

# AGI standard model

## — trying to establish a consensus

YKY

May 16, 2022

## 0 Introduction

The “standard model” is a way of thinking, that may help us better understand the general theory of AGI systems.

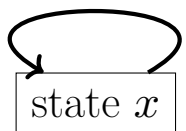
The essence of the standard model is just to identify a **Working Memory** or “state” of the AGI system.

One benefit of our theory is that it relates Transformers / BERT / GPT to AGI systems. These language models are phenomenally intelligent, yet many people criticize them as not “truly” intelligent. The standard model suggests that they are indeed linked to AGI. There are other benefits.

## 1 Reinforcement learning

This is the simplest form of a **dynamical system**:

transition function  $F$



= “working memory”

(1)

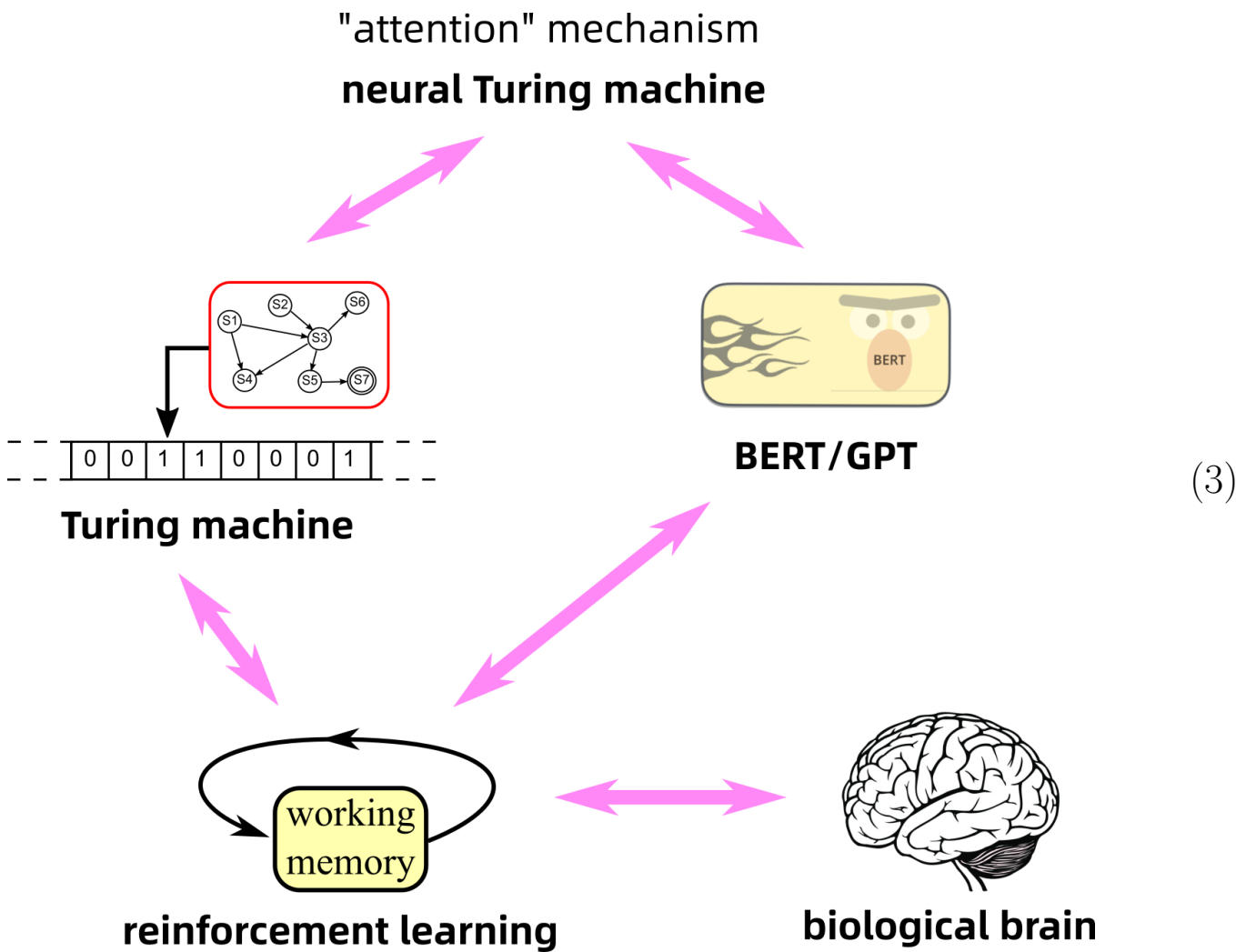
When we add a “control” or “action” variable  $a$  to it, it becomes the most basic **control system**:

$$\begin{array}{c}
 F(x, a) \\
 \begin{array}{c} \curvearrowright \\ \boxed{x} \end{array}
 \end{array}
 \tag{2}$$

which is the setting for Dynamic Programming or **Reinforcement Learning**. The optimal solution for such systems is governed by the **Hamilton-Jacobi-Bellman equation**.

I call this the “standard model” because of the extreme simplicity of this setup, and that I don’t know of other alternative models that deviate much from it.

The following diagram shows how the standard model relates to several other important areas, so we can reap profits from their interactions:



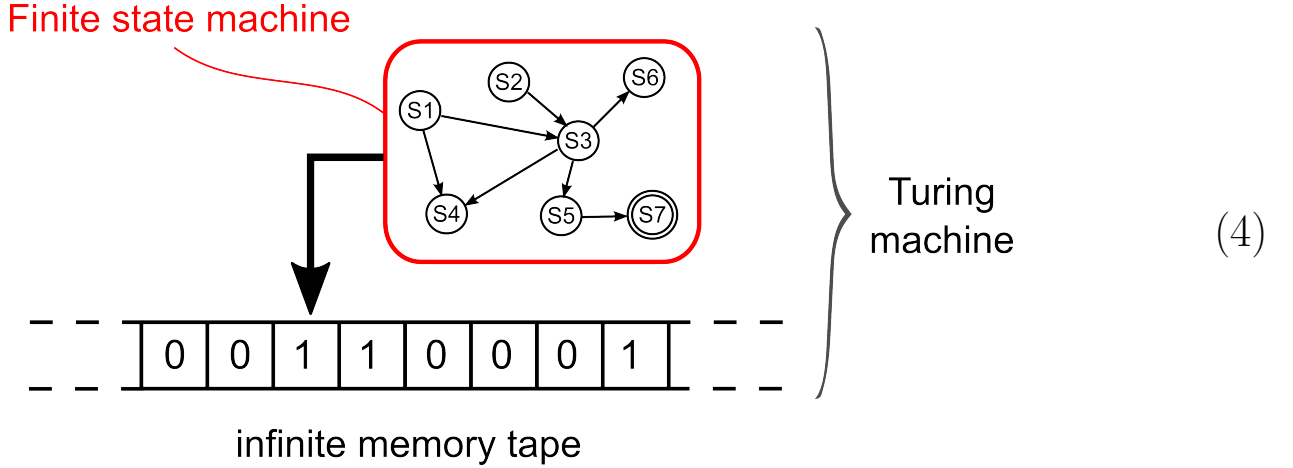
Recently, Yann LeCun’s **Energy-Based Models** offers a way to circumvent the

problem of learning probability distributions over actions, when the action space is hugely high-dimensional.

## 2 Neural Turing Machines and Transformers

The **attention mechanism** was first proposed in the “**Neural Turing Machine**” paper by Graves *et al* [2014].

Recall that a Turing machine is a **Finite State Machine** augmented with a **Memory Tape**:



In Neural Turing Machines, Graves *et al* proposed the attention mechanism for an RNN “Controller” (playing the role of the Finite State Machine) to read and write from a **Memory Matrix** (the tape), using a content-based addressing method.

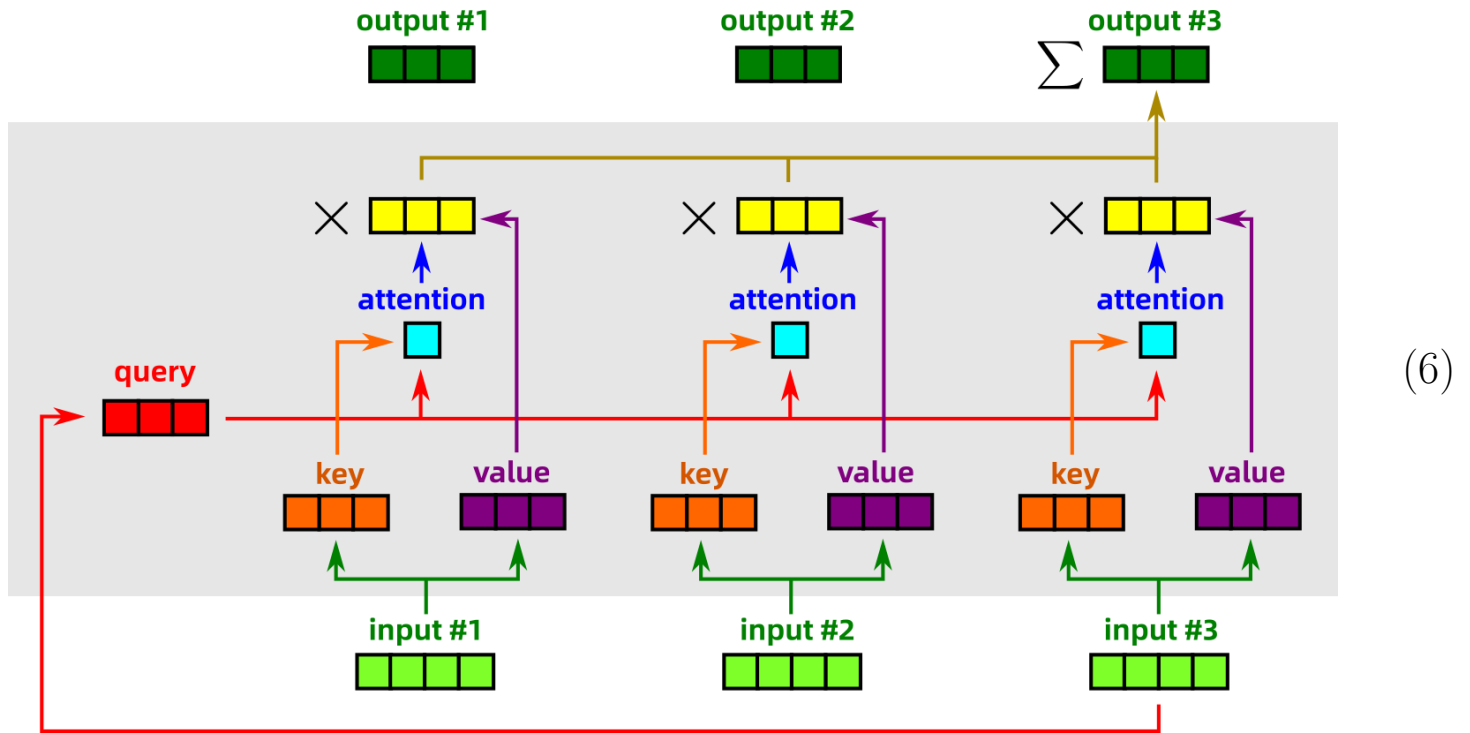
The Memory Matrix  $M$  consists of  $N$  items, each of constant size. *The discreteness of the address would introduce discontinuities in gradients of the output*, hence we need an **Attention Vector** to focus on a specific location in the memory matrix  $M$ .

The Attention Vector  $\vec{a}$  is calculated via the following formula, familiar to students of the Transformer:

$$\vec{a} = \text{soft max}_i \{ \mathcal{D}(K, M_i) \} \quad (5)$$

where  $D()$  is a similarity measure between the key  $K$  and memory item  $M_i$ . The key  $K$  is emitted by the Controller as the value that it is looking for.

This then evolved into the **Self Attention** mechanism used in all Transformers. Now let us refresh with this diagram illustrating Self-Attention (ripped from a blog article on the web):



The research done by Olah *et al*, in their 2021 paper *A Mathematical Framework for Transformer Circuits*, is very helpful towards understanding Transformers and Self-Attention. The paper is technically quite challenging, but thanks to the guidance of professor Xiao Da from Beijing I was able to understand the main ideas. Here I offer some pointers to help others understand the paper, without explaining it in full details.

The Self Attention  $A$  is calculated by matrix operations followed by **softmax**, and thus is non-linear. However, when  $A$  is held constant, the Transformer becomes a linear operation that can be expressed as **tensor products**, for example:

$$T = \underbrace{\text{Id} \otimes W_U}_{\text{The token unembedding maps residual stream vectors to logits.}} \cdot \underbrace{\left( \text{Id} + \sum_{h \in H_1} A^h \otimes W_{OV}^h \right)}_{\text{The attention layer has multiple heads. The result of each is added into the residual stream.}} \cdot \underbrace{\text{Id} \otimes W_E}_{\text{The token embedding maps tokens to residual stream vectors.}} \quad (7)$$

For example, to understand the equation for a Transformer “Head”:

$$h(x) = (\text{Id} \otimes W_O) \cdot (A \otimes \text{Id}) \cdot (\text{Id} \otimes W_V) \cdot x \quad (8)$$

it is helpful to know that  $x$  is a **matrix** with dimensions  $[n_{\text{context}}, d_{\text{model}}]$ , ie, one dimension being the number of tokens and the other dimension representing the token’s vector. When a tensor such as  $U \otimes V$  acts on  $x$ ,  $U$  acts on the “position” dimension of the matrix while  $V$  acts on the “vector” dimension. So the above equation can be collapsed to:

$$h(x) = (A \otimes W_O W_V) \cdot x. \quad (9)$$

Thus tensor products are very convenient for expressing such operations.

## Tensor product

Recall that a **linear** map is defined between two vector spaces:

$$F : U \rightarrow V. \quad (10)$$

Similarly one can define a **bi-linear** map:

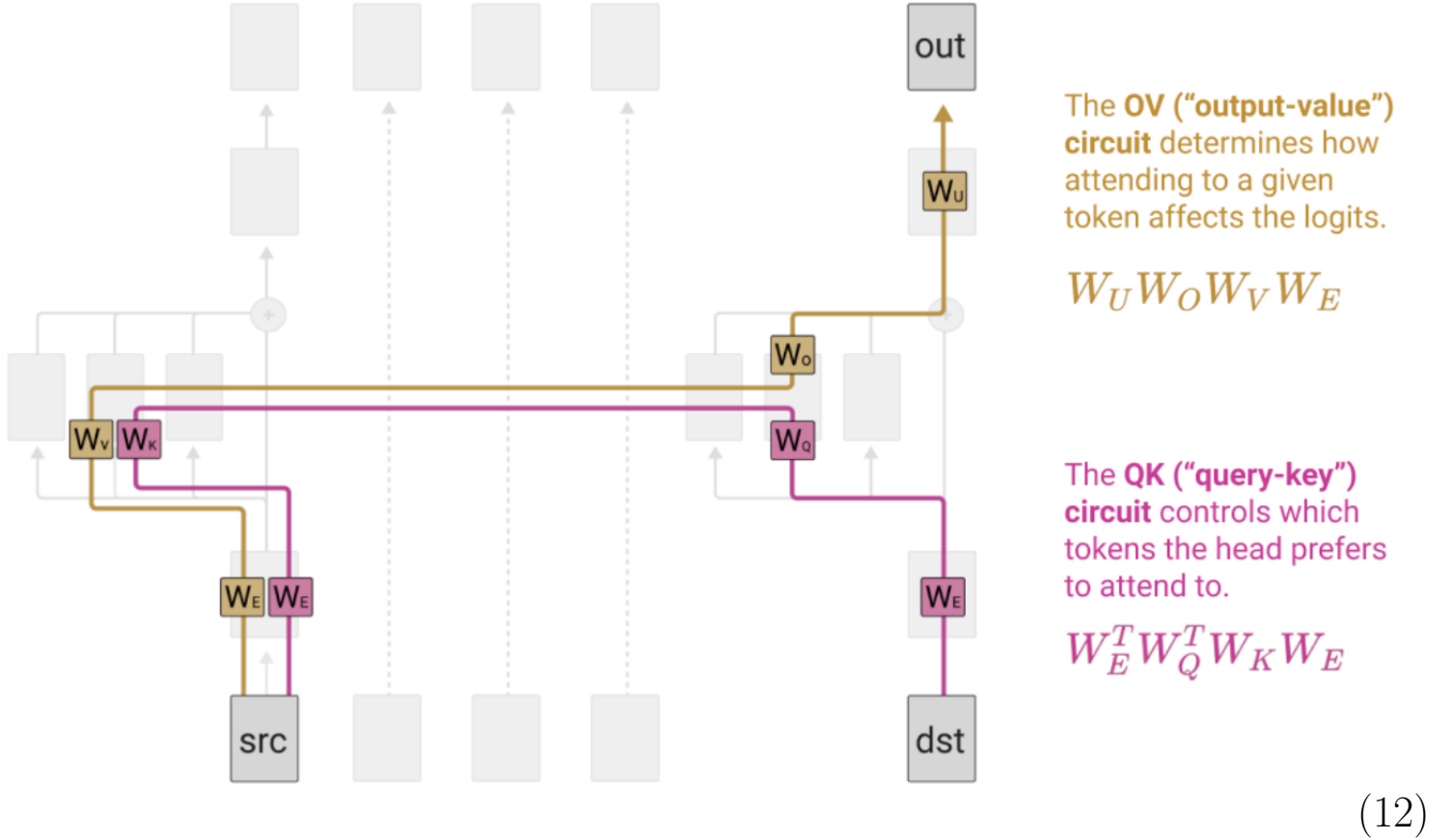
$$\Phi : V \times W \rightarrow U \quad (11)$$

that is linear in both its first and second arguments. The tensor product  $\otimes$  is the **universal** multi-linear map.

The tensor product generalizes to the mathematically important **monoidal categories**. The latter captures the associativity (and optionally commutativity, for which we have symmetric monoidal categories), but the linearity is abstracted away.

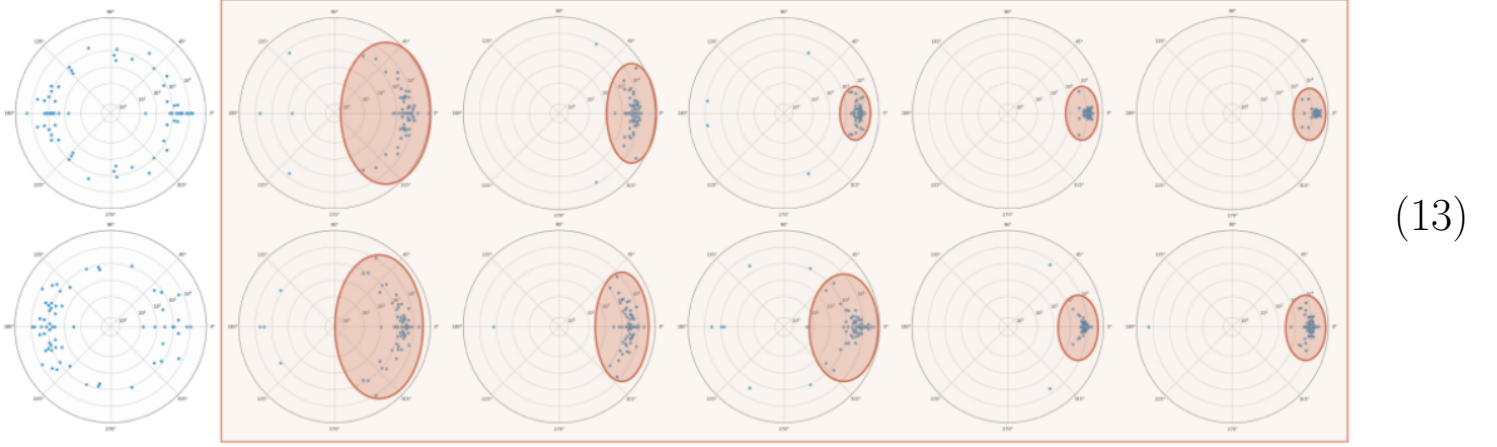
Monoidal categories can be depicted by **String Diagrams**. There are also **Neural String Diagrams** that describe the structure of neural networks. For example, the Transformer layer is **equivariant** in its inputs.

In the Transformer, we can see **Input Values** flowing through some “circuits” defined by matrices to yield the **Output Values**:



Some Attention Heads merely perform “**copying**” of a vector from a previous layer to the next. If we spell this out as a linear equation,  $W\vec{x} = \lambda\vec{x}$ , where the transformation  $W$  merely results in a **scaling** of the vector  $\vec{x}$ . This is just an eigenvector equation. So if we plot the eigenvalues of the transformation matrices on the complex plane, the “copy” maps would have eigenvalues close to positive real numbers. The results shows that such “copy” maps are ubiquitous in Attention

Heads (here 10 out of 12 maps are positive):



Such Attention Heads may be interpreted as **universally-quantified** ( $\forall$ ) logic formulas. For example when we say “all men are mortal”:

$$\forall x. Human(x) \Rightarrow Mortal(x) \quad (14)$$

any object instantiated as  $x$  (eg. “Socrates”) would have to be **copied** from the LHS to the RHS.

### 3 Relation to the biological brain

There are two distinct aspects in the brain:

- **Short-term** or Working Memory is the **electric activation** of neuronal populations.
- **Long-term** memory is stored as **synaptic strengths**, established by synaptic formation and strengthening. The transfer from STM to LTM is called **memory consolidation**.

One theory has it that the prefrontal cortex maintains a number of “thoughts” with sub-populations or, perhaps, with **micro-columns**. These activated sub-populations are in competition with each other, through **lateral inhibition**. The thought(s) that win are the thoughts we retain – they “make sense”.

### 4 Abductive reasoning

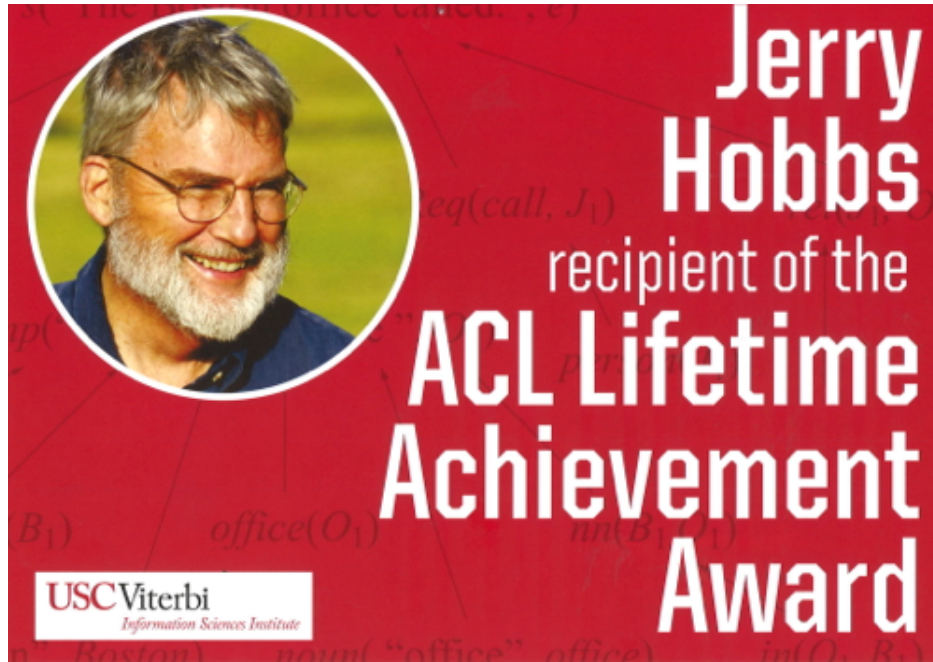
Abduction has been relatively neglected in AGI research, which had focused on forward inference. Recently there is a call to study this important aspect.



In logic, abduction means finding the **explanation** for some known facts. An explanation  $E$  is simply some propositions that imply the known fact  $F$ , ie,  $E \Rightarrow F$ .

For example, why do we think a certain actress, say Marilyn Monroe, is sexy? That's because we recognize she has some features (visual or otherwise, no need to enumerate them explicitly) that we consider sexy. So,  $E_1 \wedge E_2 \wedge \dots \Rightarrow \text{Sexy}$ . Those conditions **imply** she is sexy, and they are the **explanation** for her sexiness.

Why is abduction important? For example, when a waitress says “The Ham Sandwich left a big tip”, *Ham Sandwich* here refers to the customer who ordered it (an example of metonymy). The AI knows the plain facts such as that someone ordered a ham sandwich, and then it abduces that the most likely **interpretation** of the phrase “Ham Sandwich” is as the person associated with it. This is the basis of **Abductive Interpretation of Natural Language** proposed by Jerry Hobbs:



(15)

So abductive reasoning is basically just **bidirectional** inference.

When a system has both forward and backward connections, it forms a loop and its dynamics is likely to produce “**resonance**”. This harks back to the ART (**Adaptive Resonance Theory**) proposed by Grossberg and Carpenter beginning in the 1980s.

Such resonance behavior can be viewed as the system seeking to minimize an energy, ie, trying to find the “best explanation” to a set of facts.



This is also corroborated by neuroscientific evidence: areas in the cerebral cortex are replete with both forward- as well as **back-projections**.

## 5 Dealing with assumptions

Example of an assumption: “If I play move  $x$  now, I will checkmate in 3 moves”.

### Tic Tac Toe example

Assume the current board is  $\begin{array}{|c|c|c|} \hline & & \text{O} \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & & \\ \hline \end{array}$  and it's **X**'s turn to play.

**X** can do the “double fork” by playing  $\begin{array}{|c|c|c|} \hline & & \text{O} \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & & \text{X} \\ \hline \end{array}$ .

But how can an AGI know (or prove) this?

If the current board is  $\begin{array}{|c|c|c|} \hline & & \text{O} \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & & \text{X} \\ \hline \end{array}$  then a double fork exists. We need a predicate to detect double forks.

We need to reason that even if **O** plays the “blocking” move  $\begin{array}{|c|c|c|} \hline \text{O} & & \text{O} \\ \hline \text{O} & \text{X} & \\ \hline \text{X} & & \text{X} \\ \hline \end{array}$ , **X** can still win.

We can easily express the conditions for **X**-can-win, but the difficult part is to make the assumption in **red**, in other words:

$$\text{red-move} \rightarrow \text{X-can-win} \quad (16)$$

The difficulty lies in that the LHS is **not true** under the current facts. This conditional statement must be proven by, first, assuming the LHS, and then deriving the RHS. Then the assumption is **discharged** and the conditional statement is proven, via the ( $\rightarrow$  I) rule.

In the old days, in classical AI, assumptions are handled with **Truth Maintenance Systems** that keep track of inference traces symbolically. These systems can get quite complicated with the need to track multiple assumptions. For example, when we plan a bank robbery, we need to consider many possible forking scenarios.

Suppose  $M, N$  are proofs of  $M : \phi \rightarrow \psi$  and  $N : \phi$ . Then the proof of  $\psi$  would be the application of  $M$  to  $N$ , denoted as  $@(M, N)$  or simply  $MN$ .

The assumption rule (Ax):

$$\Gamma, \phi \vdash \phi \quad (\text{Ax}) \quad (17)$$

for example can be written as:

$$x : \phi, y : \psi \vdash x : \phi \quad (\text{Ax}) \quad (18)$$

which is why we say that an assumption is a  **$\lambda$ -variable**.

**Discharging** an assumption (ie, using the  $\rightarrow$ I rule) results in a  **$\lambda$ -term**.

An AGI needs the ability to place an implication  $\phi \rightarrow \psi$  into working memory, and to prove it using the ( $\rightarrow$  I) rule, ie, by making an assumption.

**Algorithm:** Put the assumption  $A$  in Working Memory. Make inferences, marking all conclusions with the prefix  $A \rightarrow *$ . When enough conclusions are obtained, remove the assumption  $A$  from Working Memory.