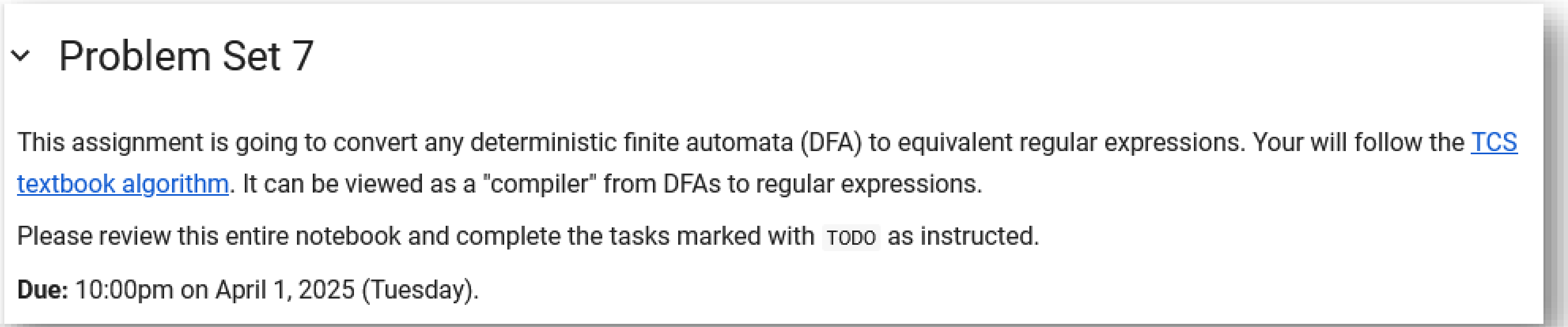
PS7 due next Tue (April 1)
Coming soon: PS8, PRR9

Photo: Movie Poster

Class 17: Turing Machines

University of Virginia
cs3120: DMT2
Wei-Kai Lin


Problem Set 7

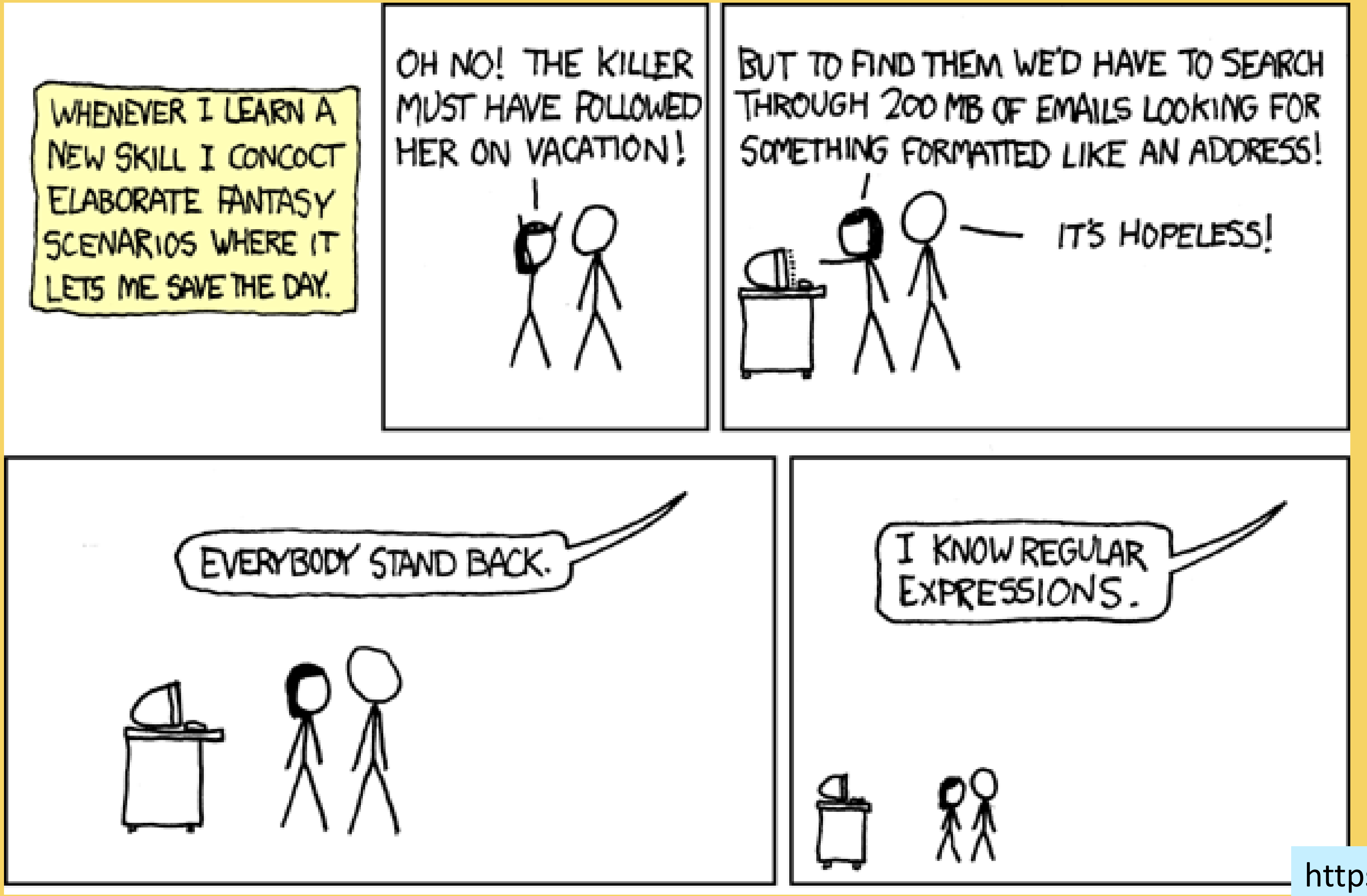
This assignment is going to convert any deterministic finite automata (DFA) to equivalent regular expressions. Your will follow the TCS textbook algorithm. It can be viewed as a "compiler" from DFAs to regular expressions.

Please review this entire notebook and complete the tasks marked with `TODO` as instructed.

Due: 10:00pm on April 1, 2025 (Tuesday).

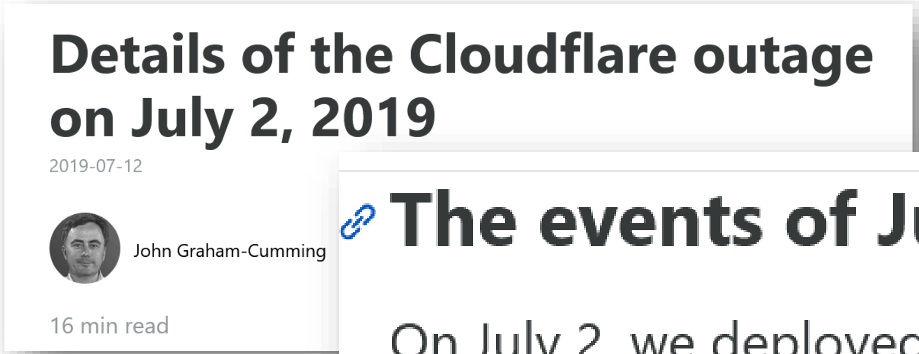
a lot of hands on example for learning

How can I learn to appreciate the theoretical realm of computing if I have always worked with concrete examples and systems?


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.

Regular Expressions in Practice

https://xkcd.com/208/



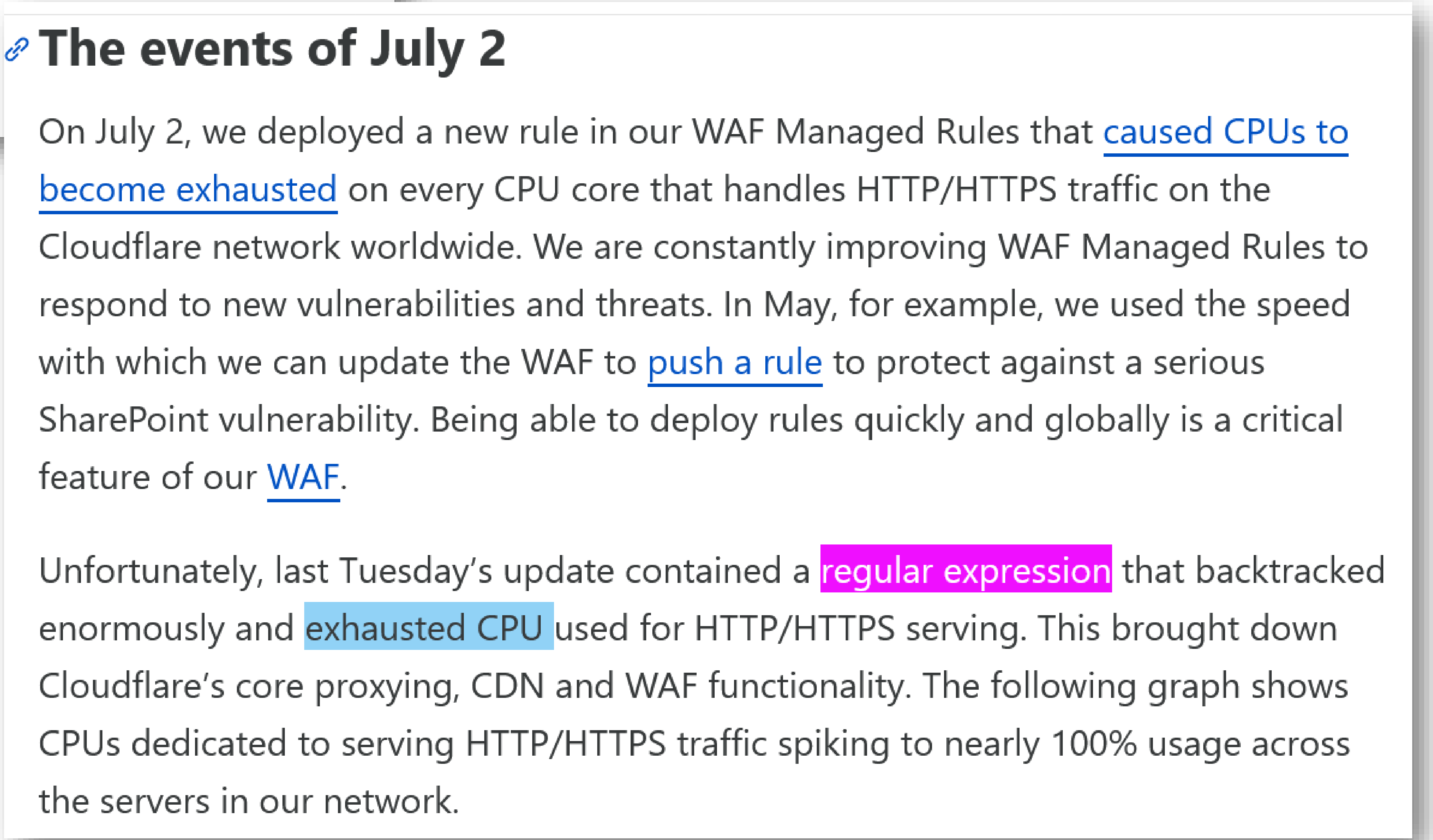
Details of the Cloudflare outage on July 2, 2019

2019-07-12

John Graham-Cumming

16 min read

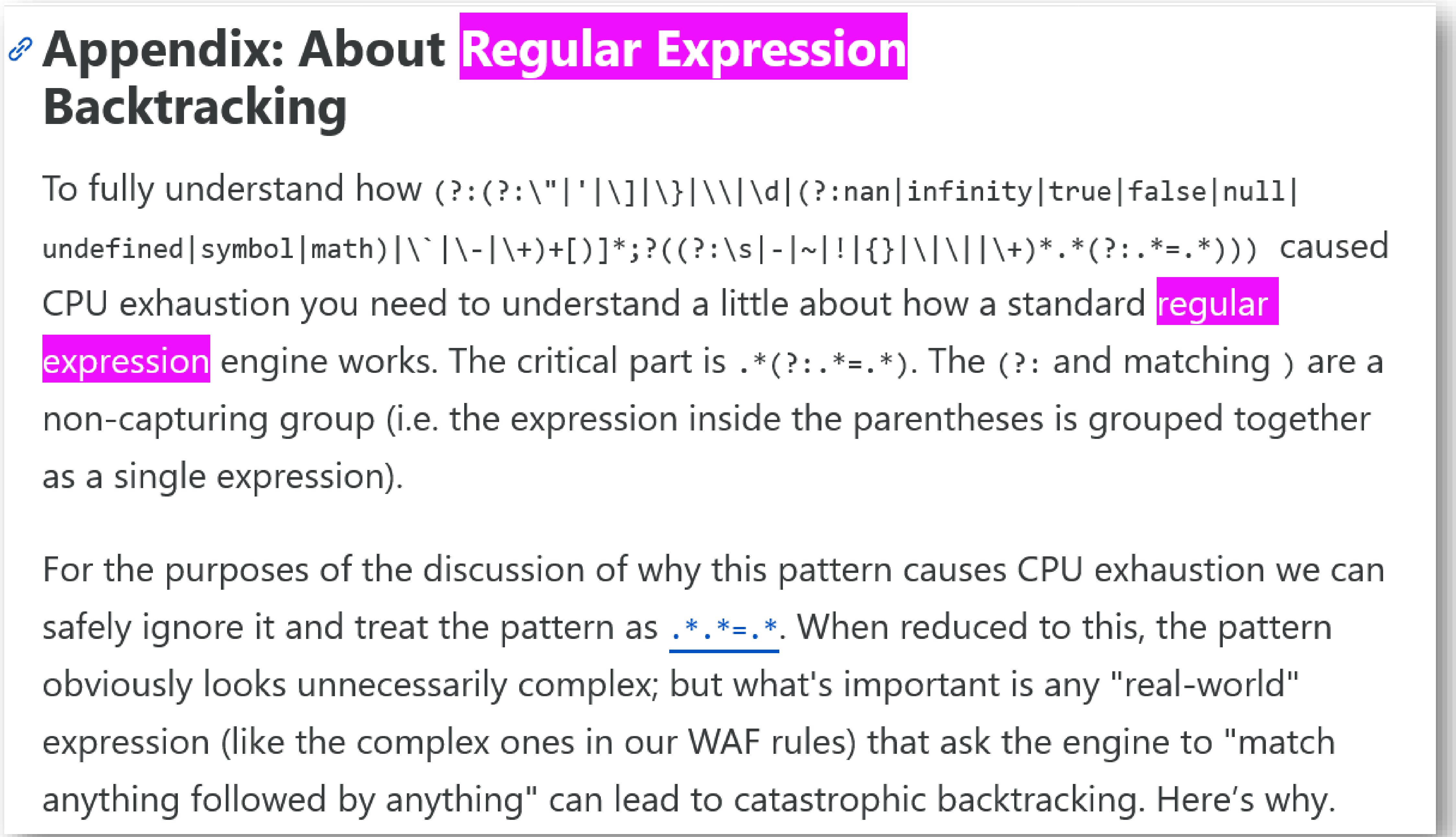
https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/



The events of July 2

On July 2, we deployed a new rule in our WAF Managed Rules that caused CPUs to become exhausted on every CPU core that handles HTTP/HTTPS traffic on the Cloudflare network worldwide. We are constantly improving WAF Managed Rules to respond to new vulnerabilities and threats. In May, for example, we used the speed with which we can update the WAF to push a rule to protect against a serious SharePoint vulnerability. Being able to deploy rules quickly and globally is a critical feature of our WAF.

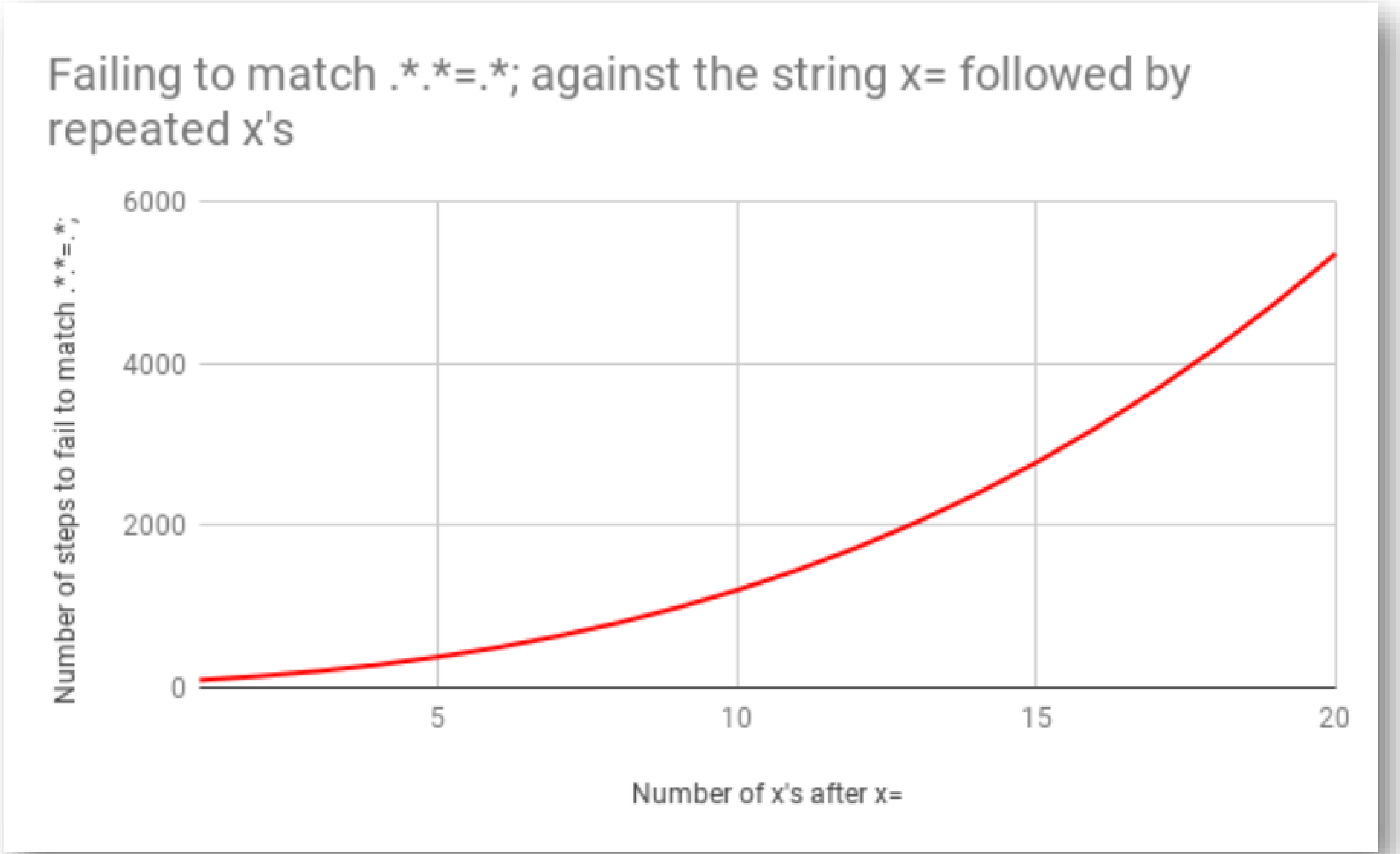
Unfortunately, last Tuesday's update contained a regular expression that backtracked enormously and exhausted CPU used for HTTP/HTTPS serving. This brought down Cloudflare's core proxying, CDN and WAF functionality. The following graph shows CPUs dedicated to serving HTTP/HTTPS traffic spiking to nearly 100% usage across the servers in our network.



Appendix: About Regular Expression Backtracking

To fully understand how ``(?:(?:\"|'|\]|\}|\\|\d|(?:nan|infinity|true|false|null|undefined|symbol|math)|\`|\-|\+)+[)]*;?((?:\s|-|~|!|{}|\|\||\+)*.*(?:.*=.*)))`` caused CPU exhaustion you need to understand a little about how a standard regular expression engine works. The critical part is `.*(?:.*=.*)`. The `(?:` and matching `)` are a non-capturing group (i.e. the expression inside the parentheses is grouped together as a single expression).

For the purposes of the discussion of why this pattern causes CPU exhaustion we can safely ignore it and treat the pattern as `.*.*=.*`. When reduced to this, the pattern obviously looks unnecessarily complex; but what's important is any "real-world" expression (like the complex ones in our WAF rules) that ask the engine to "match anything followed by anything" can lead to catastrophic backtracking. Here’s why.


Failing to match .*.*=.*; against the string x= followed by repeated x's
6000
4000
2000
0
5
10
15
20
Number of steps to fail to match .*.*=.*;
Number of x's after x=

RE: $\overset{e_1}{\underset{\text{any char}}{.}*}\quad \overset{e_2}{.*}\quad \overset{e_3}{\underset{\text{"="}}{=}}\quad \overset{e_4}{.*}$

String: $\boxed{x{=}xx}$?

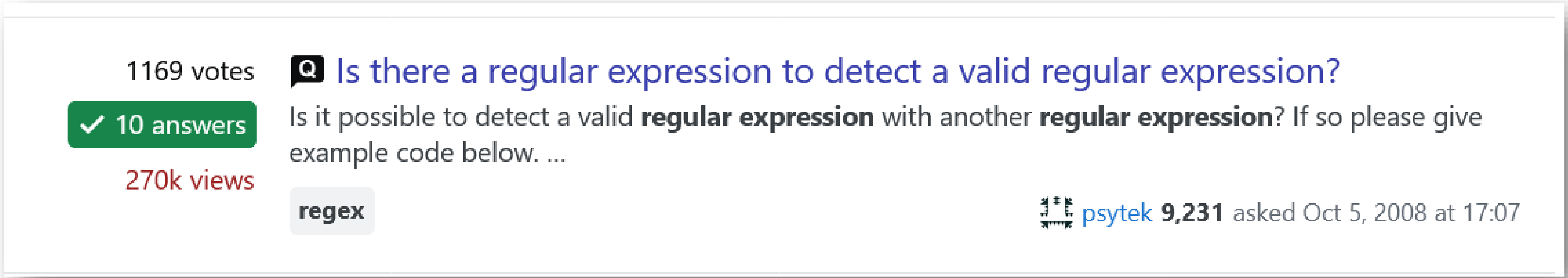
$\underbrace{x}_{e_1}\underbrace{=}_{e_3}\underbrace{xxxxxx}_{e_4}$

X=XXXXXXXXXXXXXXXXXXXXXXXXXX

e_1 ✗

e_1 ✗

Write regular expression to match (and only match) valid regular expressions



1169 votes

✓ 10 answers

270k views

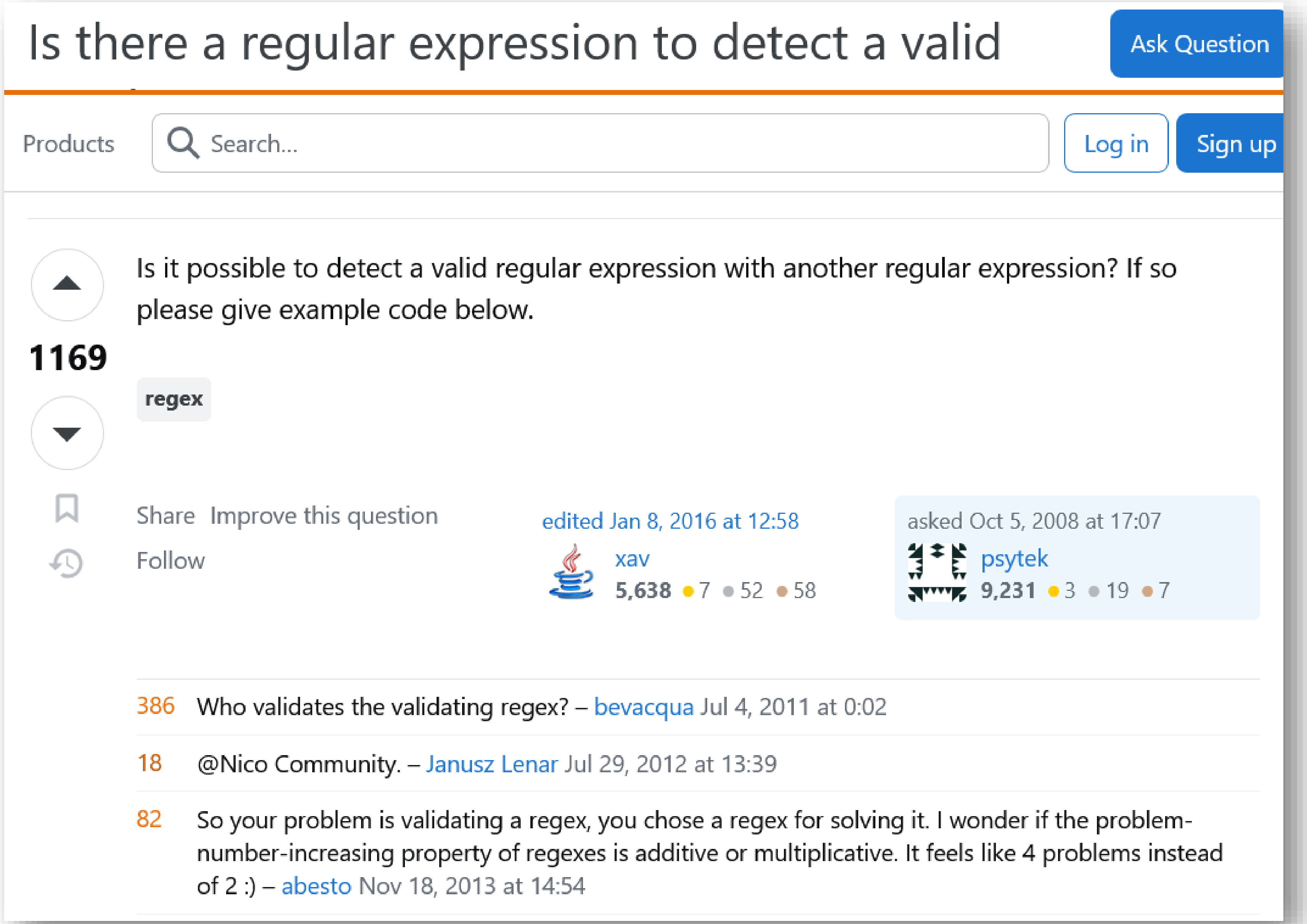
Is there a regular expression to detect a valid regular expression?

Is it possible to detect a valid **regular expression** with another **regular expression**? If so please give example code below. ...

regex

psytek **9,231** asked Oct 5, 2008 at 17:07

https://stackoverflow.com/search?q=regular+expressions


Is there a regular expression to detect a valid

Is it possible to detect a valid regular expression with another regular expression? If so please give example code below.

1169

regex

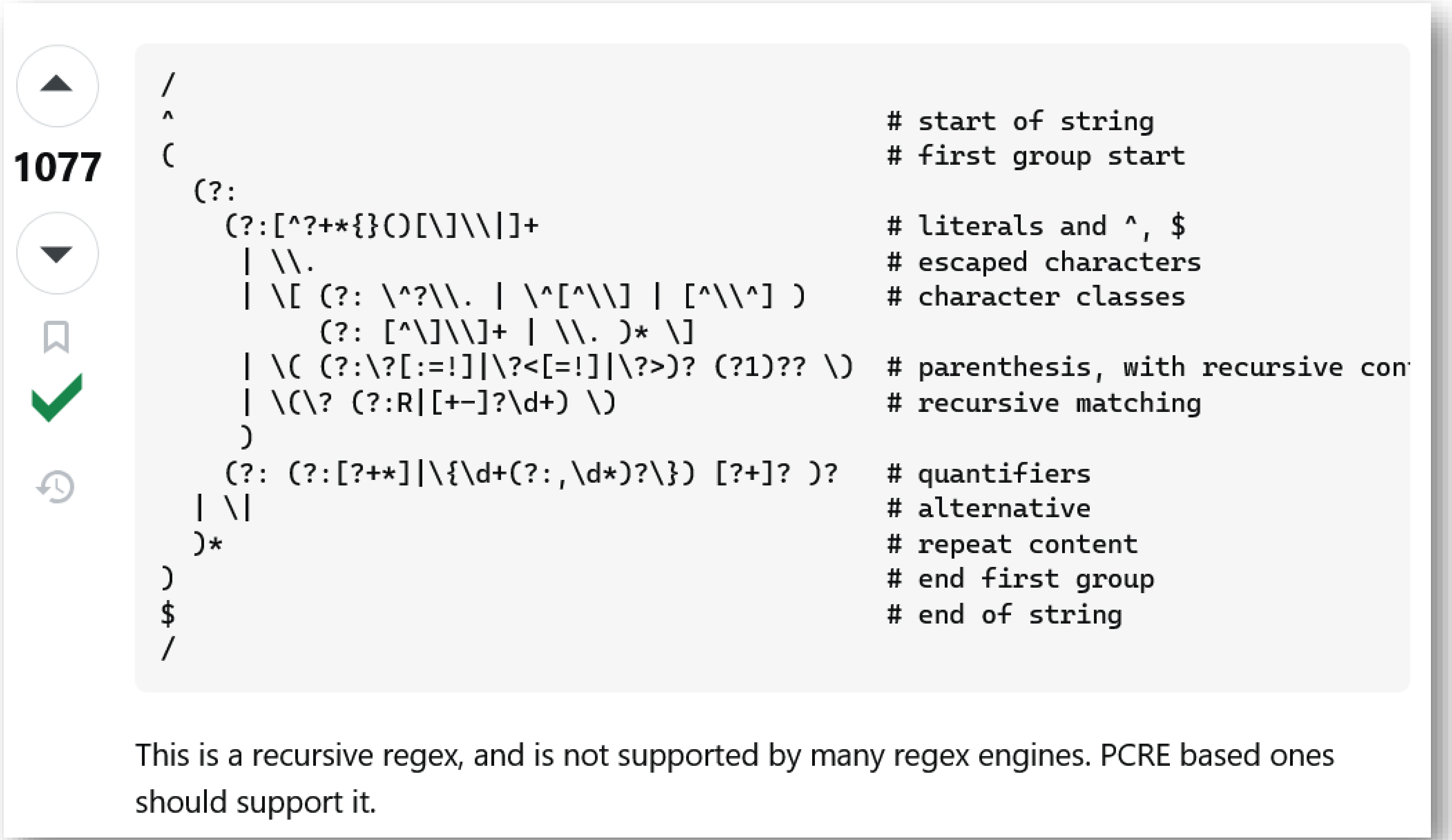
edited Jan 8, 2016 at 12:58
xav
5,638 ● 7 ● 52 ● 58

asked Oct 5, 2008 at 17:07
psytek
9,231 ● 3 ● 19 ● 7

386 Who validates the validating regex? – bevacqua Jul 4, 2011 at 0:02

18 @Nico Community. – Janusz Lenar Jul 29, 2012 at 13:39

82 So your problem is validating a regex, you chose a regex for solving it. I wonder if the problem-number-increasing property of regexes is additive or multiplicative. It feels like 4 problems instead of 2 :) – abesto Nov 18, 2013 at 14:54



```
/
^                                             # start of string
(                                             # first group start
  (?:
    (?:[^?+*{}()[\]\\|]+                      # literals and ^, $
     | \\.                                    # escaped characters
     | \[ (?: \^?\\. | \^[^\\] | [^\\^] )     # character classes
          (?: [^\]\\]+ | \\. )* \]
     | \( (?:\?[:=!]|\?<[=!]|\?>)? (?1)?? \)  # parenthesis, with recursive con
     | \(\? (?:R|[+-]?\d+) \)                 # recursive matching
     )
    (?: (?:[?+*]|\{\d+(?:,\d*)?\}) [?+]? )?   # quantifiers
  | \|                                        # alternative
  )*                                          # repeat content
)                                             # end first group
$                                             # end of string
/
```

This is a recursive regex, and is not supported by many regex engines. PCRE based ones should support it.

Write regular expression to match (and only match) valid regular expressions

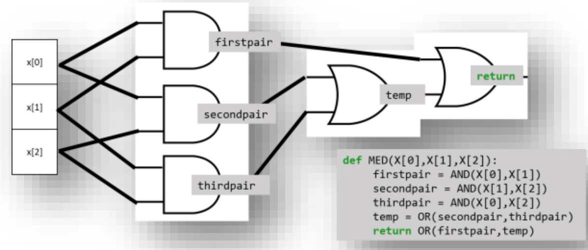
Is it possible?

$\{0^n 1^n$: for some integer $n\}$ is not regular

$\{(^n)^n$: for some integer $n\}$ is not regular

$\{x \in \{(,)\}^*$: parathesis in x is paired $\}$ is not regular

Computation models (so far)

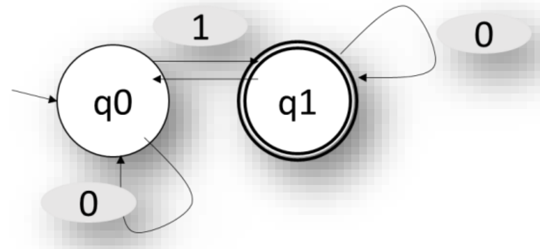


```
def MED(X[0],X[1],X[2]):
    firstpair = AND(X[0],X[1])
    secondpair = AND(X[1],X[2])
    thirdpair = AND(X[0],X[2])
    temp = OR(secondpair,thirdpair)
    return OR(firstpair,temp)
```

Boolean circuits:

☺ Can compute every function on fixed input length

☹ Need a different circuit for each input length.



DFA (and RE, NFA):

☺ Can describe one algorithm/method for all inputs

☹ It is limited in how it accesses its inputs (one pass)

☹ It is limited in how much memory it has (constant)

Can we get the best of both

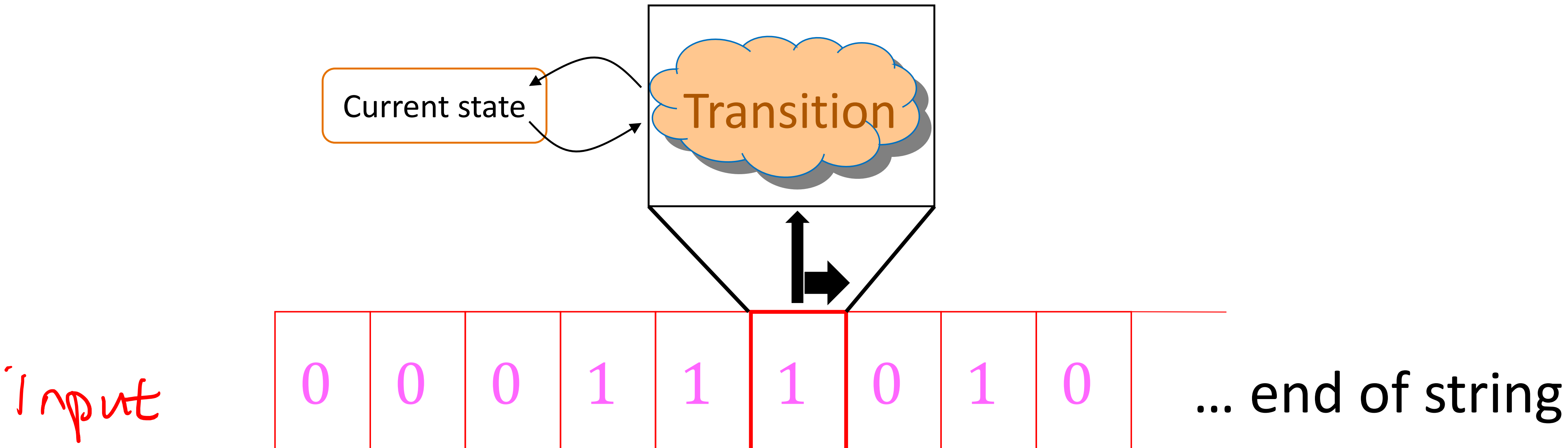
Turing Machines!

☺ A single algorithm for all input lengths

☺ Can read its input back and forth

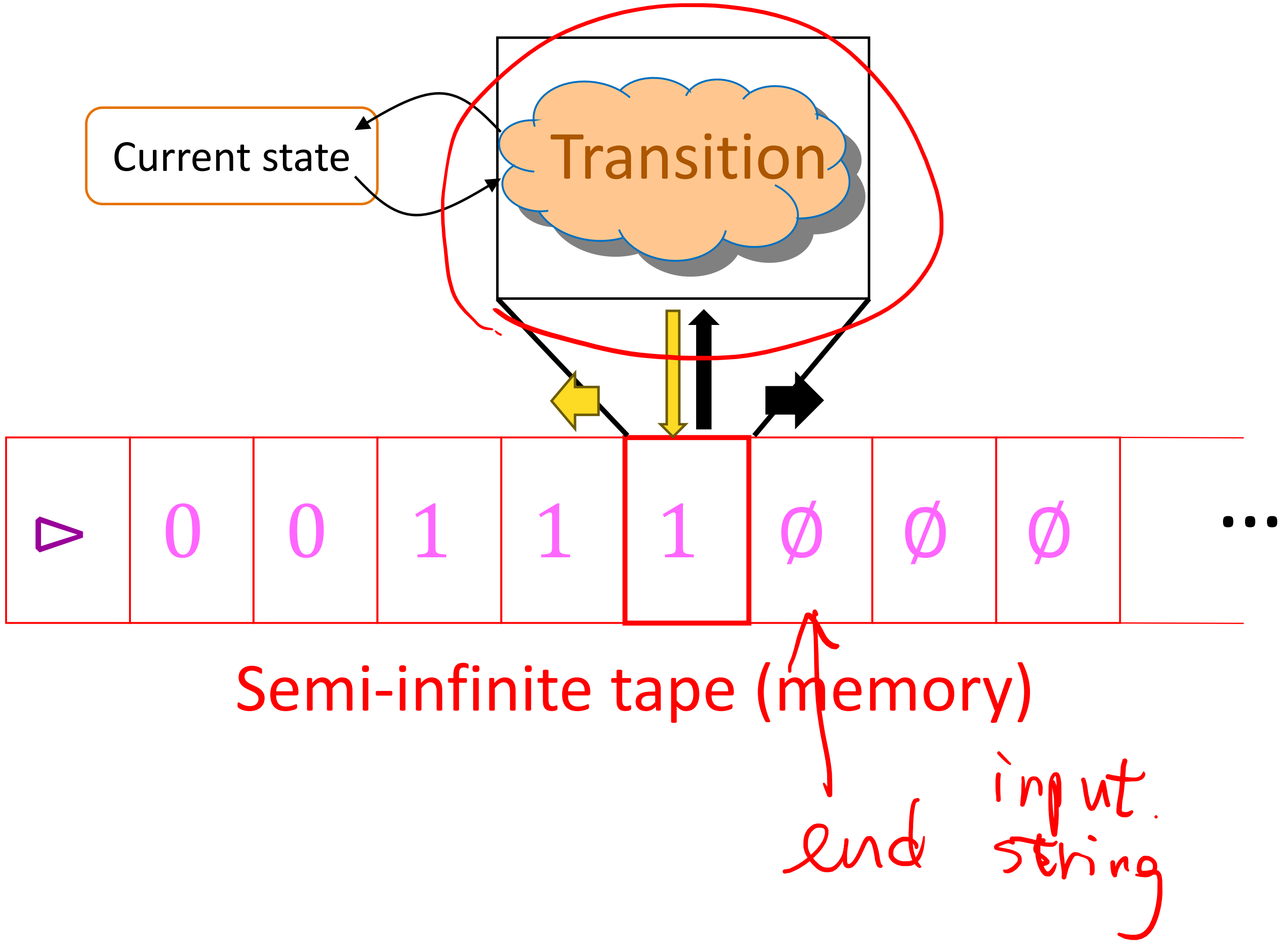
☺ Can store data and read back (storage as memory)

How DFA worked



Turing Machine

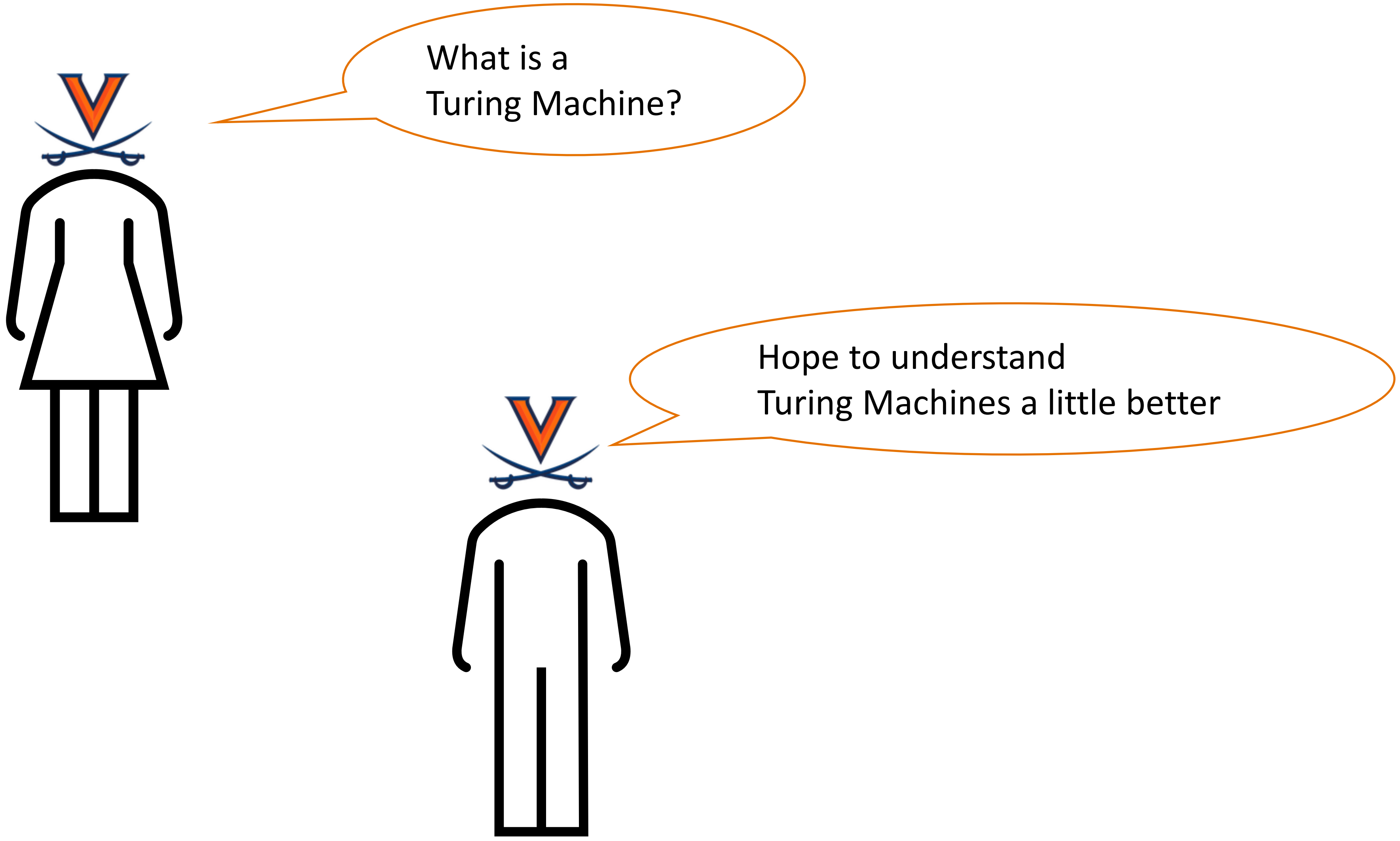
Tape initially contains the input followed by blanks

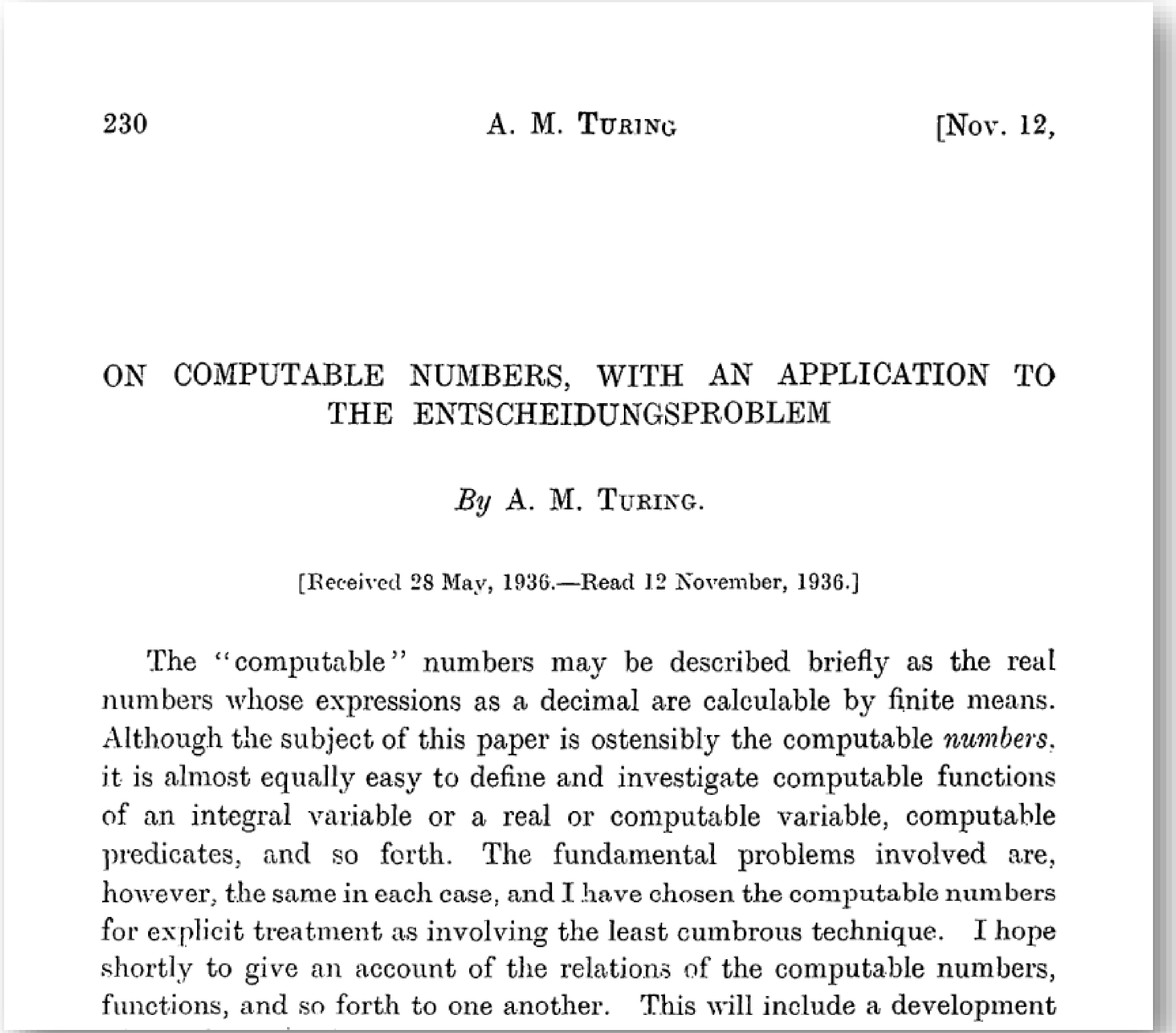


If DFA "halts", → tape contains the output

Turing Machines

Registration Survey


What is a Turing Machine?
Hope to understand Turing Machines a little better



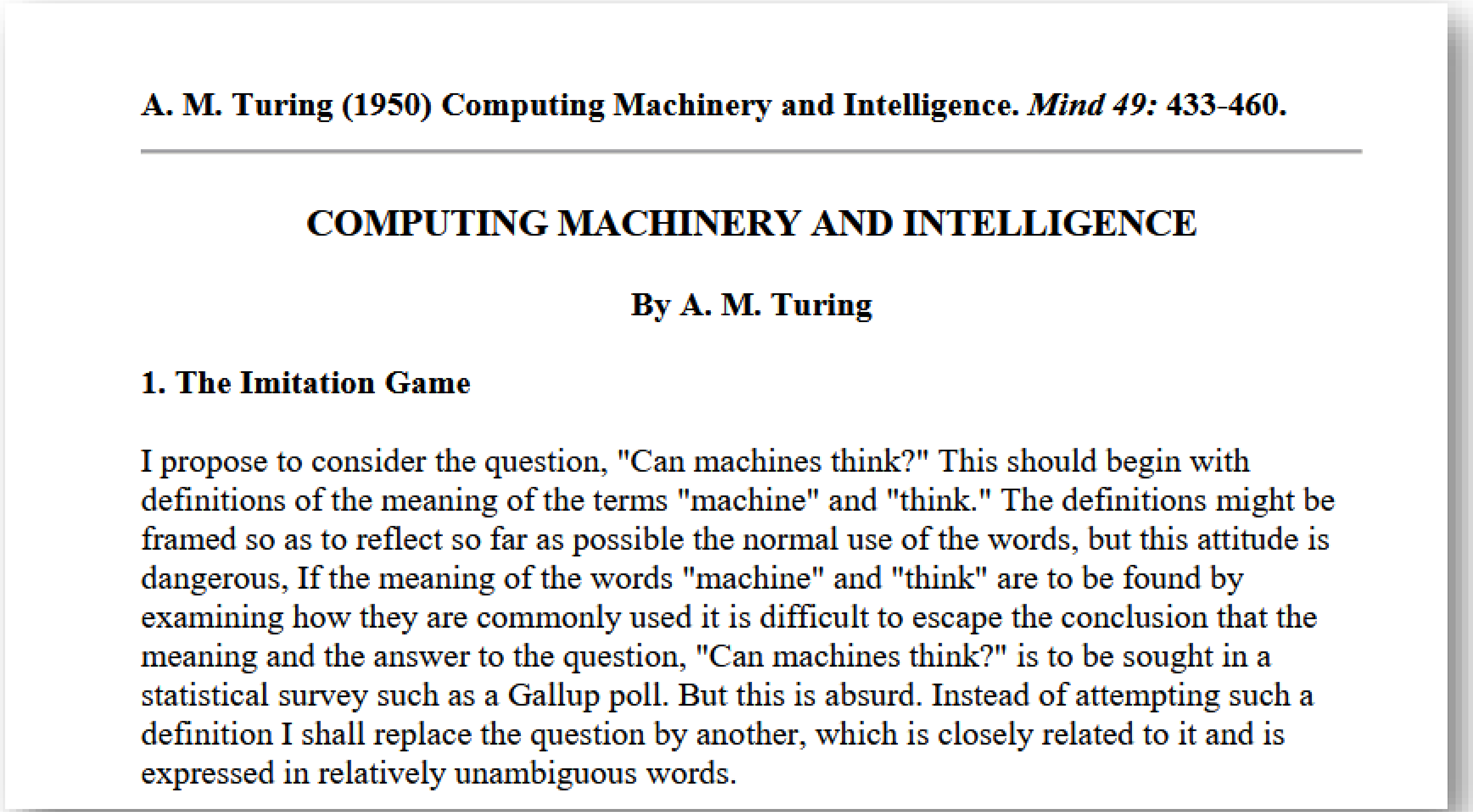
ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development

https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf



A. M. Turing (1950) Computing Machinery and Intelligence. *Mind 49:* 433-460.

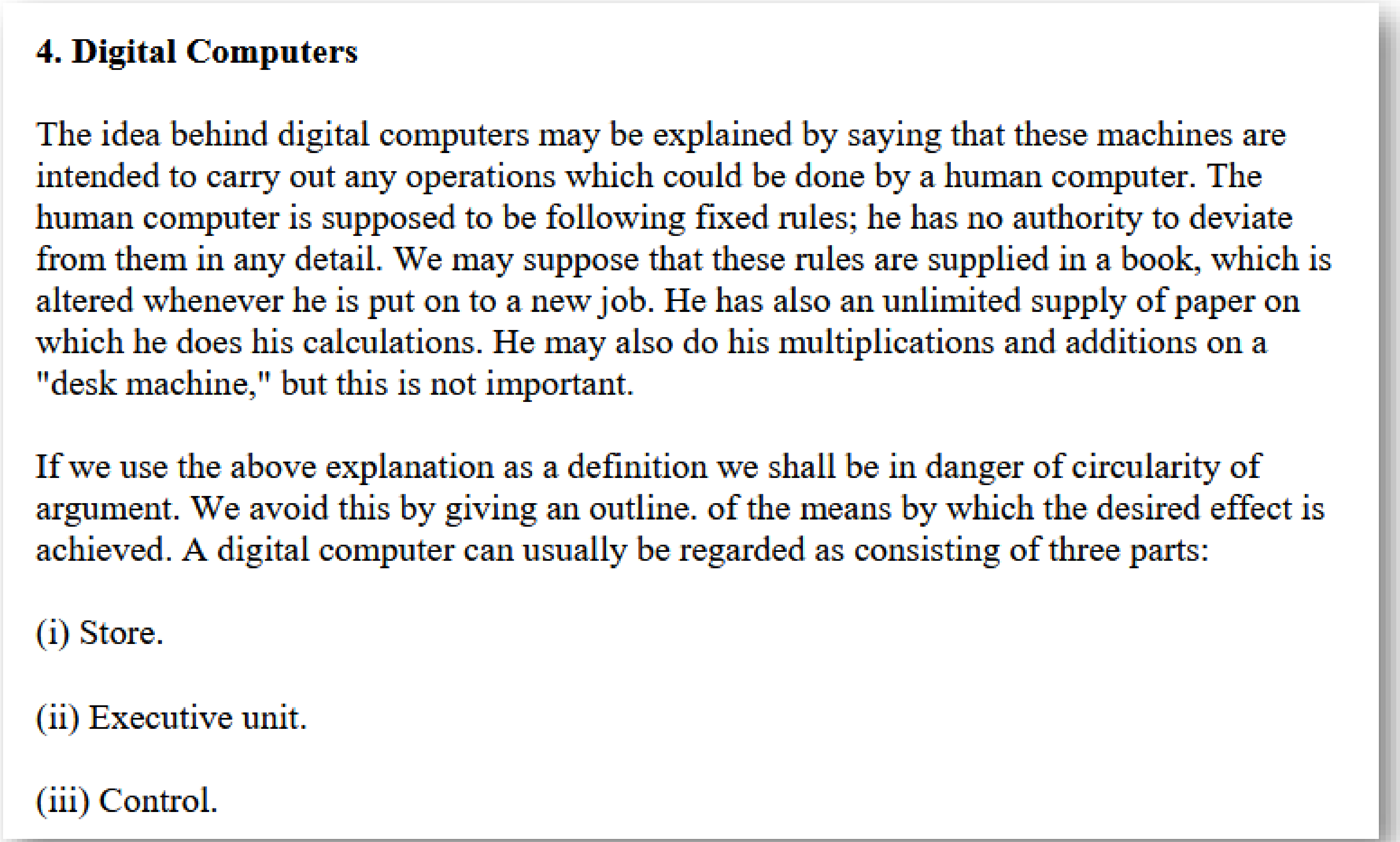
COMPUTING MACHINERY AND INTELLIGENCE

By A. M. Turing

1. The Imitation Game

I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous, If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

https://courses.cs.umbc.edu/471/papers/turing.pdf



4. Digital Computers

The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer. The human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail. We may suppose that these rules are supplied in a book, which is altered whenever he is put on to a new job. He has also an unlimited supply of paper on which he does his calculations. He may also do his multiplications and additions on a "desk machine," but this is not important.

If we use the above explanation as a definition we shall be in danger of circularity of argument. We avoid this by giving an outline. of the means by which the desired effect is achieved. A digital computer can usually be regarded as consisting of three parts:

(i) Store.

(ii) Executive unit.

(iii) Control.

What are we *modeling*?

Local [Follow simple rules

transition function — finite fun

Global [Remember what you are doing

memory inf

Turing Machine Model

A *Turing Machine,* is defined by (Σ, k, δ):

$\boldsymbol{k} \in \mathbb{N}$: a finite number of $\underline{\text{states}}$ $\{0, 1, \ldots, k-1\}$.

Σ : alphabet − finite set of symbols

$$\Sigma \supseteq \{\underline{0, 1}, \underline{\rhd, \emptyset}\}$$

$\longrightarrow$ special symbols.

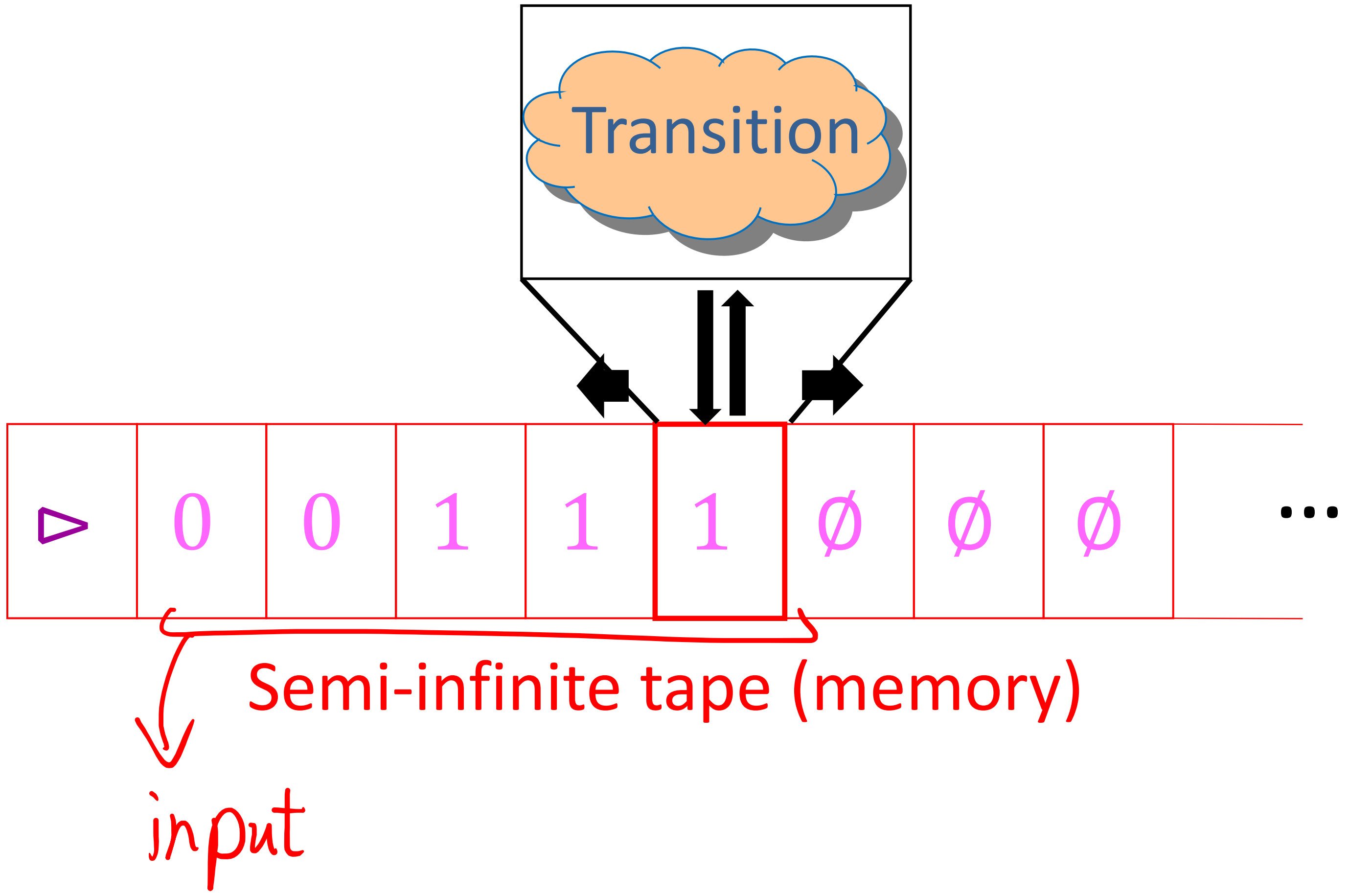
$\boldsymbol{\delta}$: transition function

$$\boldsymbol{\delta}: [k] \times \Sigma \rightarrow \underline{[k]} \times \boxed{\Sigma \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}, \mathbf{H}\}}$$

Left, **R**ight, **S**tay, **H**alt

Turing Machine

Tape initially contains the input followed by blanks



A *Turing Machine,* is defined by (Σ, k, δ):
$\boldsymbol{k} \in \mathbb{N}$: a finite number of states
Σ : alphabet $-$ finite set of symbols
$$\Sigma \supseteq \{0, 1, \rhd, \emptyset\}$$
$\boldsymbol{\delta}$: transition function
$$\boldsymbol{\delta}: [k] \times \Sigma \rightarrow [k] \times \Sigma \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}, \mathbf{H}\}$$

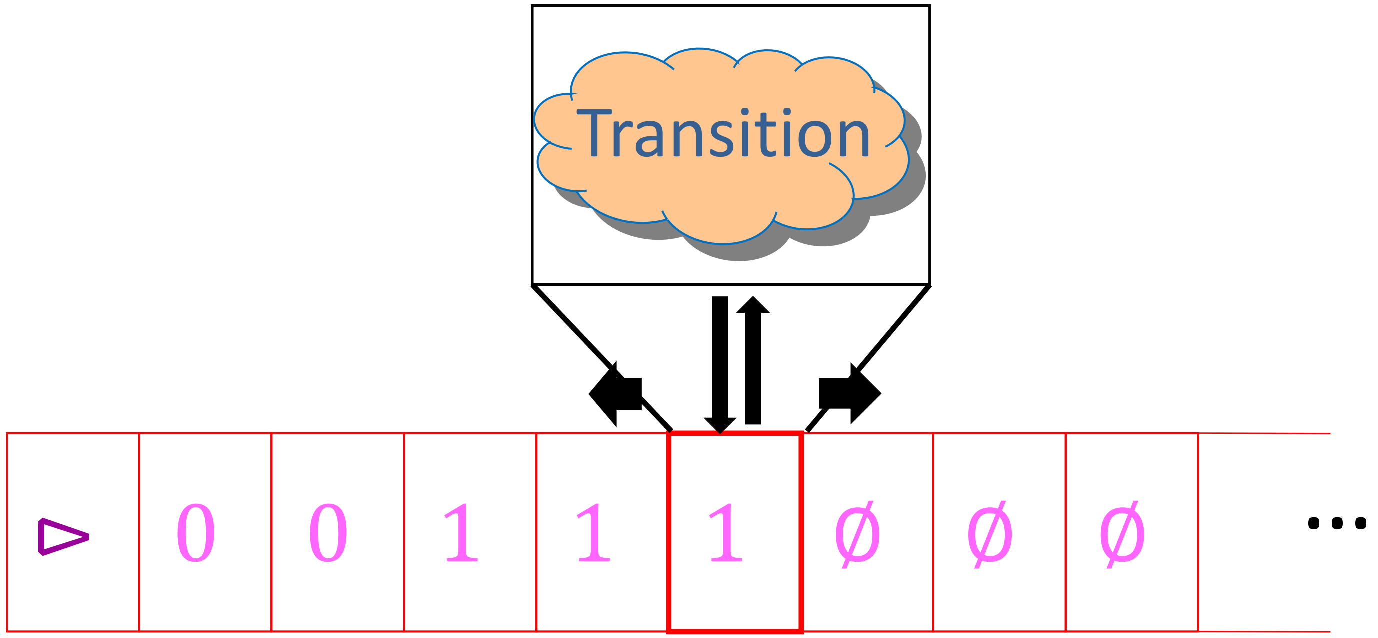
Execution of a TM

Definition. The execution of a TM, $M = (\Sigma, k, \delta)$ on input $x \in \{0, 1\}^*$ is this process:

```
1. Initialize $T$ as $\rhd, x_0, x_1, \dots, x_{n-1}, \emptyset, \emptyset, \emptyset, \dots$ where $n = |x|$.
        That is, $T[0] = \rhd$, $T[i+1] = x_i$ for $i \in [n]$, and $T[i] = \emptyset$ for $i > n$.
2. Initialize two natural number variables, $i = 0, s = 0$.
3. **repeat**
        1. $(s', \sigma', D) = \boldsymbol{\delta}(s, T[i])$
        2. $s := s', T[i] := \sigma'$
        3. if $D = \mathbf{R}$: $i := i + 1$
           if $D = \mathbf{L}$: $i := \max\{i - 1, 0\}$
           if $D = \mathbf{H}$: $\mathbf{break}$
```

Turing Machine

Tape initially contains the input followed by blanks



Semi-infinite tape (memory)

If transition "halts", → tape contains the output

Output of executing TM

Definition. The **output** of the execution of a TM, $M = (\Sigma, k, \delta)$ on input $x \in \{0,1\}^*$ is the result of this process:

```
1. Initialize $T$ as $\triangleright, x_0, x_1, \ldots, x_{n-1}, \emptyset, \emptyset, \emptyset, \ldots$ where $n = |x|$.
        That is, $T[0] = \triangleright$, $T[i+1] = x_i$ for $i \in [n]$, and $T[i] = \emptyset$ for $i > n$.
2. Initialize two natural number variables, $i = 0, s = 0$.
3. **repeat**
        1. $(s', \sigma', D) = \boldsymbol{\delta}(s, T[i])$
        2. $s := s', T[i] := \sigma'$
        3. if $D$ = $\mathbf{R}$: $i := i + 1$
           if $D$ = $\mathbf{L}$: $i := \max\{i - 1, 0\}$
           if $D$ = $\mathbf{H}$: $\mathbf{break}$
4. If the process finishes (the repeat breaks), the result of this process is:
        $M(x) = \underbrace{T[1], \ldots, T[m]}$ where $m > 0$ is the smallest integer, $T[m+1] \underline{\notin \{0,1\}}$.
   Otherwise (i.e., the machine does NOT halt), then define $M(x) = \perp$. bot(tom)
   [warning: $\perp$ is not an actual output]
```

Special Simpler Case: No Input Execution

Definition. The **output** of the execution of a TM, $M = (\Sigma, k, \delta)$ is the result of this process: $x = $ " "

$M($" "$)$

$M(\ )$

1. Initialize T as $\underline{\triangleright, \emptyset, \emptyset, \emptyset, \ldots}$
2. Initialize two natural number variables, $i = 0, s = 0$.
3. **repeat**

```
    1. $(s', \sigma', D) = \boldsymbol{\delta}(s, T[i])$
    2. $s := s', T[i] := \sigma'$
    3. if $D$ = $\mathbf{R}$: $i := i + 1$
       if $D$ = $\mathbf{L}$: $i := \max\{i - 1, 0\}$
       if $D$ = $\underline{\mathbf{H}\text{: }\mathbf{break}}$
```

4. If the process finishes (the repeat breaks), the result of this process is:

$$M(x) = T[1], \ldots, T[m] \text{ where } m > 0 \text{ is the smallest integer, } T[m+1] \notin \{0,1\}.$$

Otherwise (i.e., the machine does NOT halt), then define $M(x) = \perp$.
[warning: $\perp$ is not an actual output]

Who needs input?

Given a TM, $M = (\Sigma, k, \delta)$ and input x, produce a TM, $M' = (\Sigma, k', \delta')$ such that the output $\underline{M'(x)}$ in the $\underline{\text{No-Input}}$ model, is the same as the output $\underline{M(x)}$ in the normal model.

$M'(" ")$

M computes on x

Why always possible?

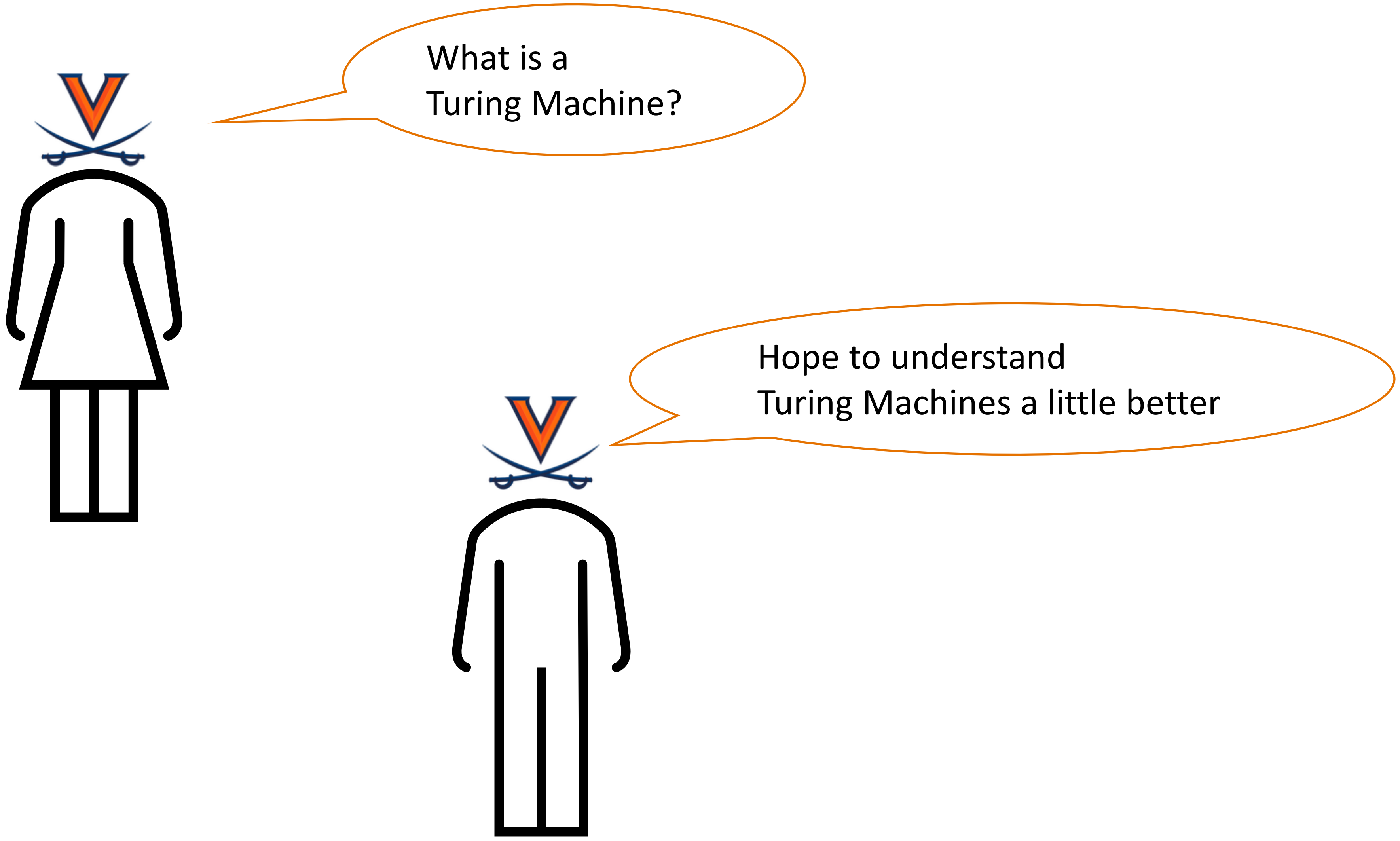
$M'( ) = M(x)$

$x \left(\begin{array}{l} y = M(x) \text{ if halts} \\ M' \text{ s.t. } M'() \text{ output } y \end{array} \right)$

M_x s.t. $M_x()$ ~~write~~ output x

M on the same tape.

Registration Survey



What's a Turing Machine?

A *Turing Machine,* is defined by (Σ, k, δ):
$\boldsymbol{k} \in \mathbb{N}$: a finite number of states
Σ : alphabet – finite set of symbols
$$\Sigma \supseteq \{0, 1, \rhd, \emptyset\}$$
$\boldsymbol{\delta}$: transition function
$$\boldsymbol{\delta}: [k] \times \Sigma \to [k] \times \Sigma \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}, \mathbf{H}\}$$

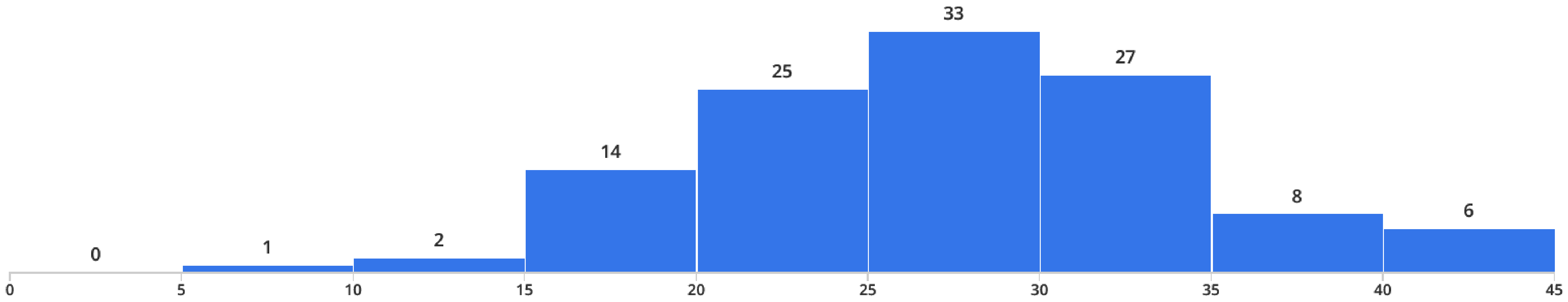
- Extension of DFA $\quad \supseteq$ DFA
  Read/write many times.
- Inf amount memory
  Computes any finite func $\supseteq$ Circuit
- Limit length of Mem/Tape : SPACE hierarchy. ?

Simulating Turing Machines

http://morphett.info/turing/turing.html

```
0 0 0 r 0
0 1 1 r b
0 _ _ * halt-accept
b 0 0 r b
b 1 1 r b
b _ _ * halt-reject
```

Midterm statistics



Minimum	Median	Maximum	Mean	Std Dev
9.0	27.5	45.0	26.89	7.04

Charge

Regular expressions

Limits, failures

Turing Machines

Photo credit

PS7 due next Tue (April 1)
Coming soon: PS8, PRR9