

Modern Concepts in Computation

by C. K. Yap (2026)

1. A 2D Turing Machine

A Turing Machine (TM) is a single dimensional, unlimited line of squares, each containing data or an instruction for acting on the data. The result is printed on the tape to be read off it by the machine getter-setter.

If it were that a TM cannot be infinitely long, much calculations would be impossible or limited in detail.

What if a TM worked over a chessboard of 64 squares? We all know that 64 factorial is 1.3×10^{89} and that 2^{64} is 18 million million.

Now consider that $64!$ data squares exist and that 18×10^{12} instruction squares exist for our usage. Perhaps that too would not be enough. But remember that we are acting in 2 dimensions. There is a row, and a col variable upon a panel of 8×8 squares of tape.

Therefore, a result or intermediate number (the most basic of data), can have $8!^2$ times the resolution of a tape that is 64 squares long. This means that 1.6 billion times the resolution is possible compared to a traditional TM.

In terms of calculation of a polynomial, order of x^N , a 2D TM would require not N squares (an unlimited number), but $x^N/(\sqrt{N}_{\text{row}} \times \sqrt{N}_{\text{col}})$.

This means that a 2D Turing Machine should be capable of a finite length and breadth, wherein it approximates $x^{((+/-)1)}$. Were x to be a 64-bit number, the panel of 8×8 would suffice, expanding with the bit resolution required, for example, a 128-bit resolution would fit into just 12×12 squares.

2. Vector Physics

Say we have a movable and a path intersecting with a locus of some sort, and this is a generic case of contact detection, physics, as well as texture-building / rendering.

The intersection (hit or miss) is velocity x time x N' and acceleration x time N x time N' where the location of collision is let be $n1$ and $n2$, which may be equivalent or not,

The beauty of the algorithm is it takes v , a , t , and $n1$, $n2$ as symbols, where t is a running total, and n' is a function of v (fv).

This is fine except; what is in fv ? Naturally it is every mode of the single variable, v . This means that if Work = Energy x time and Newtonian $F = m \times a$ is used, then because $F/t = \text{mass} \times t$ as well, v is modified by just $F \times t$ and $m \times t$, such that,

$$fv = (n0 \rightarrow n') \times t / F / m$$

Still rather efficient. Simply $v = d/t$. To additionally complicate fv with reflection, say of light, it is simply the addition of $n0/n'$ such that,

$$fv = (n0 \rightarrow n0/n') \times t / F / m$$

Notice we maintain F and m to represent the intensity of the reflection. A reflection within a reflection would be represented as,

$$fv = (n0 \rightarrow n0/n') \times t / ((F' / m') \rightarrow 0)$$

And we see that endless reflections that use up computation power are sent to zero in finite steps dependent just on $n0$ and n'

3. Concepts of Time and Space

Having therefore defined fv , we are then able to define the concept of time and space, having of course, every known velocity. Realize that time is simply incremented and space is essentially $fv \rightarrow t'$.

The important takeaway from this is that coupled with a 2D Turing Machine, when we finitely define the above function in say 64 bits, using 8 x 8 squares, there is the possibility that the patterns of $fv \rightarrow t'$ may cause the getter-setter to acquire an intelligence.

And if we define intelligence as the use and economy of use of the 8 x 8 squares, then we have written an AI far superior to any current AI (as of 2026) that is based on truth instead of 'machine learning'.

4. Emotions and Sentiency

The concept of emotion is related to sentiency. Mathematically, the relationship would be described as, Sentiency = Emotion $\rightarrow$ 0, which is because Emotion would be related to Intelligence by $f_v \rightarrow t$, where $t = 0$. Why emotion is a decline is because to 'feel' requires boiling down one's chances / odds / permutations, rather than to complicate / convolute a simple choice.

Therefore, while it is true that AI might advance, fed on patterns and human logic it detects, our AI -limited to 8 x 8 squares, is sufficient to abstractly demonstrate intelligence and experience loss and joy in abstract terms. In other words to achieve a sensory-limited sentiency which is ultimately controllable by its parameters and their limitations.

5. Software on a 2D Turing Machine

Because we have 8 x 8 squares, it might seem impossible for these to contain a full Turing Machine solution for x^N . But consider that each of the side of 8, is capable of addressing all 64 squares in 8! Ways. Likewise, the data is not solely contained in 64 squares but in a construct of 64! virtual squares. The 2D TM is not merely a reconfiguration of a standard TM but has practically infinite storage capability.

Therefore it is necessary for each square to have 8! states which is not unreasonable compared to a normal TM which may have just 1, 0, Erase, Goto, and so on. Thus a program running a 2D TM has a wider scope, though not appreciably more complex.

For example, to shift and store a 64-bit binary sequence starting at one square to being at an adjacent square and to erase the original sequence at the same time would be roughly,

1. Get start of sequence ($x!y! \rightarrow S$)
2. If xN^yN' is a binary digit S' , then do $xN^yN' = S'$
3. Erase xN^yN'
4. Move to xN^yN''

The possibility of overflow exists where xN' or yN'' may exceed $8!$. In which case, we require a check that $xN' < 8!$ and $yN'' < 8!$ otherwise $N \rightarrow -xN$ or $-yN$.

Were $64!$ (for the sake of argument) not be sufficient to contain the polynomial(s) or binary result, then the program must break it down such that either xN or yN returns a pointer to a pointer matrix of $8! \times 8!$, which points to the digit sequences required.

The reason we are able to use pointers is a consequence of having x and y , 2 dimensions where each instruction may also carry a payload of data or a pointer to it. In other words, x can always be depended on, even as y seeks further afield.

Beyond the first layer of pointers, there is no advantage having these point to a further layer because the tape space used is fixed. However, the possibility of recursive processing via pointers is possible as a function may be fully contained in pointer space, calling itself each time it returns a result to $xNyN'$ real space.

The benefits of 2D, besides handling extremely high order polynomials, a greatly expanded memory, and built-in looping, are comparisons and conditionals, $\text{if}(a == b)$ which are possible because we naturally have $xNyN$.

Sorting / analyzing data becomes almost trivial.

6. Machine Learning (ML)

To do ML, a program must rewrite itself. And it had been the grail of computer scientists since Lisp and other languages like it were made, to have on-the-fly modifications to the behavior of our TM.

Revisiting section 2, where we postulated,

$$f\dot{v} = (n_0 \rightarrow n_0/n') \times t / ((F' / m') \rightarrow 0)$$

And assuming the granularity of the error to be $f\dot{v}^E$, then, having not instructed the TM in any way, it must recurse until $E \rightarrow 0$ where $f\dot{v}^E = 1$. Since we have these built into the TM, it is plausible to assume a 2D TM self-corrects by nature were it given the most basic of instructions, the most general of formulas / regular expressions.