Comparison of Logic AI and Deep Learning

This is the most basic operation of classical logic AI:

$$\text{state}_t \quad \left\{ \begin{matrix} \text{set of} \\ \text{propositions} \end{matrix} \right\} \xrightarrow{\quad \text{Knowledge Base} \quad} \text{state}_{t+1} \quad \left\{ \begin{matrix} \text{set of} \\ \text{propositions} \end{matrix} \right\} \tag{1}$$
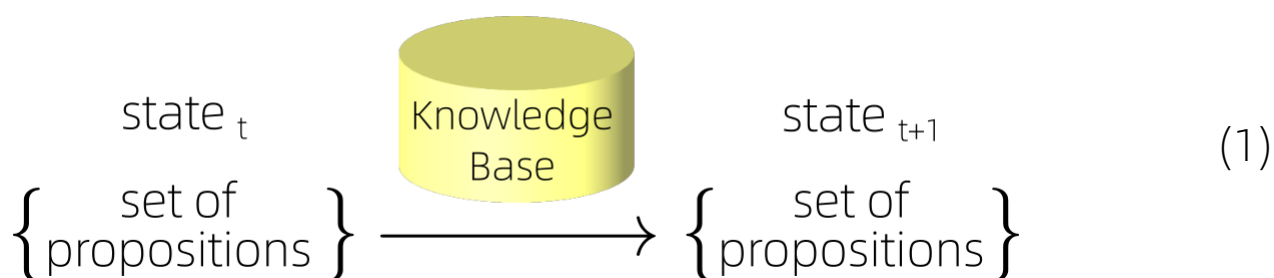
It actually contains two algorithms:

- **matching** (unification):
  Logical rules are conditional propositions involving variables,
  For example: $\forall x. human(x) \Rightarrow mortal(x)$.
  Unification determines whether a rule can be applied to a proposition,
  For example: *human(Socrates)* can unify with the left side of the formula.
  The result of Matching is to get a proposition that is instantiated (= specialized, ie, 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.
  For example: *human (Socrates) ∧ human(Socrates) ⇒ mortal(Socrates)*
  From which can be deduced: *mortal (Socrates)*.

The special thing about deep learning is that it imitates the inference process

$$\text{state}_t \vdash \text{state}_{t+1} \tag{2}$$

with a very complex non-linear function (ie, a deep neural network). After doing so, the logic rules are "mingled" together so that it's hard to tell them apart. But it is precisely because of this "intertwining" that the deep neural network compresses a huge number of combinatorial logic rules into a smaller number of layer-by-layer parameters. It performs 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 knowledge be used to constrain/accelerate deep neural networks? The answer seems to be positive. The current state-of-the-art CNN (for vision) and GPT (for language) both have specialized internal structure, **rather than fully-connected**, and this internal structure corresponds to the structure of the data being processed. Therefore, we have reason to believe that logical structure can be used to constrain the structure of the deep neural network to achieve acceleration.
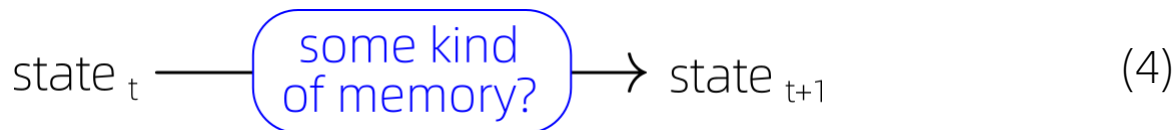
Next, let's look at the structure of logical systems in detail.

There are many 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):


(3)
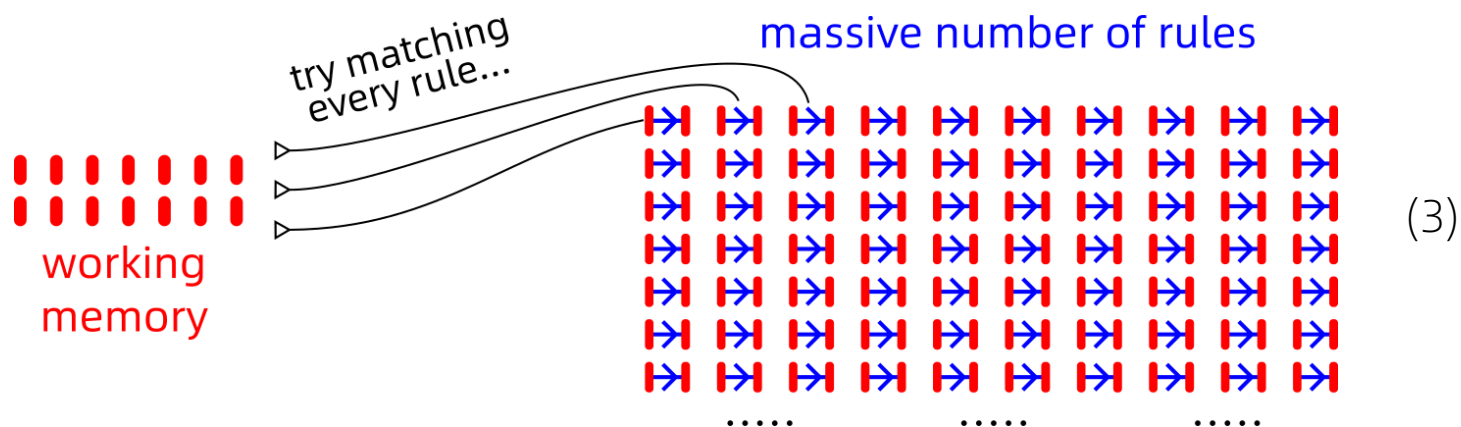
Successfully matched rules can derive new conclusions and add them back to the state / working memory.

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

$$\text{state}_t \longrightarrow \boxed{\begin{array}{c}\text{some kind}\\\text{of memory?}\end{array}} \longrightarrow \text{state}_{t+1} \qquad (4)$$

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


(5)
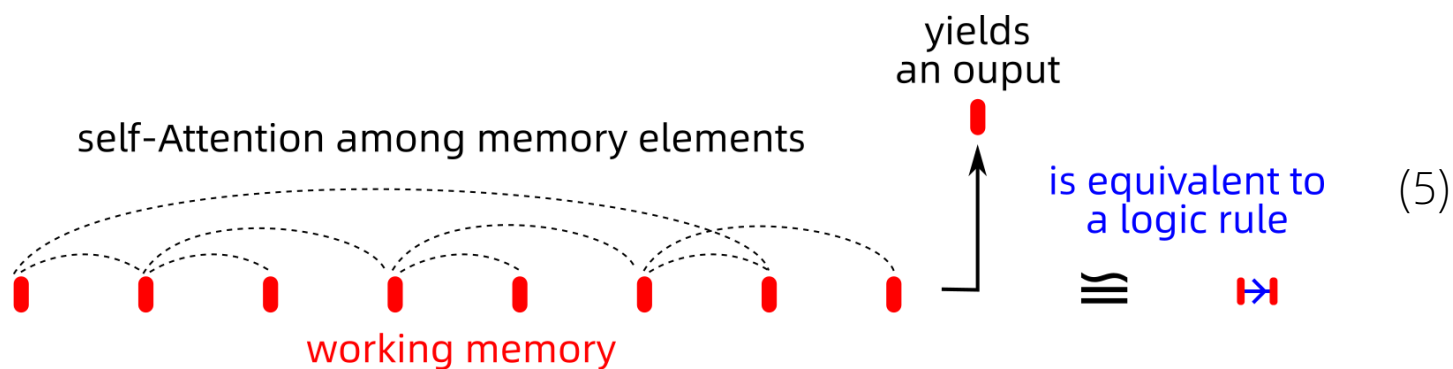
In other words, there is some sort of (distorted) logic rules inside the Transformer. The natural question then is: Can we find additional structures of logic / logic systems? That is, what kind of algebraic structure constrains (4)? For this question, we can take insights from categorical logic and classical logic-based AI systems.

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

- The state is **granular**, made of elements of a certain set, and such elements can be permuted, which corresponds to the equivariance of the Transformer. (Note: Transformers are equivariant, but equivariance does not necessarily require Transformers)

- **Deep structure**: Such as multi-layer networks, each layer is a composition of functions. Transformer also has a deep structure.

- Logic has granularity at the **proposition** level and at the **sub-propositional** level. The latter is the structure of **predicate** logic, for example: *loves(John, Mary)* can be regarded as the **product** of **algebraic elements**, as in: *John · loves · Mary*, the latter is also called a "word". The details are unimportant. Our focus is on how to apply this 2-level granular structure to deep neural networks.

- Each step of logic inference produces **one** new conclusion (or its probability distribution), and this new conclusion is added to the old state as an element of a proposition set, and the old state also needs to **forget** some old propositions, otherwise infinite memory is required. This is a bit different from the Transformer which always outputs the **same number** of tokens as its input. (We are also unsure whether Transformer tokens correspond to propositions or to predicates / atomic concepts).

- A logic rule usually depends on some premises while other premises are **irrelevant**; For example: *talk ∧ dark ∧ handsome ⇒ attractive to women*, where *rich* or *poor* is irrelevant. The Transformer's **softmax** also seems able to exclude some irrelevant tokens.

- (There are possibly other structural features.....)

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

(6)

atomic concepts   propositions   vector components
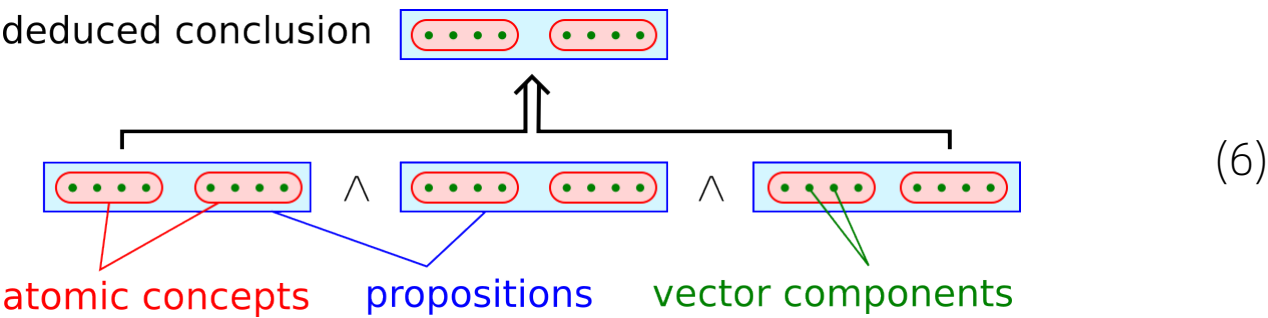
In contrast, the algebraic relation $p \wedge q = p \wedge q$ expresses only this structure:

(7)
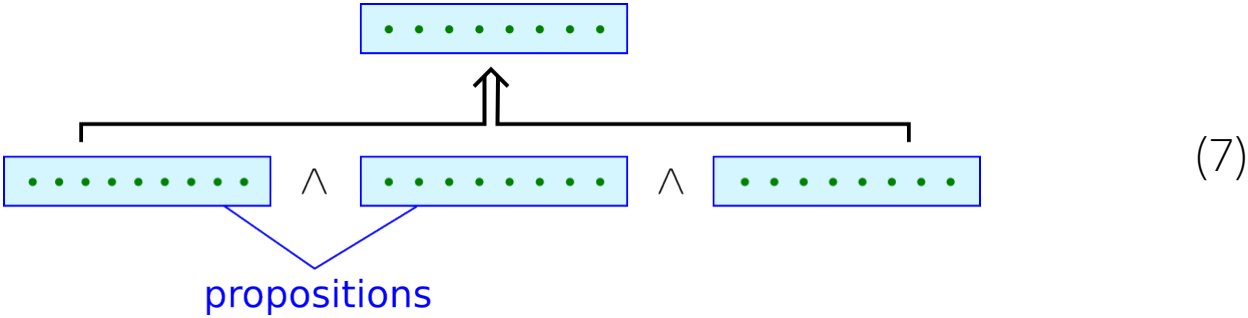
propositions

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

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

(8)

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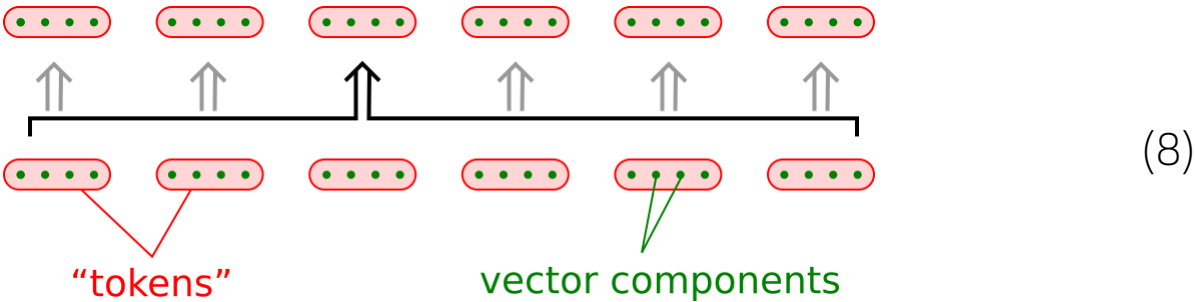
It does not represent propositions explicitly, but it uses a special "stop" token to signify the **end** of sentences, and there are other "tricks" such as **positional encoding**. It seems to be a rather *ad hoc* design, and we should be able to improve it.

Now we try to answer <u>the crucial question</u>: 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?

 (9)

This is like asking the difference between $0...9 \times 0...9$ and $00...99$ (they are isomorphic).

Similarly,

$$\{ \text{John, Mary} \} \times \{ \text{human, god, worm} \} \tag{10}$$

and the $2 \times 3 = 6$ propositions

$$\{ \text{John is human} , \text{ Mary is human} , ...\} \tag{11}$$

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

But since ∈ a non-commutative free group (the group with the least structure), it does not have a simple symmetry formula like $a \cdot b = b \cdot a$.

After some analysis I got the following condition for "propositions are made of conceptual atoms":

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

$$P_i = a_{i1} \cdot ... \cdot a_{iK} \tag{12}$$

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:*

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

and the transformation τ has to accord with ∀ or ∃ as adjunctions to a substitution functor.


In fact, τ only needs to be a continuous function to meet the above requirement. So the essence of the Atomic Condition lies in the two **equations** (12) and (13), which are actually very simple. The category-theoretic description of ∀ and ∃ as adjoint functors is complicated, and we will explain them in appendix A.

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

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

It is due to such "movements" that constitute the structure of "propositions as composed of concepts".

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


$$(15)$$
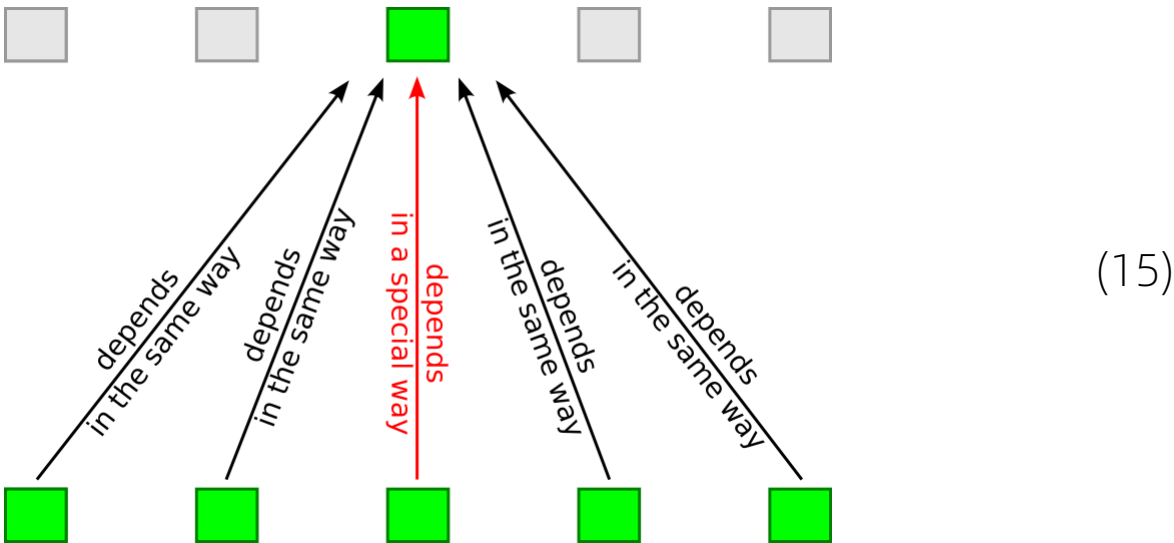
This vertical "axis" (red) is repeated per each input token. So, when the input elements are **swapped**, the output is also swapped. This is how Self-Attention achieves **equi-variance**.

We want to apply a similar method for logical structure:

$$(16)$$
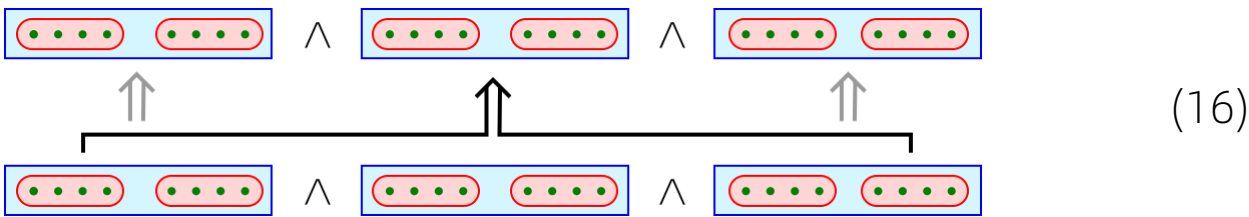
Here is a very important point[1]: The precursor of Self-Attention is the **content-addressable memory** from Graves *et al*'s "Neural Turing Machines". So we have good reason to regard Self-Attention as a form of **memory**.

We want to compare two approaches. One is the naïve classical rule-base structure, and the other uses Self-Attention instead of a rule base:

$$(17)$$

$$(5)$$
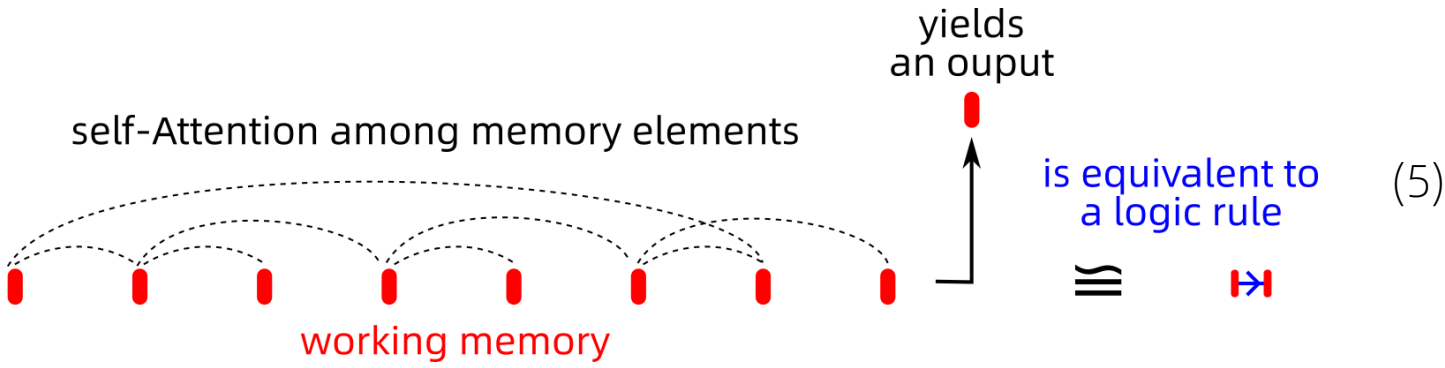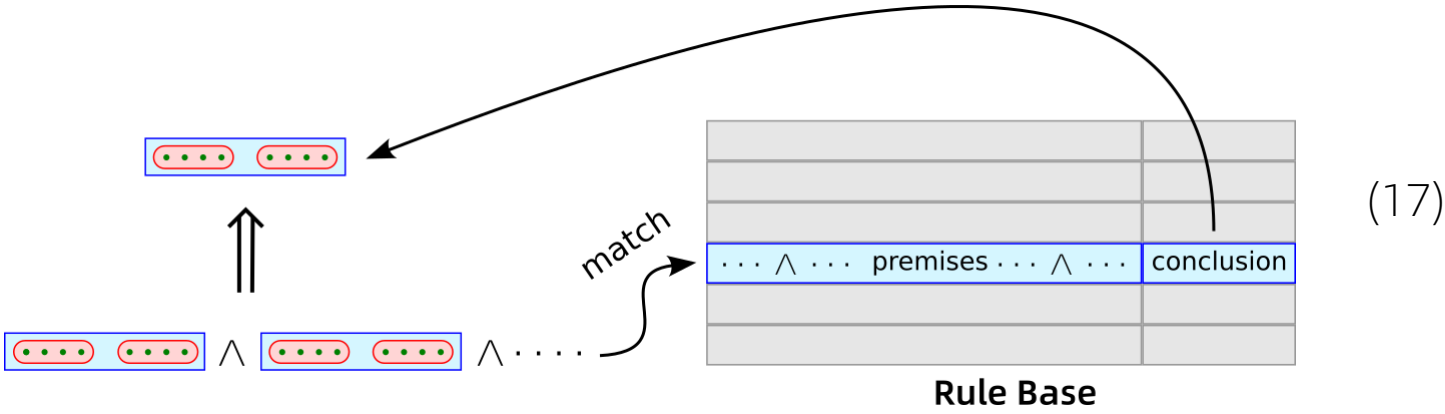
You should get the sense that the Transformer is a very "twisted" way of representing rules. The correspondence is so indirect that it is difficult for us to see what the rules look like on the Transformer side. However, I think the designers of the Transformer might have had at least an inkling of its similarities to rule-based systems. In particular, look at the following logic rule:

$$\forall X, Y, Z. \ \mathsf{grandfather}(X, Z) \leftarrow \mathsf{father}(X, Y) \wedge \mathsf{father}(Y, Z) \qquad (14)$$

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. But Self-Attention also ignores the symmetry of $A \wedge B$, so there may be room for improvement.

---

[1]Thanks "Ziyu" for telling me this important information.

Some key questions to be answered now:

- According to eg. Qian Liu's paper [1], Transformers often fail to perform certain logical and grammatical operations. Where is the problem? It doesn't seem that Transformers cannot learn the syntax at all, but that it cannot do it only with prompts. But do prompts actually have a deeper meaning, or are they just a hack? We didn't explicitly "tell" the Transformer what it should do, so is it a real shortcoming that it failed to solve those problems? I find it difficult to judge, and the research direction of prompts is shrouded in mystery.

- Now consider the naïve learning algorithm for the graph (17) that is the rule base. This algorithm is of course very slow, since two sets (working memory and rule head) similarity. Assuming that the size of the two sets is fixed at N, then $N \times N$ times of dot products are needed, and this is just a comparison of a rule. All rules need to be added with softmax. When the rule base When it is very large, this algorithm seems impractical.

- figure (17) may also have the problem of the old logic rule learning algorithm, that is "plateau problem". For example, write append in Prolog language function:

```
append(X,Y,Z) :-
        list(X), head(X,X1), tail(X,X2), append(X2,Y,W), cons(X1,W,Z).
```
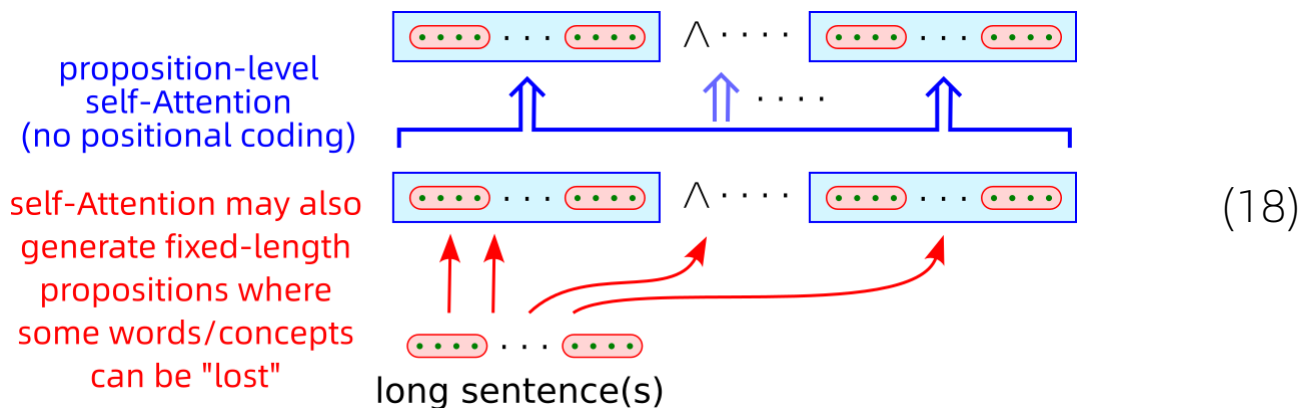
  This rule has 5 premises. When the rule is learned, the premise is added one by one, but the "score value" of the rule is always zero, until the last premise is added, the score suddenly rises to 100%. For machine learning, this situation is Terrible. And the Transformer twists the rules together. Will this approach help avoid being trapped in the local minima?

- May have an algorithm between Transformer and naïve rule base, it has a stronger logical structure than Transformer, but uses more similar self-Attention's matrix operation to speed up?

---

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

A more feasible architecture that I think of now:

proposition-level
self-Attention
(no positional coding)

self-Attention may also
generate fixed-length
propositions where
some words/concepts
can be "lost"

long sentence(s)

(18)

- In the blue structure, each proposition shall consist of fixed # of concept atoms.
- But the abstract self-Attention structure can also allow **variable length** propositions, as long as the **dot product** (similarity) between the propositions can be calculated, and the fixed-length **matrix** memory.
- red structure: how to generate propositions of varying number and length from sentences of indefinite length? This is the sequence to sequence-of-sequences problem discussed before. This can also be solved with "lossy" self-Attention.

In this appendix, we explain the theory of ∀ and ∃ in a simple way.

The following is a relation R, such as "Y loves X", where X, Y are both sets of "persons"; two identical copies. For example, the diagonal represents loving yourself (some people don't love themselves). Note that this graph is not diagonally symmetrical, otherwise there will be no "broken love". Projecting the **projection** of the relation R onto the X axis yields the $∀_Y$ and $∃_Y$ sets. The $∀_Y$ set represents those X that "everyone loves", and the $∃_Y$ set represents those X that "someone loves him":

$$(19)$$
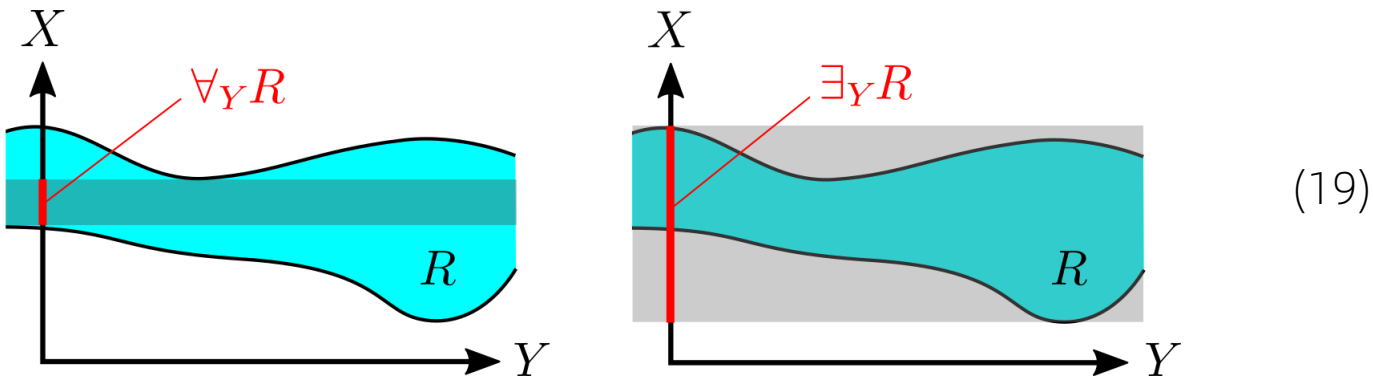
The category theory master Lawvere found that ∀ and ∃ are a so-called **weakening functor** "accompanying functor", the so-called weakening is to expand from a **discourse domain** with only X variables to There is a domain of two variables X, Y, and here Y is purely a **dummy** variable:

$$(X) \xrightarrow[\exists_Y]{\overset{\forall_Y}{\longleftarrow} \overset{\text{weakening}}{\longrightarrow} \overset{}{\longleftarrow}} (X, Y) \qquad (20)$$

For example, $\text{Love}(Y, X)$ is a logical expression with two variables, but $\forall Y. \text{Love}(Y, X)$ actually does not have the variable Y, Because it is ∀ **bound**.

The so-called **adjoint** (adjoint) means: There are two categories: left and right. You can move things from the left to the right, and do "**comparison**" in the category on the right, and this comparison can also move things to the left, and the two comparisons are **equivalent**. Here "comparison" means morphism within a category, for example in the category **Set** comparison is set inclusion.

Adjoint functors are not unique, so weakening has two adjoints ∀ and ∃ respectively.

Lawvere's work made the ∀ and ∃ quantifiers more general: the "simple" weakening functor is based on the Cartesian product $X \times Y$, but Lawvere expanded it to arbitrary **substitution** functors. (I don't know of examples of this, so I'm not sure what advantages this could bring to our application.)

In categorical logic there are **Beck-Chevalley** conditions and **Frobenius** conditions, perhaps the symmetry we need? But after a closer look, I found that the problem still cannot be solved... For completeness, I will describe it, and you can skip it if you are not interested.

Consider first the easier-to-understand **Frobenius** condition. Logically, it is equivalent to saying:

$$\exists x.[\phi \wedge \psi(x)] \equiv \phi \wedge \exists x.\psi(x). \tag{21}$$

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

$$\forall x.[\phi \vee \psi(x)] \equiv \phi \vee \forall x.\psi(x). \tag{22}$$

But the problem is, the left and right sides of the formula (22), the corresponding neural network (6) is the same (no difference). In other words, this difference may be too subtle, and it does not affect the neural network we actually implement.

As 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 the fibration structure:

(23)
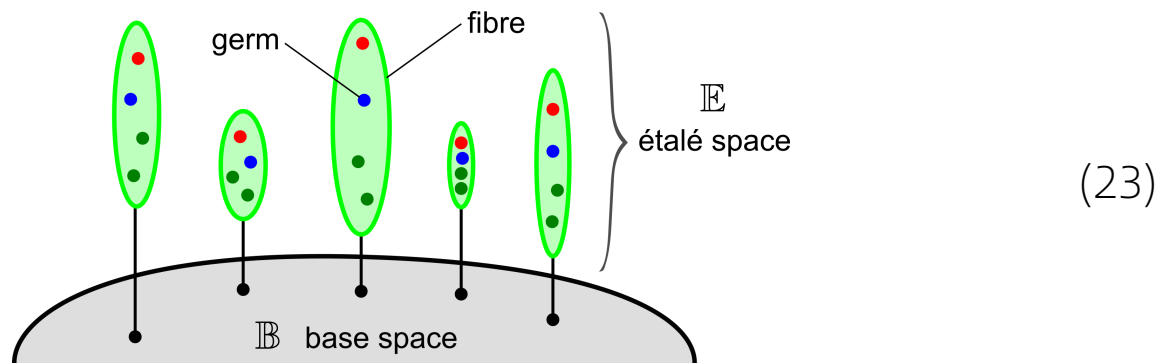
This whole structure is called **bundle**, and **sheaf** is bundle plus some special topology.

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

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

(24)

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

The **Beck-Chevalley** condition says that the following image commute:

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

(25)

where π is the projection representing the quantifier $\forall$ or $\exists$, which are the accompanying maps of the weakening map π^*.

The Beck-Chevalley condition is not entirely hollow; it may not hold. There is a counter-example from Pitts: Consider $X \times Y$, where $X = Y = \mathbb{N} \cup \{\infty\}$ is the natural number plus ∞ as top element; but Y uses discrete order, that is, all orders are = 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 the graph (25) not to commute. (I do not understand the details of Scott closure)

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

$$\boxed{\text{output proposition } O_i \text{ is composed of atoms } b_i} \quad O_i = [b_1...b_K] \quad (26)$$

$$\boxed{\text{input proposition } P_i \text{ is composed of atoms } a_i} \quad P_i = [a_1...a_K] \quad (27)$$

$$\boxed{\text{Self-Attention}} \quad O_i = \alpha(P_i ; P_1...\hat{P}_i...P_N) \quad (28)$$

$P_1...\hat{P}_i...P_N$ means $P_1...P_N$ except P_i.
$\alpha(P_i ;...)$ 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)?

$$\arg\min_i \sum_i \min_j \langle P_i, Q_j \rangle \quad (29)$$

- 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.

(30)
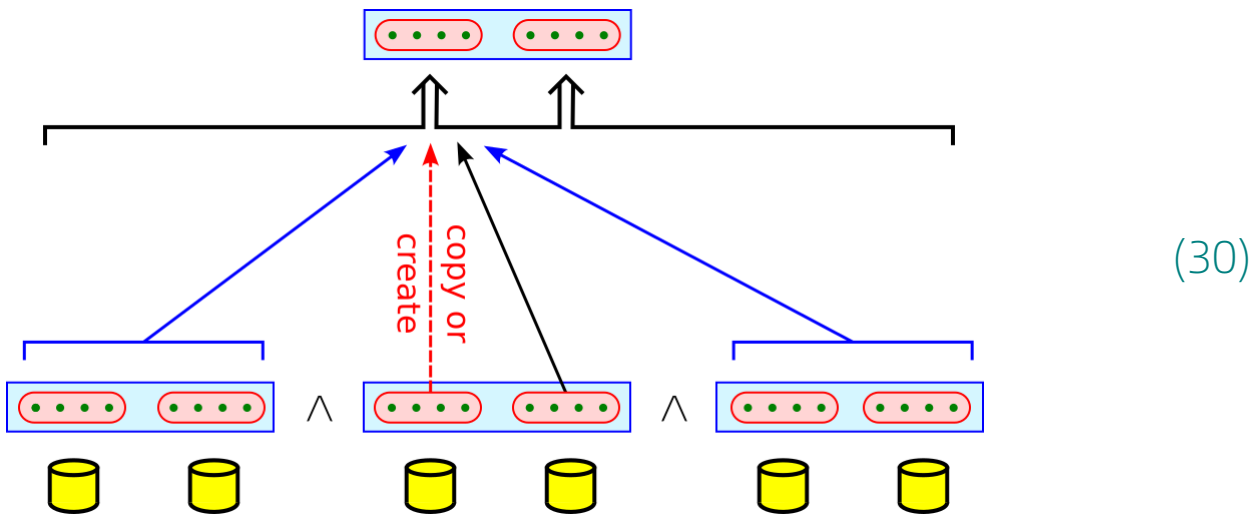
Copy:
- Copy requires no special mechanism, the output is the input. But the question is how to combine with "create" operation.
- can of course use the familiar softmax: α copy $+ \beta$ create, where α, β are outputs of softmax.
- Another idea is content-addressable. Use table-lookup to find executable rules. But there is a problem with variable matching.

Create:
- What kind of function is needed?