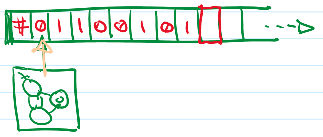
Turing machines, definitions (and why they're not important)

Wednesday, April 23, 2025 12:07 PM

Administrivia.

Turing machine.

Just DFA + Tape. (as memory).



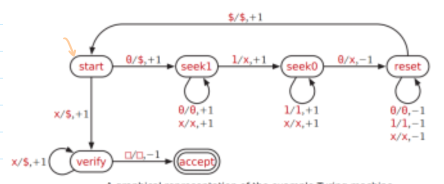
1. Tape is infinite!
2. "cursor" is movable.
   in two ways.

- Transition fcn $\delta : \underset{\text{symbol at curr. ptr}}{\Gamma} \times \underset{\text{curr state.}}{Q} \rightarrow \overset{\text{how curr ptr should be moved.}}{\{\leftarrow, \rightarrow\}} \times \underset{\text{next state}}{Q} \times \overset{M[\text{curr}] \leftarrow a}{\Gamma}$
- alphabet $\Gamma : \{0, 1\} \sqcup \{\Box\}$
- states Q
- start state s. accepting state Acc, rejecting state REJ
- memory tape. $M : \mathbb{Z} \rightarrow \Sigma$

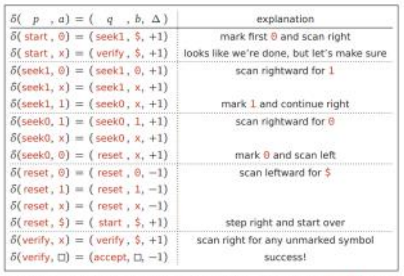
Example. $L = \{0^n 1^n 0^n : n \geq 0\}$

$\Gamma = \{0, 1, \$, x, \Box\}$
$\Sigma = \{0, 1\}$
$Q = \{\text{start}, \text{seek1}, \text{seek0}, \text{reset}, \text{verify}, \text{accept}, \text{reject}\}$

$\xrightarrow{a/b,\ \pm 1}$

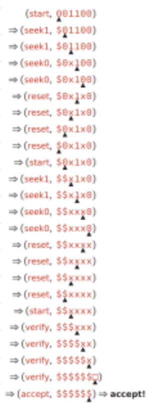


A graphical representation of the example Turing machine



δ(p , a) = (q , b, Δ)	explanation
δ(start , 0) = (seek1 , $, +1)	mark first 0 and scan right
δ(start , x) = (verify , $, +1)	looks like we're done, but let's make sure
δ(seek1, 0) = (seek1 , 0, +1)	scan rightward for 1
δ(seek1, x) = (seek1 , x, +1)	
δ(seek1, 1) = (seek0 , x, +1)	mark 1 and continue right
δ(seek0, 1) = (seek0 , 1, +1)	scan rightward for 0
δ(seek0, x) = (seek0 , x, +1)	
δ(seek0, 0) = (reset , x, +1)	mark 0 and scan left
δ(reset , 0) = (reset , 0, −1)	scan leftward for $
δ(reset , 1) = (reset , 1, −1)	
δ(reset , x) = (reset , x, −1)	
δ(reset , $) = (start , $, +1)	step right and start over
δ(verify, x) = (verify , $, +1)	scan right for any unmarked symbol
δ(verify, □) = (accept, □, −1)	success!

The transition function for a Turing machine that decides the language $\{0^n1^n0^n \mid n \geq 0\}$.



```
   (start, 001100)
⇒ (seek1, $01100)
⇒ (seek1, $01100)
⇒ (seek0, $0x100)
⇒ (seek0, $0x100)
⇒ (reset, $0x1x0)
⇒ (reset, $0x1x0)
⇒ (reset, $0x1x0)
⇒ (reset, $0x1x0)
⇒ (start, $0x1x0)
⇒ (seek1, $$x1x0)
⇒ (seek1, $$x1x0)
⇒ (seek0, $$xxx0)
⇒ (seek0, $$xxx0)
⇒ (reset, $$xxxx)
⇒ (reset, $$xxxx)
⇒ (reset, $$xxxx)
⇒ (reset, $$xxxx)
⇒ (start, $$xxxx)
⇒ (verify, $$$xxx)
⇒ (verify, $$$$xx)
⇒ (verify, $$$$$x)
⇒ (verify, $$$$$$□)
⇒ (accept, $$$$$$) ⇒ accept!
```

```
STRINGINL?(ω)
    input : string ω
    output : Is ω in L ?

    i ← 0.
    If ω[i] = 0 :
```

```
output : Is w in L ?
i ← 0.
If w[i] = 0 :
    w[i] ← $          « mark match 0.1.0 »
    find smallest j > i s.t. w[j] = 1
    w[j] ← X
    find smallest k > j s.t. w[k] = 0
    w[k] ← X
    find largest i s.t. w[i] = $   « reset »
    i++
If w[i] = $ or X :   « verify »
    while w[i] ≠ □ :
        REJECT if w[i] ≠ $ or X
        i++
ACCEPT
```

```
⇒ (verify, $$$$xx)
⇒ (verify, $$$$$x)
⇒ (verify, $$$$$$□)
⇒ (accept, $$$$$$) ⇒ accept!

   (start, 00100)
⇒ (seek1, $0100)
⇒ (seek1, $0100)
⇒ (seek0, $0x00)
⇒ (reset, $0xx0)
⇒ (reset, $0xx0)
⇒ (reset, $0xx0)
⇒ (start, $0xx0)
⇒ (seek1, $$xx0)
⇒ (seek1, $$xx0)
⇒ (seek1, $$xx0) ⇒ reject!
```

STRINGINL?(w)

```
input : string w
output : Is w in L ?
while w not all marked :
    marked the first subseq. 0.1.0
If all symbols in w are marked :
    ACCEPT
else :
    REJECT
```

STRINGINL?(w)

```
input : string w
output : Is w in L ?
ACCEPT if w is of the form $0^n 1^n 0^n$
```

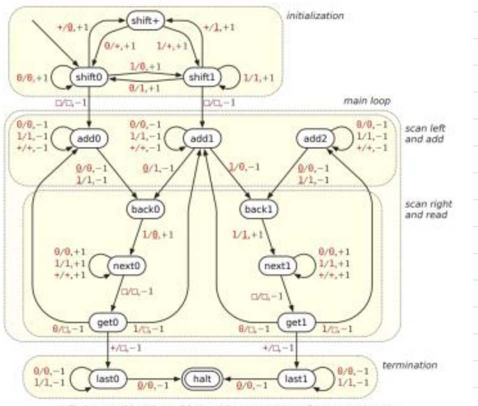
Pseudocode :

High-level description of an algorithm that can be realized by TM.

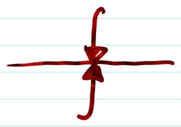
Example. Adding binary numbers.

Input. $\boxed{x} + \boxed{y}$

Output. $\boxed{z}$

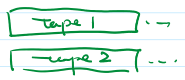


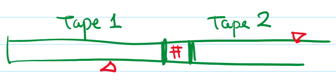
A Turing machine that adds two binary numbers of the same length.



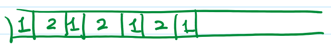
Now, why does it have to be a tape?

- multi-tape

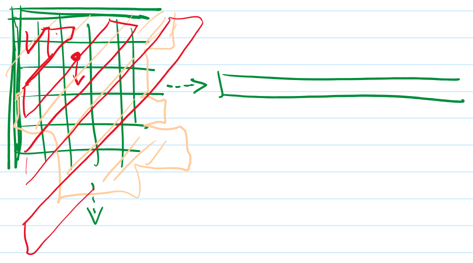




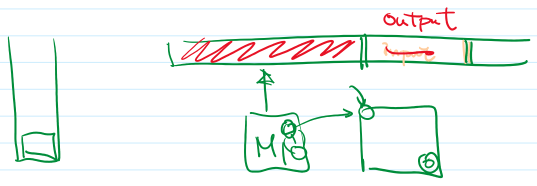
- multi-head



- 2d-tape (paper)
- recursion (stack + another paper)
- RAM ([address • data])



Example $L = \{0^n 1^n 0^n : n \geq 0\}$



Church-Turing Thesis. [1936]

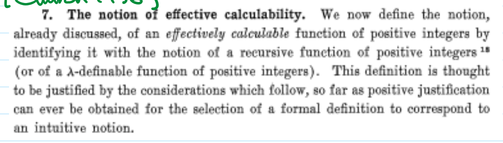
Any problems computable by machines can be computed by TM.

input → output
algorithms.

physical?
abstract?

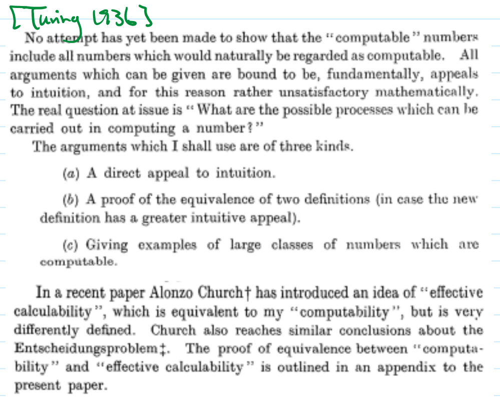
turn into?
emulated?

[Church 1936]



7. The notion of effective calculability. We now define the notion, already discussed, of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers[18] (or of a λ-definable function of positive integers). This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion.

[Turing 1936]



No attempt has yet been made to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is "What are the possible processes which can be carried out in computing a number?"

The arguments which I shall use are of three kinds.

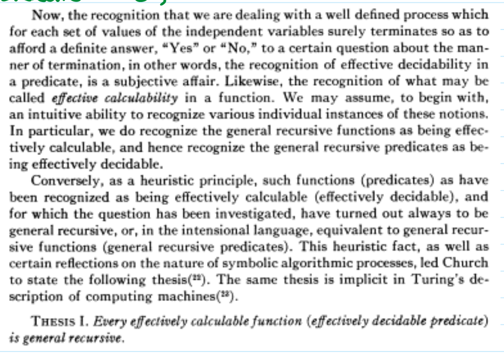
(*a*) A direct appeal to intuition.

(*b*) A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).

(*c*) Giving examples of large classes of numbers which are computable.

In a recent paper Alonzo Church† has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem‡. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.

[Kleene 1943]



Now, the recognition that we are dealing with a well defined process which for each set of values of the independent variables surely terminates so as to afford a definite answer, "Yes" or "No," to a certain question about the manner of termination, in other words, the recognition of effective decidability in a predicate, is a subjective affair. Likewise, the recognition of what may be called *effective calculability* in a function. We may assume, to begin with, an intuitive ability to recognize various individual instances of these notions. In particular, we do recognize the general recursive functions as being effectively calculable, and hence recognize the general recursive predicates as being effectively decidable.

Conversely, as a heuristic principle, such functions (predicates) as have been recognized as being effectively calculable (effectively decidable), and for which the question has been investigated, have turned out always to be general recursive, or, in the intensional language, equivalent to general recursive functions (general recursive predicates). This heuristic fact, as well as certain reflections on the nature of symbolic algorithmic processes, led Church to state the following thesis(22). The same thesis is implicit in Turing's description of computing machines(23).

THESIS I. *Every effectively calculable function (effectively decidable predicate) is general recursive.*

Corollary. We can compute any fcn / solve any problems computable by TM if we can simulate TM.