge. This would solve the problem of "hallucinations". The remaining question is how to train the Type L model *efficiently*, but this is more of an engineering rather than theoretical problem.

Update: need to handle **variable-length propositions**. Need tree-rewriting logic. In a certain direction, the neural network may end up with a form very close to or even identical to current Transformers.



Now we are ready to re-formulate the Transformer as applying a set of logic / symbolic rules to its inputs. In order for such rules to be **differentiable**, we can't have N rules at one moment and then suddenly $N+1$ rules the next moment (that would not be differentiable). The only solution I can think of is to maintain a *fixed* number, M, of rules. Every rule may potentially "morph" into any other rule in this space of rules. The set of rules would look like a *rectangular* matrix: (This part needs modification for tree-based logic rules instead of fixed-length rules; Tree-based rules require special syntax. This may not be too difficult, as tree-rewriting is equivalent to term-rewriting, which the author already has some familiarity with. [Goertzel 2021] also developed a "reflective metagraph rewriting as a foundation for an AGI language of thought")

For a rule to be applicable, its atoms must match with propositions in Working Memory. An atom may contain constants (such as "Socrates", that must be matched correspondingly) or variables (such as x, that matches any entity). To achieve this, I introduce a trick called **cylindrification factor** (Fig.4).

$$\gamma = \begin{cases} \approx 0 \Rightarrow \text{constant, do matching, TV} = \text{match degree} \cdot \oslash(1-\gamma) \\ \approx 1 \Rightarrow \text{variable, do substitution, TV} = 1 \cdot \oslash(\gamma) \end{cases} \tag{1}$$

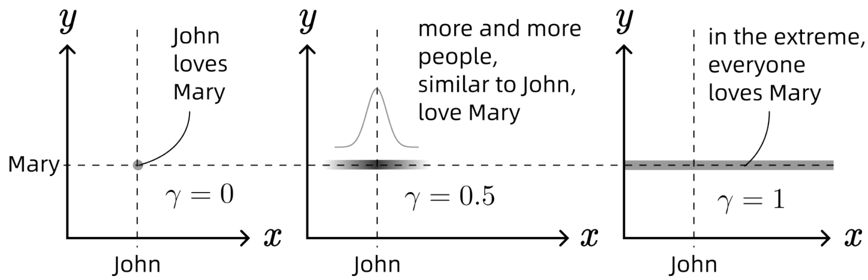


Fig. 4. Illustration of the cylindrification factor γ.

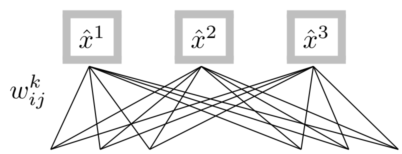
Each position inside a rule can be either a constant or variable, and their treatments are obviously very different, but the results of the two streams would be super-imposed (added) together, pretty much like the **superposition** of quantum states.

Transformer Token Vectors Are Not Points in Space.

To make **variable substitution** differentiable is even trickier. Consider this logic rule:

$$\mathsf{father}(X_1, X_2) \wedge \mathsf{father}(X_2, X_3) \rightarrow \mathsf{grand\text{-}father}(X_1, X_3) \tag{2}$$

The actual implementation leads to formulas like these:



$$\mathsf{father}(x_{11}, x_{12}, \bullet_{13}) \wedge \mathsf{father}(\bullet_{21}, x_{22}, x_{23}) \rightarrow \mathsf{grand\text{-}father}(x_1, \bullet_2, x_3) \tag{3}$$

$$\hat{x}^k := \underset{\text{predicates}}{\forall i \in} \; \underset{\text{arguments}}{\forall j \in} \left\langle \operatorname{soft}\max_{ij} w^k_{ij} \;,\; x_{ij} \right\rangle \tag{4}$$

which is similar to the **Self-Attention** of Transformers:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \operatorname{soft}\max \frac{\langle \mathbf{Q}, \mathbf{K} \rangle}{\sqrt{d_k}} \mathbf{V} \tag{5}$$

except that softmax does not commute with inner products, so they are not exactly equivalent, but qualitatively similar. Let's pause for a moment to reflect that we tried to substitute variables (some kind of syntactic manipulation), and ended up with something like Self-Attention. Also recall that Self-Attention originated as a kind of **content-addressable memory** for Neural Turing Machines, invented for its differentiability.

A final obstacle concerns the limitations of first-order logic. As Ben Goertzel pointed out during an online discussion, my earlier design lacked mechanisms for handling **Skolem functions** and thus existential quantifiers. One way to get around this problem is to define ∃ via a set of **higher-order logic** rules. Without explicitly specifying how to do so (though I believe this can be done), we simply use a more powerful "higher-order" logic in the sense that rules admit substitutions in all (predicate or argument) positions, and hope that machine learning will learn the required axioms.

Armed with these ideas, it is not too difficult to work out a new kind of differentiable "Logic Transformer" (Fig.**??**). A detailed explanation is contained in my on-going research thesis [**YKY**].

Conclusions

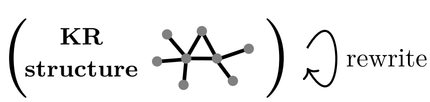
I guess I am not the only one among AGI researchers who is jealous of the success of the Self-Attention Transformer BERT-GPT lineage tradition. Many like us suspect that Transformers are unnecessarily "bulky", meaning that they can be reduced or simplified. But how?

String diagrams allow us to visually compare architectures, such as Self-Attention with my differentiable Logic Transformer. These diagrams can be "morphed" by rewriting rules. One research team proposed universal approximators as an equivalence class of architectures under rewriting, but such a class is too permissive and pretty useless. We want a more refined view of architectures such that we can distinguish which ones are more **learning-efficient** – the key question in AGI development.

If all continuous functions are equivalent, then what sets them apart? My idea is that functions such as sigmoid and softmax are "tearing" the input space irreversibly, due to finite precision, into **discrete** regions that are no longer connected. This process is crucial to eg. Transformer's ability to output discrete tokens.

In Ben Goertzel's "General Theory of General Intelligence", he formulates intelligence as the re-writing of some metagraph or hypergraph structure, which is essentially the same as my idea of tree as Working Memory and which I now link to the Transformer. Rewriting is defined categorically as certain morphisms, requiring some diagrams to commute. This defines intelligence in **algebraic** terms, and it requires the discretizing functions above to achieve such algebraic rewriting.

But the Transformer has also been shown capable of discrete (syntactic) manipulations. Does that mean that all functions with discretizing ability are equivalent? Can the Transformer be replaced by a fully-connected NN with sigmoid activation? Seems unlikely, and if not, then what sets them apart *still*? Well, an obvious answer is that we need to **count computations**.



(6)

Functor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$.

For example, in the "double-pushout" approach [3]: a rewrite rule is a pair of morphisms $L \leftarrow K \rightarrow R$, L and R are the left and right hand sides, and K is the interface graph. G is the input graph and H is the result of rewriting. The effect is that the part of L outside K is deleted from G and is replaced by the part of R outside K:

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array} \tag{7}$$

This occurs in a category where graphs are objects and homomorphisms define some notion of "matching". Now in a **differentiable** version of rewriting, all the rules are represented by a set of parameters $\Theta \in \mathbb{R}^n$, and the parameterized rules *must* cover the entire family of rewrite rules as Θ vary continuously (in practice, we can limit the rules length to make that possible). This requires the use of **discretizing** functions such as sigmoid and softmax to achieve, and may even be formulated by a fairly mechanical process. However, in doing so we have overlooked the possibility that a **deep** neural network such as Transformer may be able to aggregate all the contents of Working Memory and output some intermediate, **distributive** representations before finally outputting some discrete tokens. Such a process would have no counterpart in the form of discrete rewriting rules such as (7).

References

[1] Ben Goertzel. *The General Theory of General Intelligence: A Pragmatic Patternist Perspective*. 2021. arXiv: `2103.15100 [cs.AI]`. URL: `https://arxiv.org/abs/2103.15100`.

[2] Ralf Hinze and Dan Marsden. *Introducing String Diagrams - the art of category theory*. Cambridge University Press, 2023.

[3] Kennaway et al. "An Introduction to Term Graph Rewriting". In: *Term Graph Rewriting - Theory and Practice*. Ed. by Sleep, Plasmeijer, and Eekelen. John Wiley & Sons, 1993.

[4] Roger Maddux. *Relation algebras*. Elsevier, 2006.

[5] Gunther Schmidt. *Relational mathematics*. Cambridge, 2010.