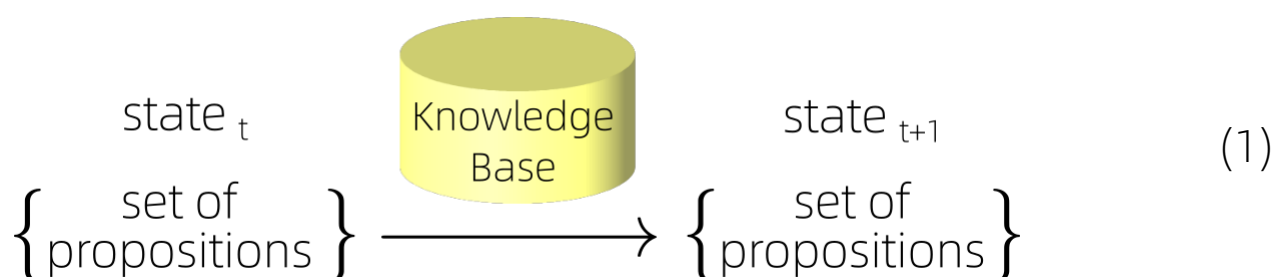


①

Comparison of Logic AI and Deep Learning

This is the most basic operation of classical logic-based AI:



It contains two algorithms:

- **matching** (unification):
Logic rules are conditional propositions involving variables,
eg: $\forall x. \text{human}(x) \Rightarrow \text{mortal}(x)$.
Unification determines whether a rule can be applied to a proposition,
eg: $\text{human}(\text{Socrates})$ can unify with the left side of the above rule.
The goal of Matching is to get an instantiated proposition (ie, specialized, does not contain variables).
- **forward- or backward-chaining** (resolution):
Deduce new conclusions from known facts, or conversely, judge whether a given conclusion can be proven.
eg: $\text{human}(\text{Socrates}) \wedge \text{human}(\text{Socrates}) \Rightarrow \text{mortal}(\text{Socrates})$
From which can be deduced: $\text{mortal}(\text{Socrates})$.

The special thing about deep learning is that it can imitate this inference process:

$$\text{state}_t \vdash \text{state}_{t+1} \quad (2)$$

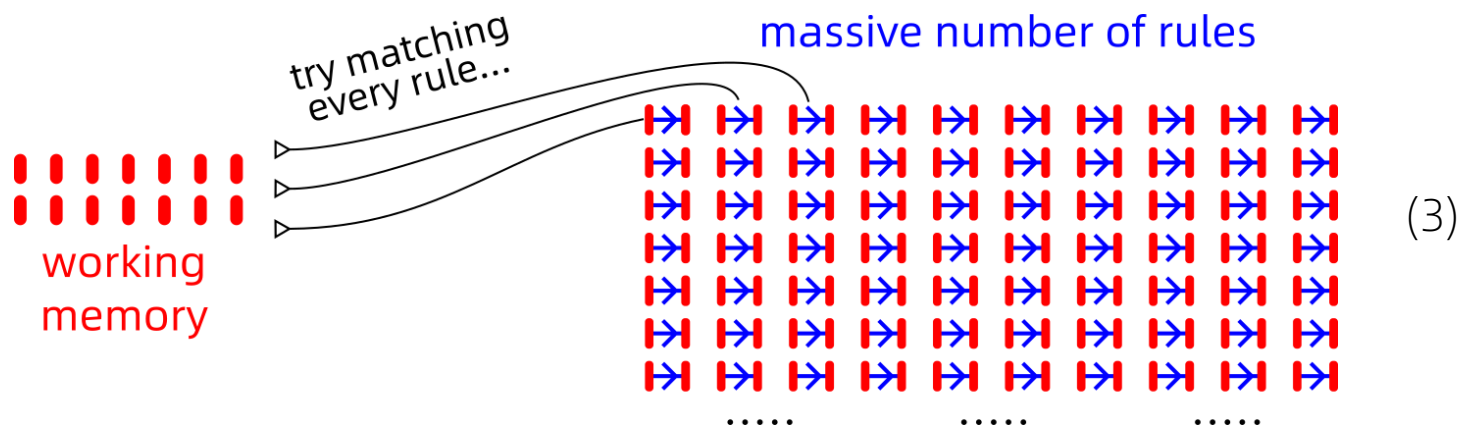
with a highly complicated non-linear function (ie, a deep neural network). By doing so, the logic rules are “mingled” together so that it’s hard to tell them apart. But it is precisely because of this “mingling” that a deep neural network compresses a huge number of combinatorial logic rules into a smaller number of parameters (network weights). It can perform both learning and inference. This simple and crude method is actually extremely efficient, and it is not easy to surpass its speed!

We know (or speculate) that an intelligent system should possess the structure of symbolic logic. Can this insight be used to constrain or accelerate deep neural networks? The answer seems to be yes. The current state-of-the-art CNN (for vision) and GPT (for language) both have specialized internal structure, instead of just being **fully-connected**, and that internal structure is suited to the structure of the data being processed. We have reason to believe that logical structure can be used to constrain deep neural networks to accelerate logical learning.

②

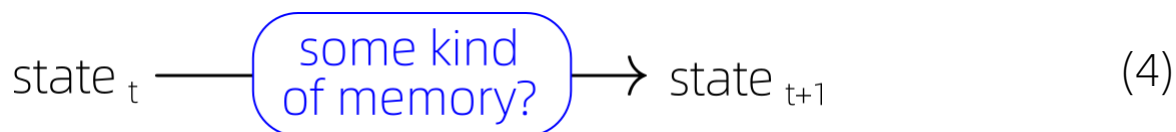
Next, let's look at the logic-based AI architecture in detail.

There are a huge number of rules in the Knowledge Base, and the system needs to match these rules one by one against propositions in the system's state (= working memory):

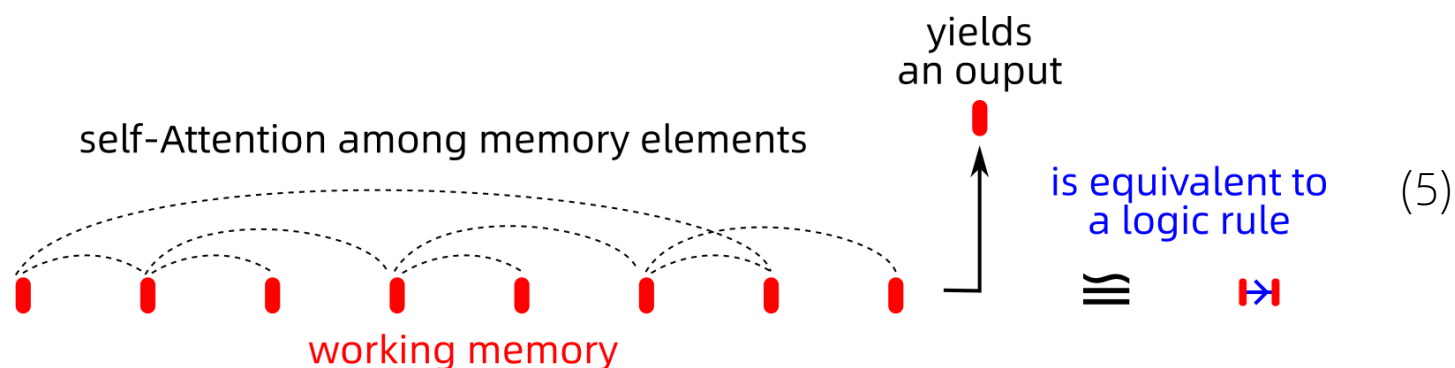


Successfully matched rules generate new conclusions that can be added back to the state / working memory.

This complicated process is entirely replaced by a neural network. Or more abstractly:



For the Transformer, this is a kind of memory stored **between** input elements (stored as the Q, K, V matrices), and it **implicitly** plays the role of logic rules:



In other words, there are some sort of "distorted" logic rules inside the Transformer. Naturally, we want to find out more structures of logic / logic-based systems. That is, what kind of algebraic structure constrains (4)? To answer this, we can take insights from categorical logic and classical logic-based AI.

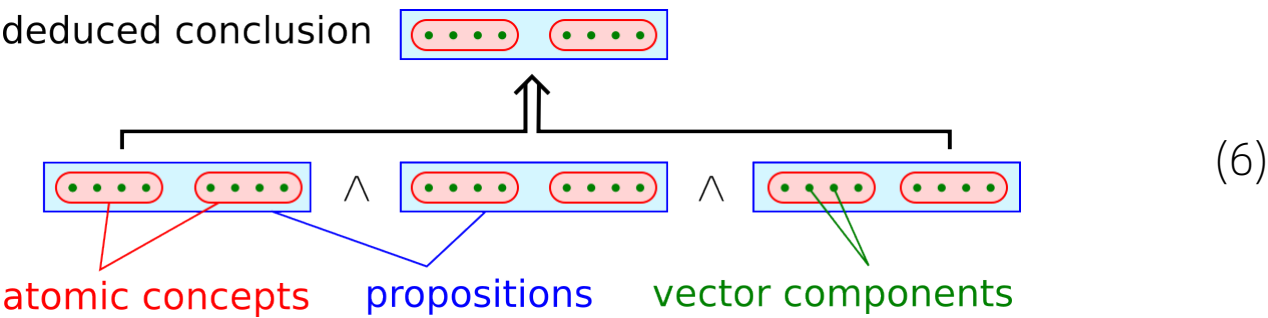
We wish to formulate, in algebraic terms, the constraints for (4), but for now it is easier to describe in words:

- The AI state is **granular**, as a set of elements, whose order can be permuted, corresponding to equivariance of the Transformer. (Note: Transformers are equivariant, but equivariance does not necessarily require Transformers)
- **Deep structure**: in the sense of having many layers, as composition of functions. The Transformer also has deep structure, with many layers of Self-Attention stacked up.
- Logic has granularity at the **proposition** level and at the **sub-propositional** level. The latter is the structure of **predicate** logic, eg: *loves(John, Mary)* can be represented as a **product** in an **algebra**: *John • loves • Mary*, also called a “word”. The details are unimportant. Our focus is on how to impose this 2-level granular structure onto deep neural networks.
- Each step of logic inference produces **one** new conclusion (or a probability distribution over conclusions), and this new conclusion is added to the old state as an element in a set of propositions, and the old state also needs to **forget** some old propositions, otherwise infinite memory is required. This is slightly different from the Transformer which always outputs the **same number** of tokens as its input. (We are unsure whether Transformer tokens correspond to propositions or to predicates / atomic concepts ¹).
- A logic rule usually depends on some premises where other premises are **irrelevant**; For example: *talk ∧ dark ∧ handsome ⇒ attractive to women*, where *rich* or *poor* are irrelevant. The Transformer’s **softmax** seems to be a mechanism to exclude irrelevant tokens.
- (There may be other structures.....)

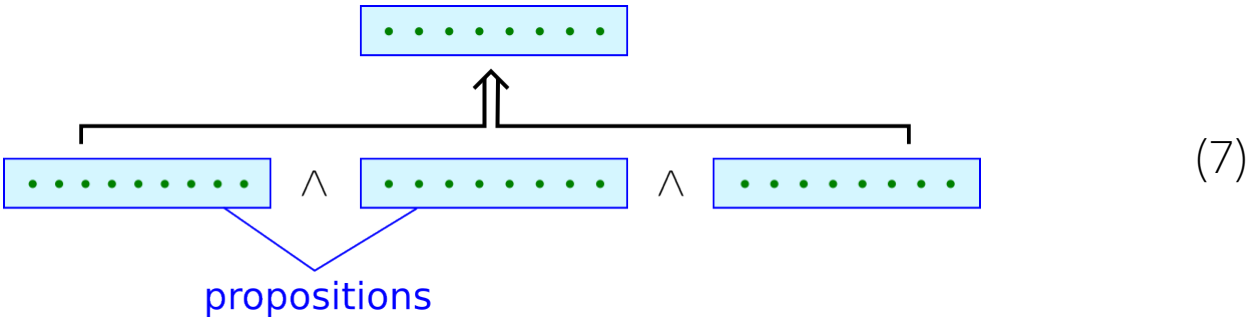
¹This is answered in the next version


4

In my theory, the ideal logical structure is something like this (the numbers of elements may vary):

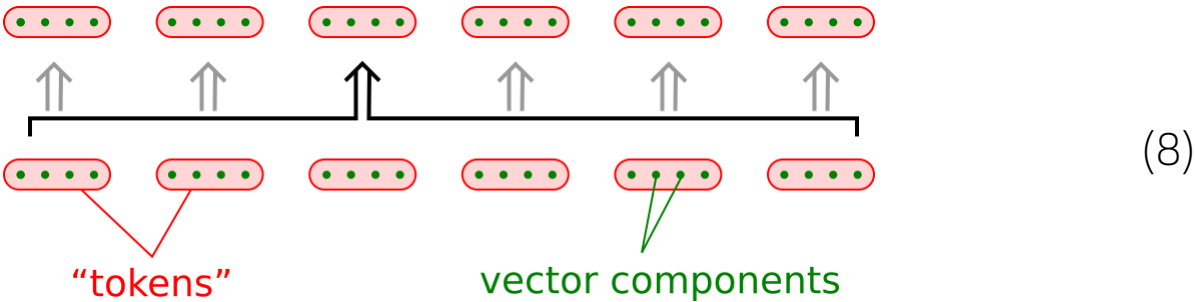


In contrast, the algebraic relation $p \wedge q = p \wedge q$ expresses only this structure:



Compared to figure (7), figure (6) is additionally constrained by the  structure. But how can we express this constraint algebraically?

And this is how the **Transformer** handles propositions and atomic concepts:



It does not represent propositions (= sentences) explicitly, but it uses a special “stop” token to signify the **end** of sentences, and there are other “tricks” such as **positional encoding**. It seems to be a rather *ad hoc* design, we should be able to improve it.

5

Now we try to answer the crucial question: how to express algebraically that “propositions are composed of conceptual atoms”?

That is to say, what is the difference between the following two structures? How to express this difference algebraically?



This is like asking the difference between $0\dots9 \times 0\dots9$ and $00\dots99$ (they are isomorphic).

Similarly,

$$\{ \text{John, Mary} \} \times \{ \text{human, god, worm} \} \quad (10)$$

and the $2 \times 3 = 6$ propositions

$$\{ \text{John is human, Mary is human,} \} \quad (11)$$

are also isomorphic. But the former is a composition of two different concepts, where components can be individually quantified by \forall or \exists ; The latter is propositional logic, where propositions are indivisible and cannot be internally quantified.

But since \dots belongs to a non-commutative free group (ie, the group with the least structure), it does not possess a simple symmetry like $a \cdot b = b \cdot a$.

After some analysis I arrived at the following condition for “propositions are made of conceptual atoms”:

Atomic Condition (AC). *Each proposition P_i is made up of K atoms:*

$$P_i = a_{i1} \cdot \dots \cdot a_{iK} \quad (12)$$

where optionally some atoms can be **copied** to other locations (with a non-linear transformation τ , if they are copied to the output layer) via:

$$a_{ih} = a_{jk} \quad \text{or} \quad a_{ih} = \tau(a_{jk}) \quad (13)$$

and the transformation τ has to accord with \forall or \exists as adjunctions to a substitution functor.

The essence of the Atomic Condition lies in the two **equations** (12) and (13), which are actually very simple. The category-theoretic description of \forall and \exists as adjoint functors is quite advanced, but not essential, and we shall explain them in appendix A. In fact, τ only needs to be a continuous function to meet the above requirement.

So where does the “=” in equation (13) come from? In fact, it is too obvious, it is just the action of syntactically “moving” variables in a logic rule, eg:

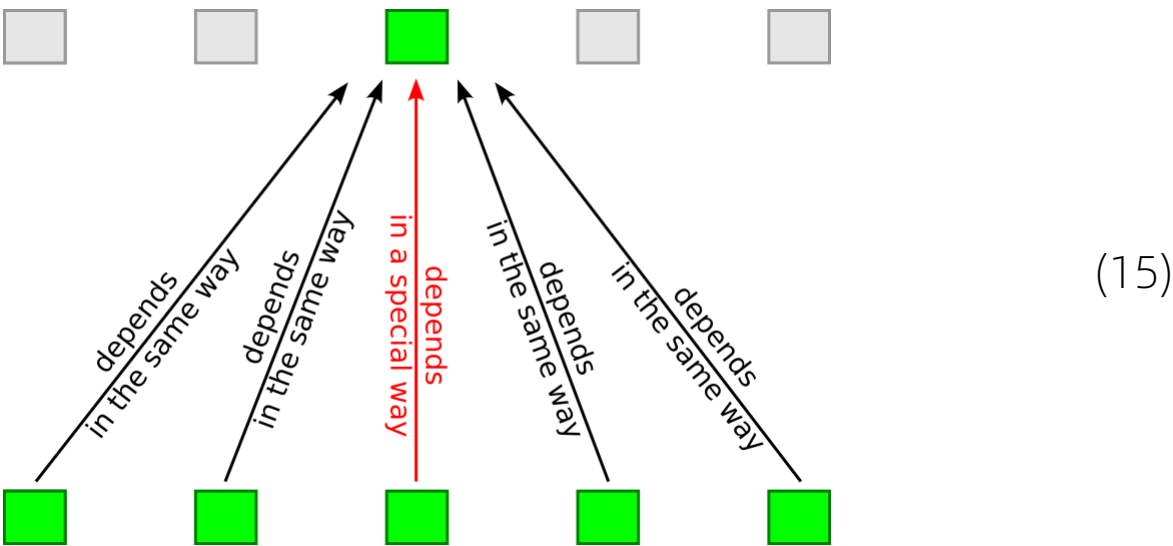
$$\forall X, Y, Z. \text{ grandfather}(\textcolor{red}{X}, \textcolor{red}{Z}) \leftarrow \text{father}(\textcolor{red}{X}, \textcolor{red}{Y}) \wedge \text{father}(\textcolor{red}{Y}, \textcolor{red}{Z}) \quad (14)$$

It is due to such “movements” that gives rise to the structure “propositions as composed of atomic concepts”.

6

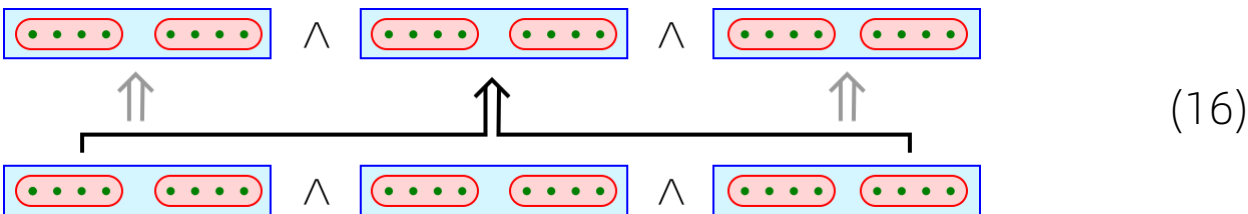
Logic and Deep Learning

The essence of **Self-Attention** can be understood as follows (what I call abstract Self-Attention):



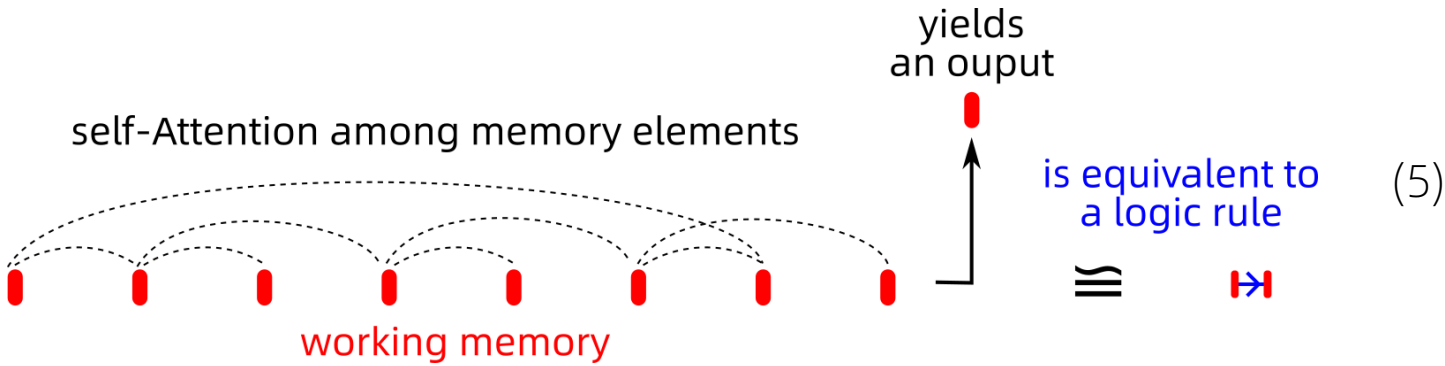
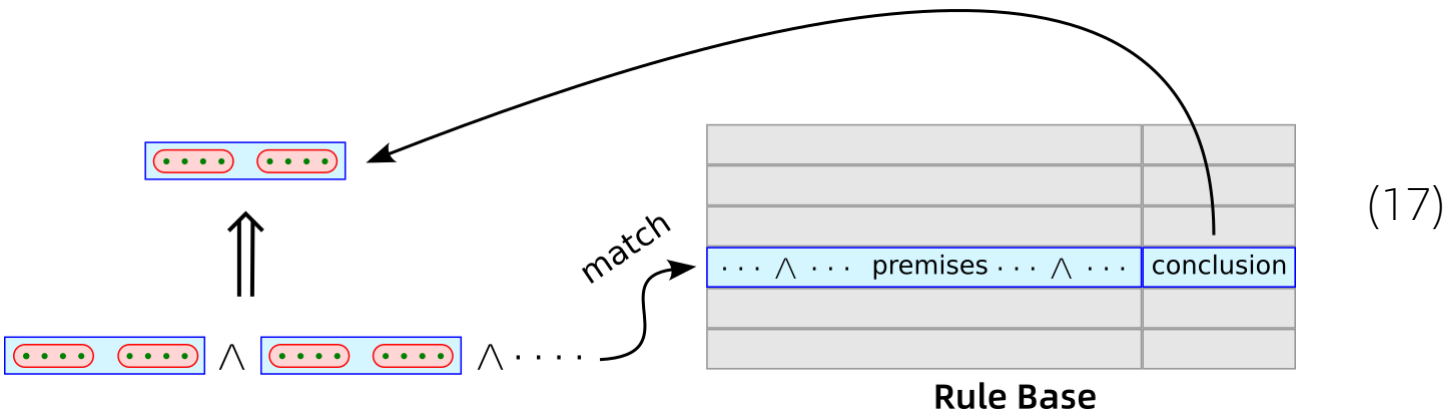
This vertical “axis” (red) is repeated per each input token. So, when the input elements are **swapped**, the output is also swapped. This is how Self-Attention achieves **equi-variance**.

We want to apply a similar method to logical structure:



Here is a very important point¹: The precursor of Self-Attention is the **content-addressable memory** from Graves *et al*’s “Neural Turing Machines” ². We have good reason to regard Self-Attention as a form of **mem-ory**.

Let’s compare the two approaches. One is the naïve classical rule-base structure, and the other uses Self-Attention instead of a rule base:



You should get the sense that the Transformer is a very “twisted” way of representing logic rules. The correspondence is so indirect that it is difficult for us to see what the rules look like inside the Transformer. However, I think the designers of Transformer might have had at least an inkling of its similarities to rule-based systems. In particular, look at the following logic rule:

$$\forall X, Y, Z. \text{ grandfather}(X, Z) \leftarrow \text{father}(X, Y) \wedge \text{father}(Y, Z)$$

(14)

The premise of this rule has two clauses, the variable **Y** that appears twice must be identical (red), for this matching to be considered successful. And this kind of **comparison** operation within the premises of the rule is exactly what Self-Attention can perform conveniently.

¹Thanks to “Ziyu” for telling me this.

²The book “Fundamentals of Deep Learning” [Buduma & Locascio 2017] has a very nice explanation of Neural Turing Machines.

Some key questions to be answered:

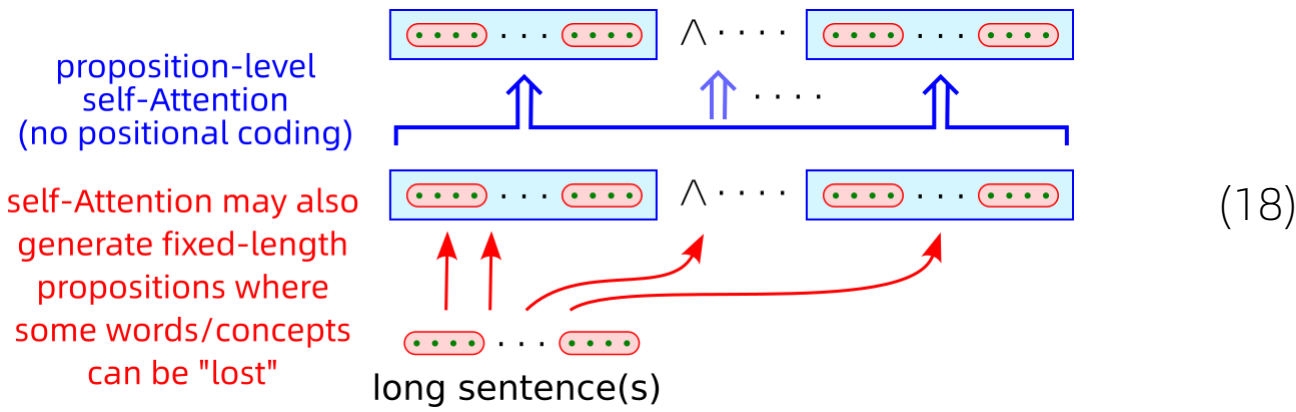
- According to eg. Qian Liu's paper ¹, Transformers often fail at certain logic and syntactic questions. Where is the problem? It is not the case that Transformers cannot learn the syntax at all, but it cannot do so with prompts only. Prompts are just a hack to encode contextual meaning in tokens. It seems that this contextual meaning is sometimes insufficient to answer certain questions. If we have not explicitly "told" the Transformer what it should do, is it a real shortcoming that it failed to solve those problems? This is a problem with Language Models which I think logic-based AGI will provide a clearer solution.
- Now consider the naïve rule-base algorithm in diagram (17). This algorithm is of course very slow, as we need to compute the similarity between two sets (working memory and rule heads). Assuming that the size of the sets are fixed at M and N , then $M \times N$ dot products need to be computed, and this is for just 1 rule. The result from all rules need to be added with softmax. When the rule base is very large, this seems impractical.
- Diagram (17) may suffer from an old problem of logic rule-learning, known as the "**plateau problem**". For example, the "append" function in Prolog:

```
append(X,Y,Z) :-
    list(X), head(X,X1), tail(X,X2), append(X2,Y,W), cons(X1,W,Z).
```

This rule has 5 premises. As the rule is learned, premises are added one by one, but the "score value" of the rule remains zero until the very last premise is added, and the score suddenly jumps to 100%. For machine learning, this situation is terrible. As the Transformer "mingles" logic rules together, it may avoid being trapped in local minima.
- Can we design an algorithm halfway between Transformer and naïve rule base, which has stronger logical structure than Transformer, but uses efficient matrix operations similar to Self-Attention?

¹Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. *Compositional Generalization by Learning Analytical Expressions*. Advances in Neural Information Processing Systems 33 (2020).

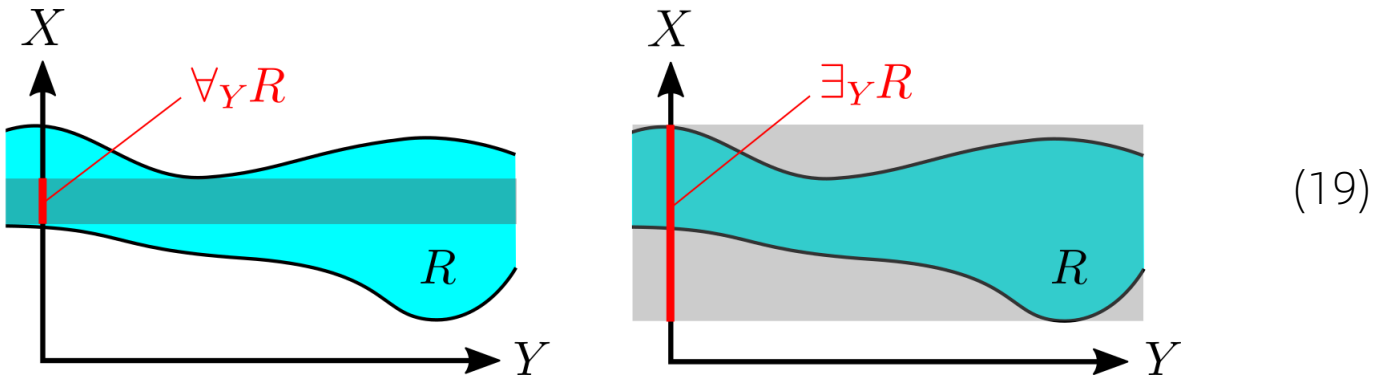
Currently I can think of a more practical architecture as follows:



- In the blue structure, each proposition shall consist of a fixed # of concept atoms.
- Abstract Self-Attention allows variable length propositions, as long as we can calculate the dot product (similarity) between propositions, and if we know how to retrieve from the (fixed-length?) memory matrix.
- Red structure: We need to read a natural-language sentence and break it down into a (variable) number of propositions (of variable lengths). This is the “sequence to sequence-of-sequences” problem that I have been talking about for some years. This may be solved with “lossy” Self-Attention.

In this appendix, we explain the categorical theory of \forall and \exists in a simple way.

The following is a relation R , such as “ Y loves X ”, where X, Y are both sets of “people”; two identical copies. For example, the diagonal represents loving oneself (some people don’t love themselves). Note that this graph is not symmetric about the diagonal, otherwise there would be no such thing as “heart break”. Projecting the relation R onto the X axis yields the \forall_Y and \exists_Y sets. The \forall_Y set represents those in X whom “everybody loves”, and \exists_Y represents those in X whom “somebody loves”:



The category theorist Lawvere discovered that \forall and \exists are **adjunctions** to the so-called **weakening functor**, which means expanding from the variable domain (X) to the variable domain (X, Y), where Y is just a **dummy** variable:

$$\begin{array}{ccc}
 & \xleftarrow{\forall_Y} & \\
 (X) & \xrightarrow{\text{weakening}} & (X, Y) \\
 & \xleftarrow{\exists_Y} &
 \end{array}
 \tag{20}$$

For example, $\text{Love}(Y, X)$ is a logic term with two variables, whereas $\forall Y. \text{Love}(Y, X)$ actually does not contain the variable Y , as it is **bound** by the quantifier \forall .

What **adjunction** means: Given two categories: left and right. You can move objects from the left to the right, and “**compare**” them in the right category. This comparison can also be performed in the left category by moving objects to the left. And these two ways of comparison are **equivalent**. By “comparison” we mean morphism in the category. For example, in the category **Set**, comparison is set inclusion.

Adjunctions are not unique, so weakening has two adjoints, \forall and \exists .

Lawvere’s work made the \forall and \exists quantifiers more general. The “simple” weakening functor is based on the Cartesian product $X \times Y$, but Lawvere generalized it to arbitrary **substitution** functors. (At this time I can’t give any examples of this, so I don’t know how exactly this generalization can be advantageous for our applications....)

(B)

In the study of categorical logic, there are the **Beck-Chevalley** condition and **Frobenius** condition. Could they be the conditions I was looking for? Upon closer look, I found that they are not the solution. For completeness, I will describe them, but the reader can skip if they're not interested.

The **Frobenius** condition seems easier to understand. Logically, it is equivalent to:

$$\exists x. [\phi \wedge \psi(x)] \equiv \phi \wedge \exists x. \psi(x). \quad (21)$$

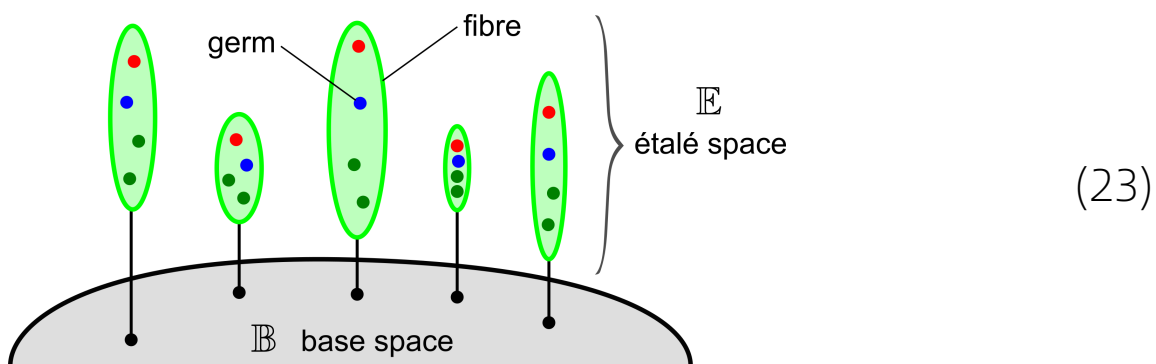
Since logic-based AI generally uses \forall and ignores \exists , I rewrite the above formula as:

$$\forall x. [\phi \vee \psi(x)] \equiv \phi \vee \forall x. \psi(x). \quad (22)$$

The problem is that the left and right sides of equation (22) correspond to identical neural networks (6). In other words, the condition is too subtle, and it does not affect the neural network we actually implement.

As my categorical logic tutorial said before, predicate logic leads to **fibration** or **indexing** constructs. The Beck-Chevalley and Frobenius conditions basically say that the fiber structure is “preserved by re-indexing functors”.

Here is a diagram of a fibration:



This whole structure is called a **bundle**, and **sheaf** is a bundle plus some topological constraint.

The **fibred product** of A and B over I can be defined on two bundles (A, f) and (B, g) , denoted as $A \times_I B$:

$$\begin{array}{ccc}
 A \times B & \xrightarrow{q} & B \\
 \downarrow p & \searrow h & \downarrow g \\
 A & \xrightarrow{f} & I
 \end{array}$$

(24)

where $h = f \circ p = g \circ q$. This is also a **pullback**.

The **Beck-Chevalley** condition says that the following diagram commutes:

$$\begin{array}{ccc}
 K \times J & \xrightarrow{u \times id} & I \times J \\
 \pi \downarrow & & \downarrow \pi \\
 K & \xrightarrow{u} & I
 \end{array}$$

(25)

where π is the projection representing the quantifiers \forall or \exists , which are adjoints to the weakening map π^* .

The Beck-Chevalley condition is not entirely vacuous; it may not hold. There is a counter-example from Pitts: Consider $X \times Y$, where $X = Y = \mathbb{N} \cup \{\infty\}$ are the natural numbers plus ∞ as top element. Y has discrete order, ie, all orders are the “=” order. A is the relationship on $X \times Y$: $A = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid x \leq y\}$. Then $\exists y. (x, y) \in A$ will be the entire set of X . If we consider the DCPO category, we require the fibration of Scott-closed subsets (ordered by inclusion) over DCPO. The condition for Scott closure of $\exists y. A$ is that it is a lower set closed under directed joins; and this Scott closure condition seems to be violated, thus causing diagram (25) not to commute. (I don’t understand the details of Scott closure)

First express the Self-Attention structure in a functional way:

output proposition O_i is composed of atoms b_i

$$O_i = [b_1 \dots b_K] \tag{26}$$

input proposition P_i is composed of atoms a_i

$$P_i = [a_1 \dots a_K] \tag{27}$$

Self-Attention

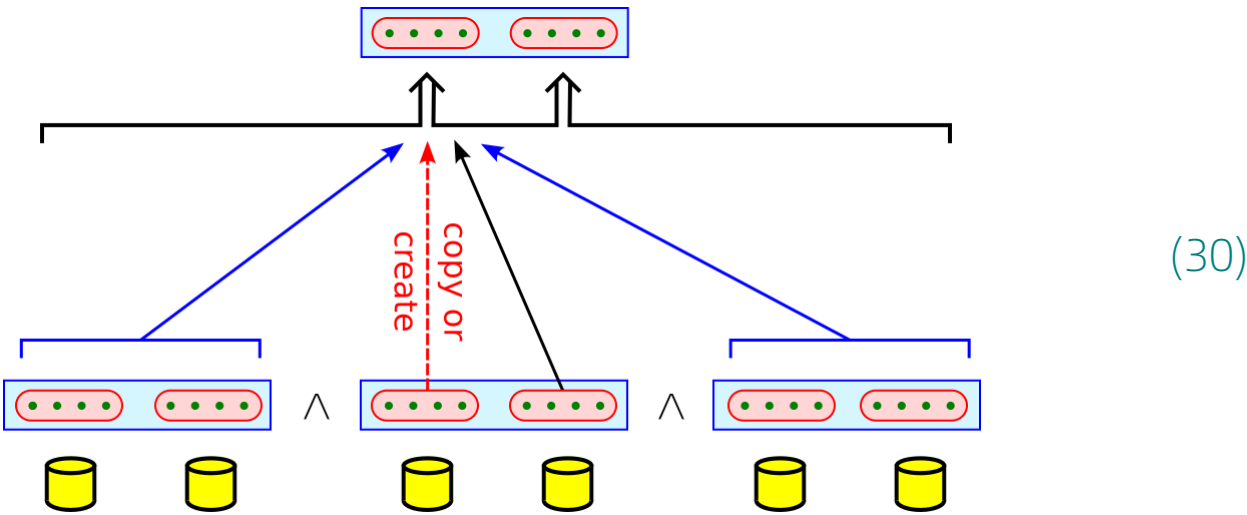
$$O_i = \alpha(P_i ; P_1 \dots \hat{P}_i \dots P_N) \tag{28}$$

$P_1 \dots \hat{P}_i \dots P_N$ means $P_1 \dots P_N$ except P_i .
 $\alpha(P_i ; \dots)$ is the function structure of the graph (15), and it can also be understood as the self Attention with P_i as the query.
How to measure similarity between (P_i, Q_i) ?

$$\arg \min_i \sum_i \min_j \langle P_i, Q_j \rangle \tag{29}$$

- First notice that the pivot structure is only useful at the propositional level, and its effect does not extend to the conceptual atomic level. However, since Transformer does not take full advantage of the exchange invariance, it becomes very efficient because it uses matrix multiplication, so from an efficiency point of view, there are also reasons to extend the axis structure to the atomic level.
- Another idea to try is: direct hard-code copying mechanism. How can this be done with Attention? It has been analyzed before, copy is not easy, because winner takes all.
- but purely using Hopfield network lacks depth. But it seems that in the RL scenario, **breadth** is also important.
In fact, as long as there is an input/output functional relationship,
- is equivalent to a KB library with logic rules. The question is what method it uses to reach its conclusions.

Self-Attention already meets our requirements, but we want to improve it. There are two main ideas: one is a more direct copy mechanism, and the other is a more detailed function dependency.



Copy:

- Copy requires no special mechanism, the output is the input. But the question is how to combine with “create” operation.
- can of course use the familiar softmax: α copy + β create, where α, β are outputs of softmax.
- Another idea is content-addressable. Use table-lookup to find executable rules. But there is a problem with variable matching.

Create:

- What kind of function is needed?