The CS 5 Post

ALIEN INVASION!!!

Claremont (AP): A party at a private college here was disrupted when uninvited aliens burst through the gates.

"Every year, we celebrate *Long Tall Penguins*," explained an angry student. "We get together, dress like the stuffiest professors, and chip bits off an iceberg to cool our drinks. This year, just as we were about to chill the mackerel, two strange alien creatures ran into the courtyard, picked everyone up, and took turns stacking us in piles."

But another student claimed that the aliens were just misunderstood. "They love to play Connect 4, and since we were wearing black and white clothes, they thought we were playing pieces. They stacked us up in a 5-ply lookahead formation. It was fun!"

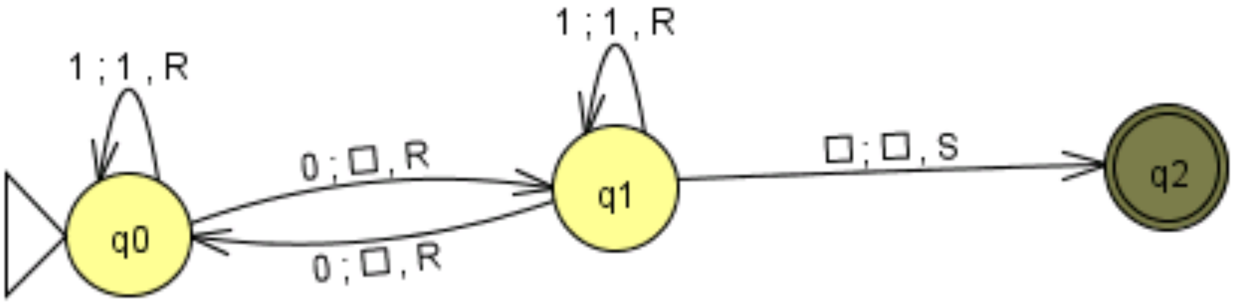
According to police, no charges will be filed because the aliens are not subject to Solar jurisdiction.

CS 5 today: more machines!

Final ideas...

Turing Machines and the **MANY** things computers can't compute... !

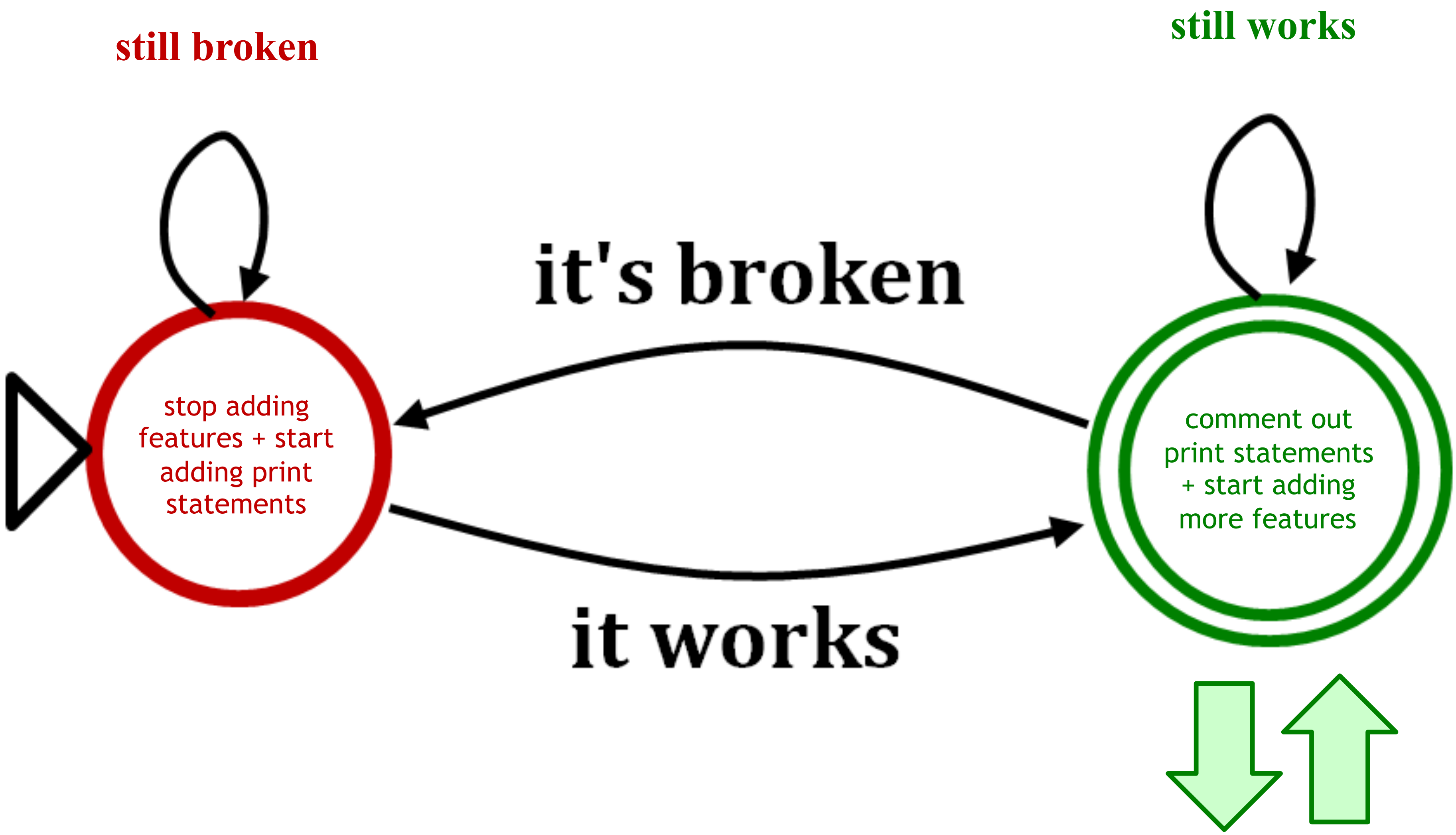
This machine doesn't look all-powerful to me!



- hw12 + milestone due on Monday, 12/4
- 5 finite-state machines due as part of hw12

Turing machines extra!

Final project state machine



still broken

still works

it's broken

stop adding features + start adding print statements

comment out print statements + start adding more features

it works

store a copy
somewhere else!

State-machine *limits*?

Are there limits to what FSMs can do?

they can't necessarily drive safely...

MIT's car, Talos

An autonomous vehicle's FSM

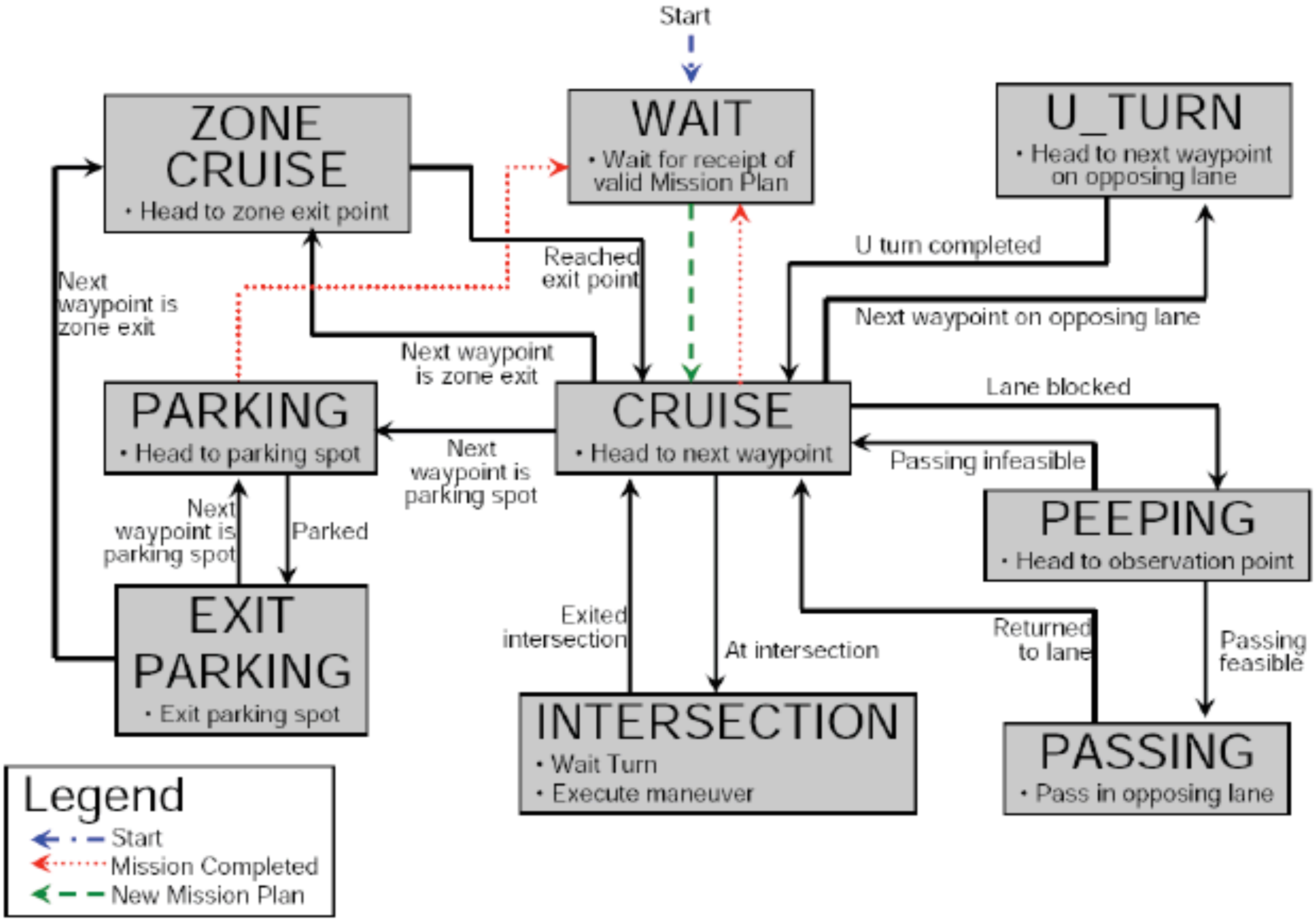


Fig. 9. Situational Interpreter State Transition Diagram. All modes are sub-modes of the system RUN mode (Fig 4(b)).

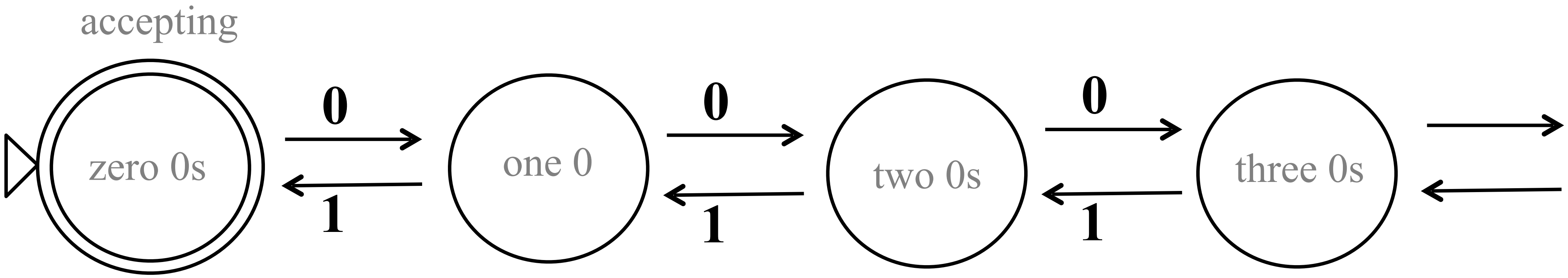
The data driving the state machine...

MIT's car,
Talos - ***and***
its sensor
suite

But are there any binary-string problems that FSMs can't solve?

State-machine ***limits***?

You don't need three
eyes to see some
problems here!



Let's build a FSM that accepts strings with
any **# of 0s** followed by the ***same*** **#of 1s**

rejected

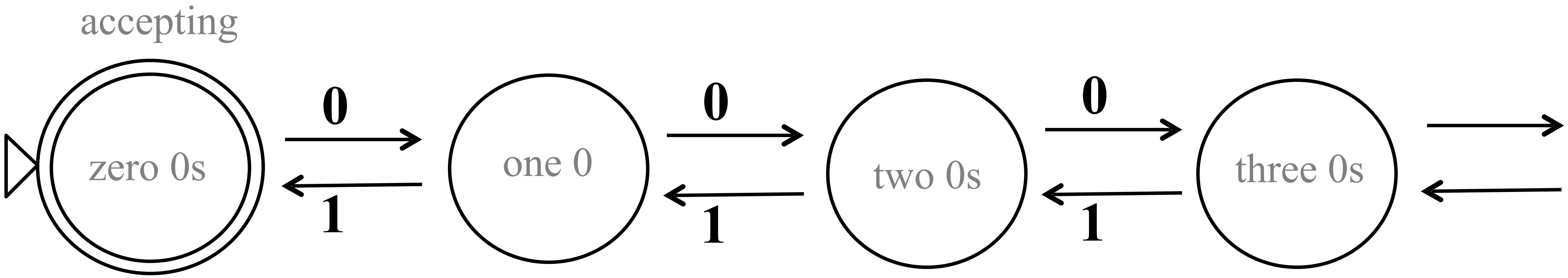
011
001
11100
00110

accepted

000111
0011
01
λ

State-machine ***limits***?

You don't need three
eyes to see some
problems here!



Let's build a FSM that accepts strings with
any **# of 0s** followed by the ***same*** **#of 1s**

rejected

011

001

11100

00110

FSMs "can't count"

at least, not arbitrarily high

accepted

000111

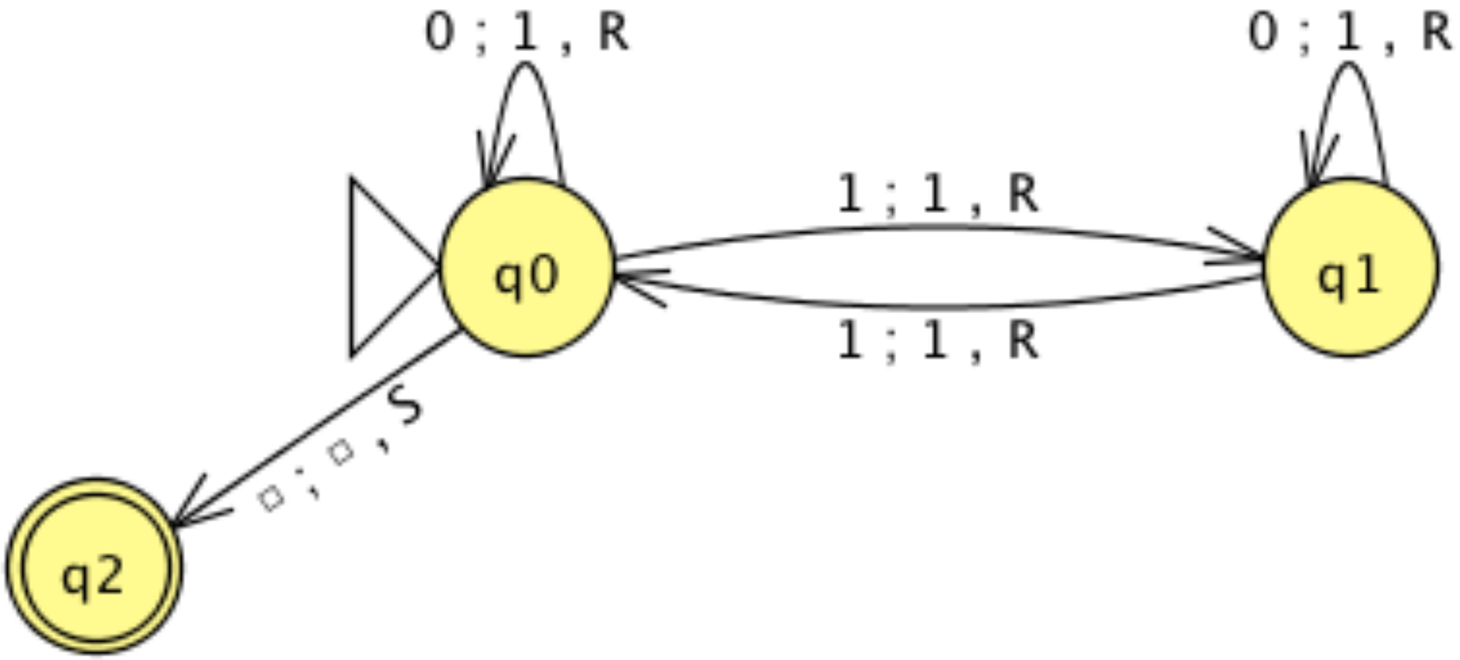
0011

01

λ

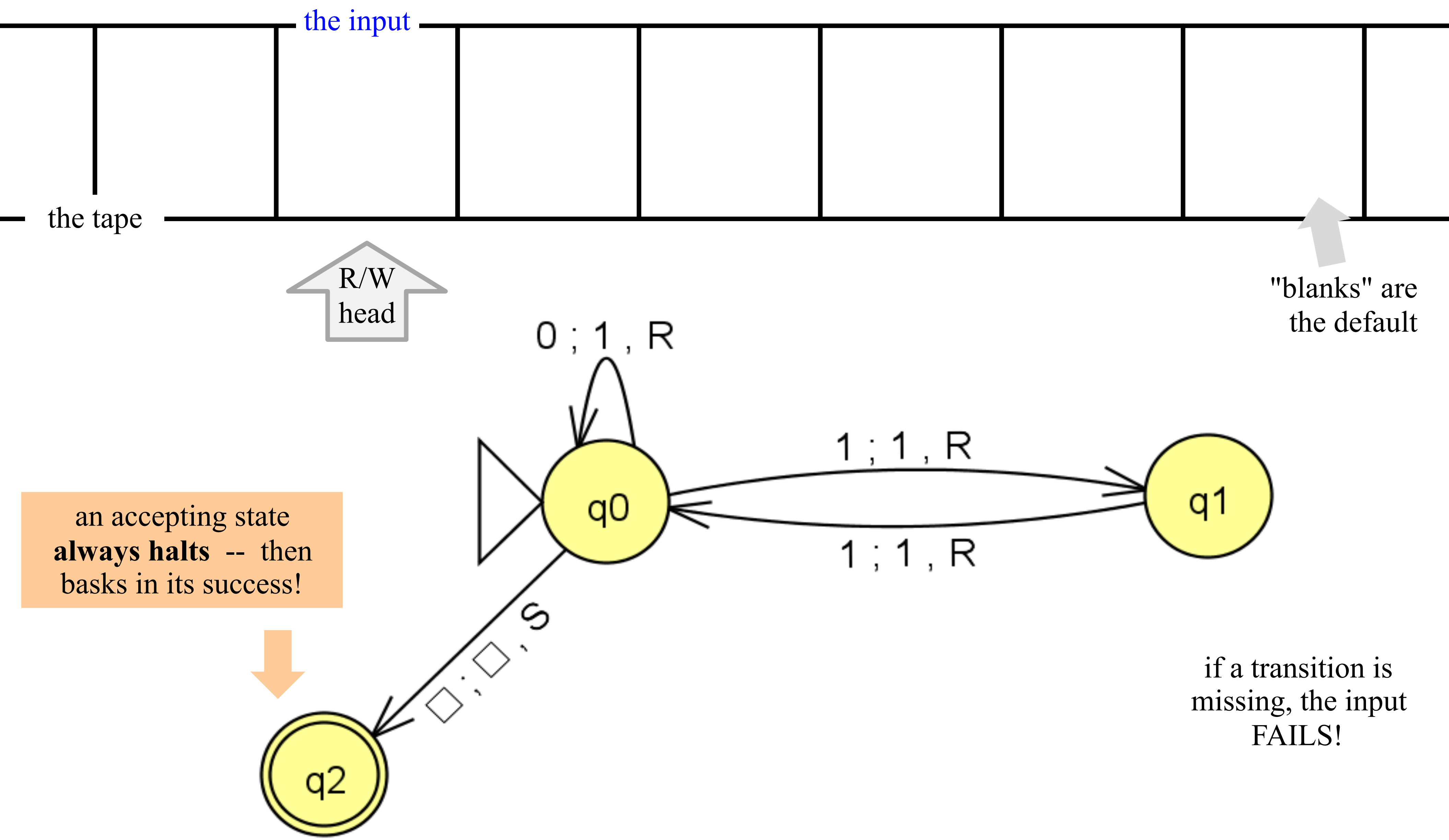
FSMs can't ***count***...

So, let's build a better machine!



Shiny!

Turing Machine



"blanks" are the default

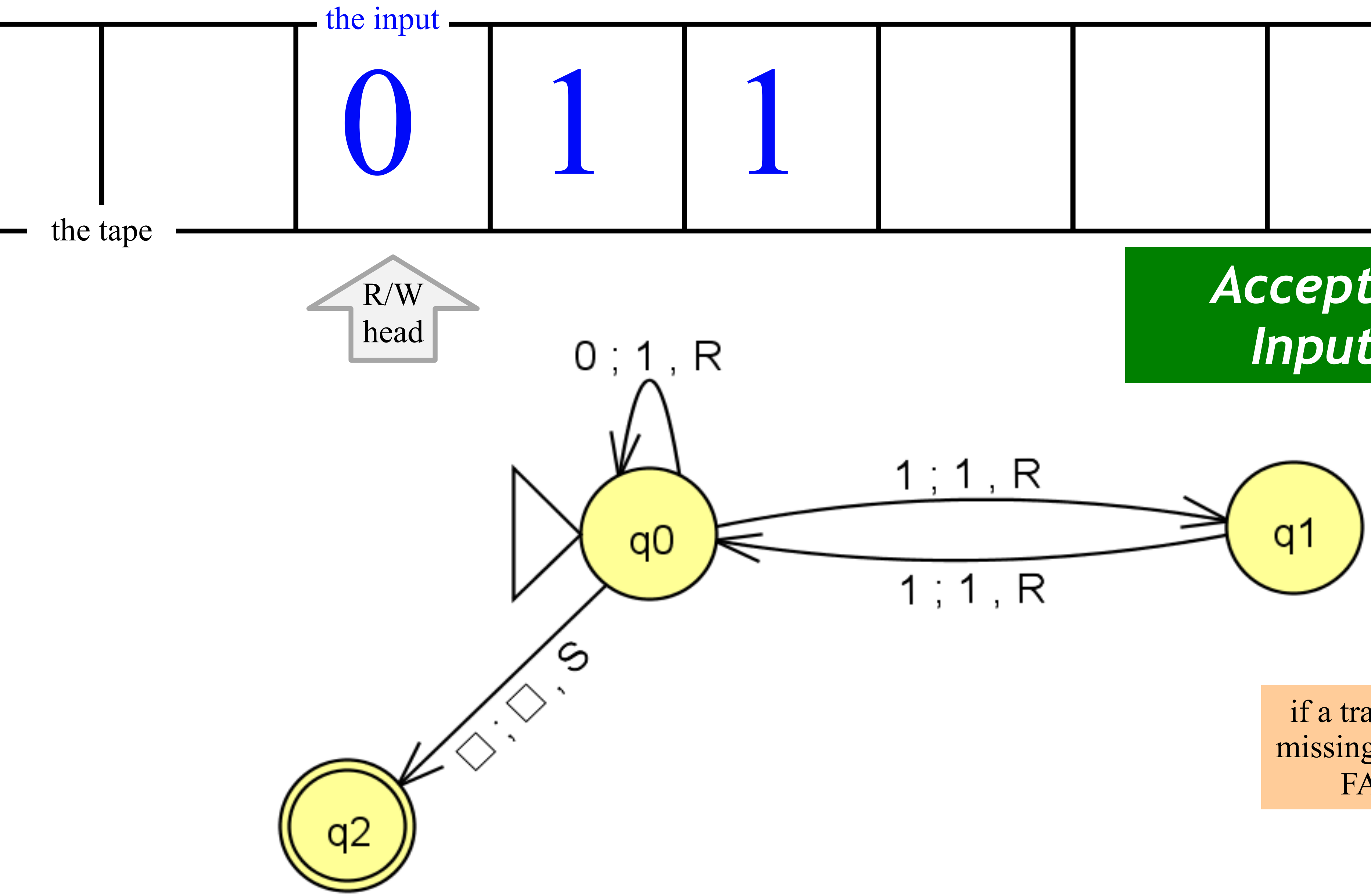
an accepting state **always halts** -- then basks in its success!

if a transition is missing, the input FAILS!

a Turing Machine rule: 0 ; 1 , R

READ WRITE MOTION

try it in JFLAP...



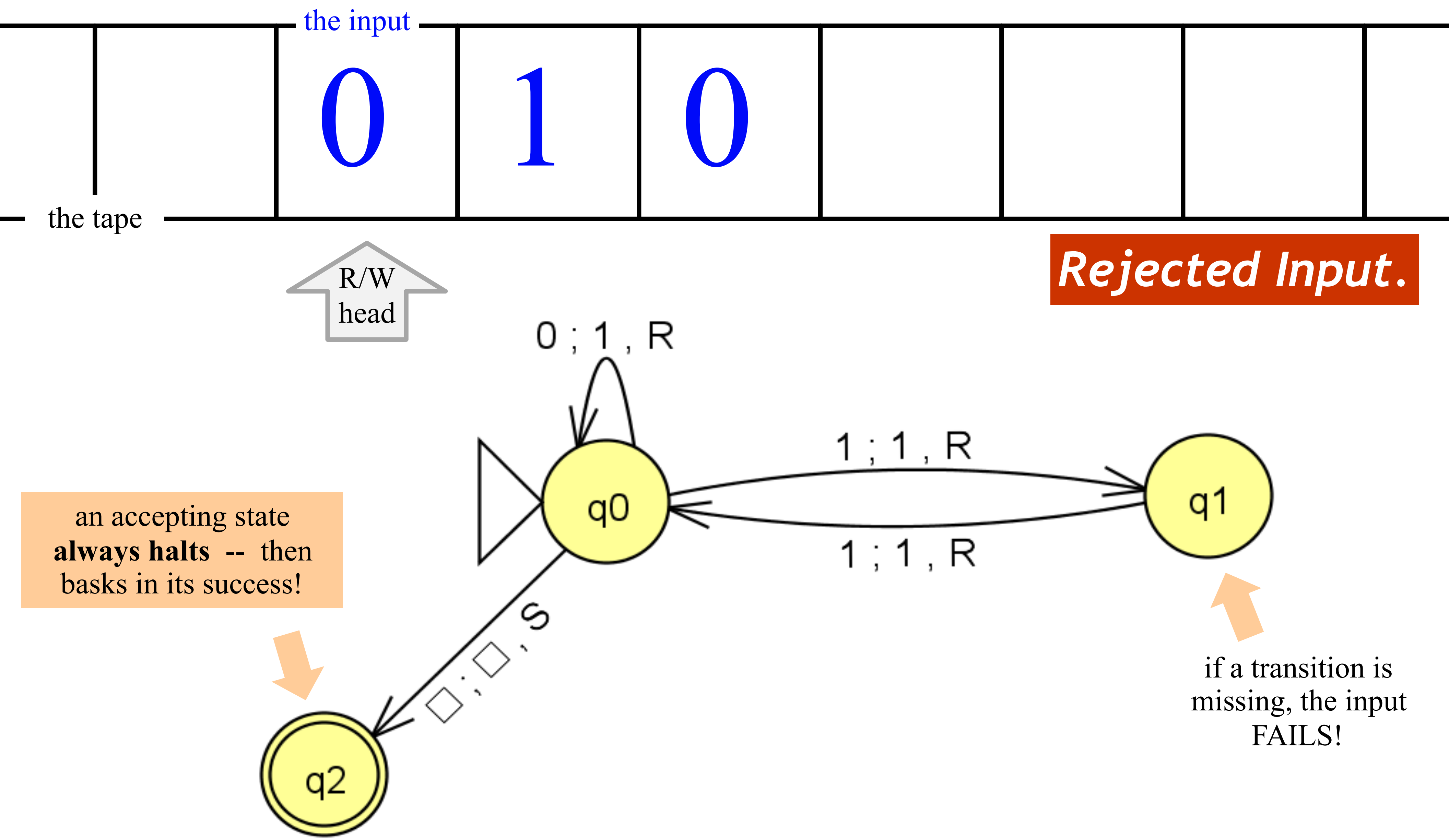
Accepted Input!

if a transition is missing, the input FAILS!

a Turing Machine rule:

0	;	1	,	R
READ		WRITE		MOTION

try it in JFLAP...



Rejected Input.

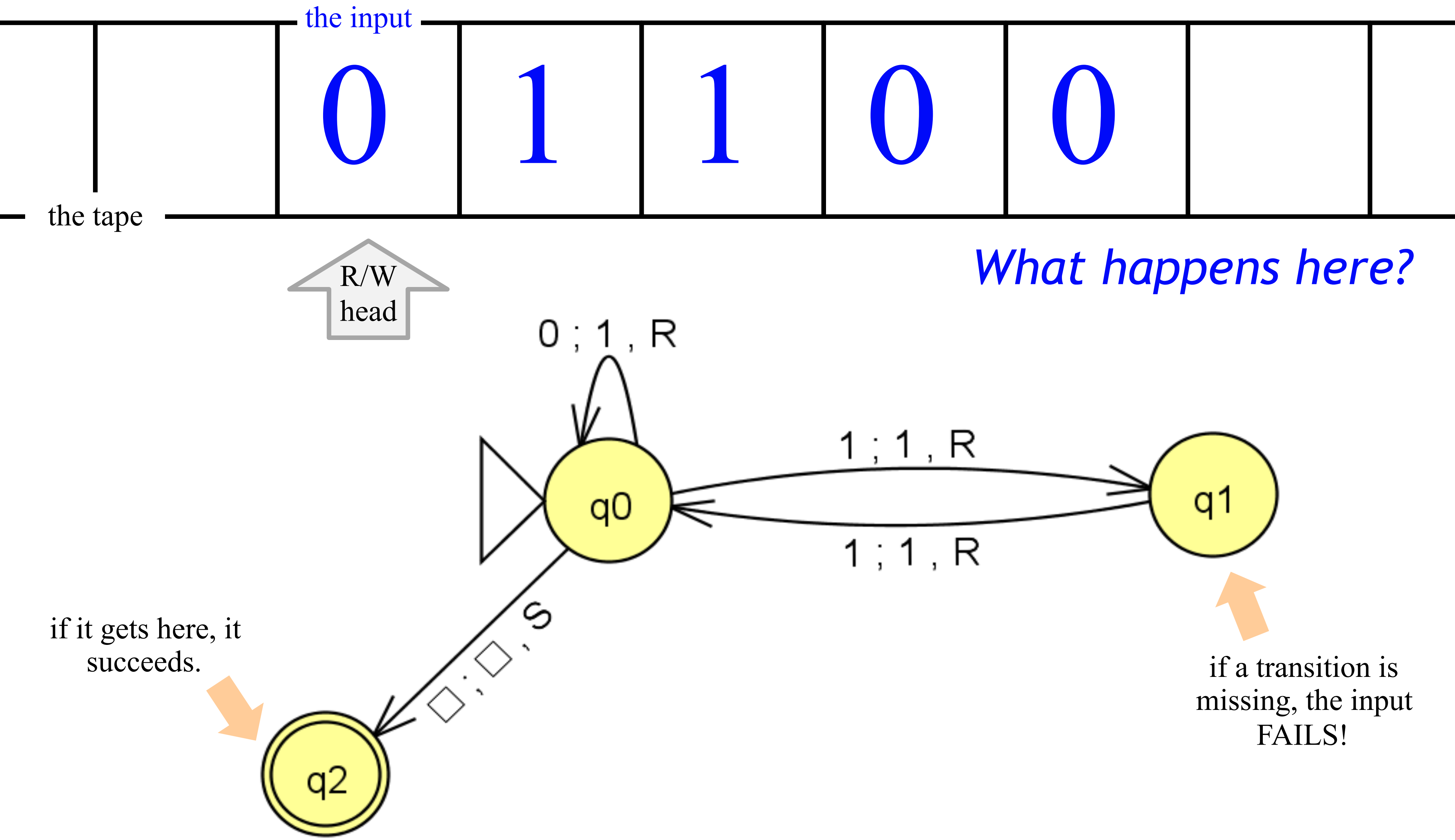
an accepting state **always halts** -- then basks in its success!

if a transition is missing, the input FAILS!

a Turing Machine rule: 0 ; 1 , R

READ WRITE MOTION

try it in JFLAP...



What happens here?

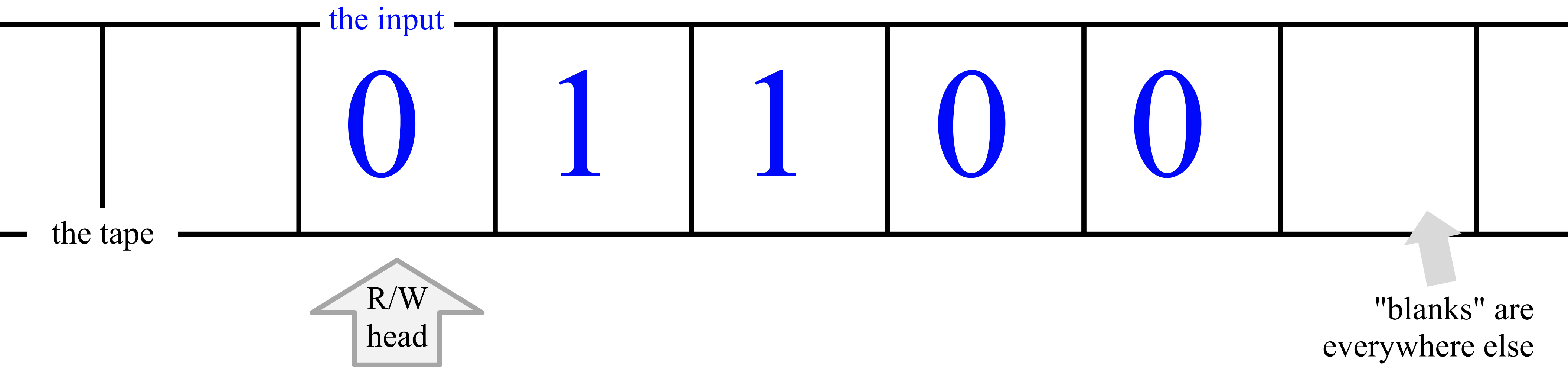
if it gets here, it succeeds.

if a transition is missing, the input FAILS!

a Turing Machine rule: 0 ; 1 , R

READ WRITE MOTION

try it in JFLAP...

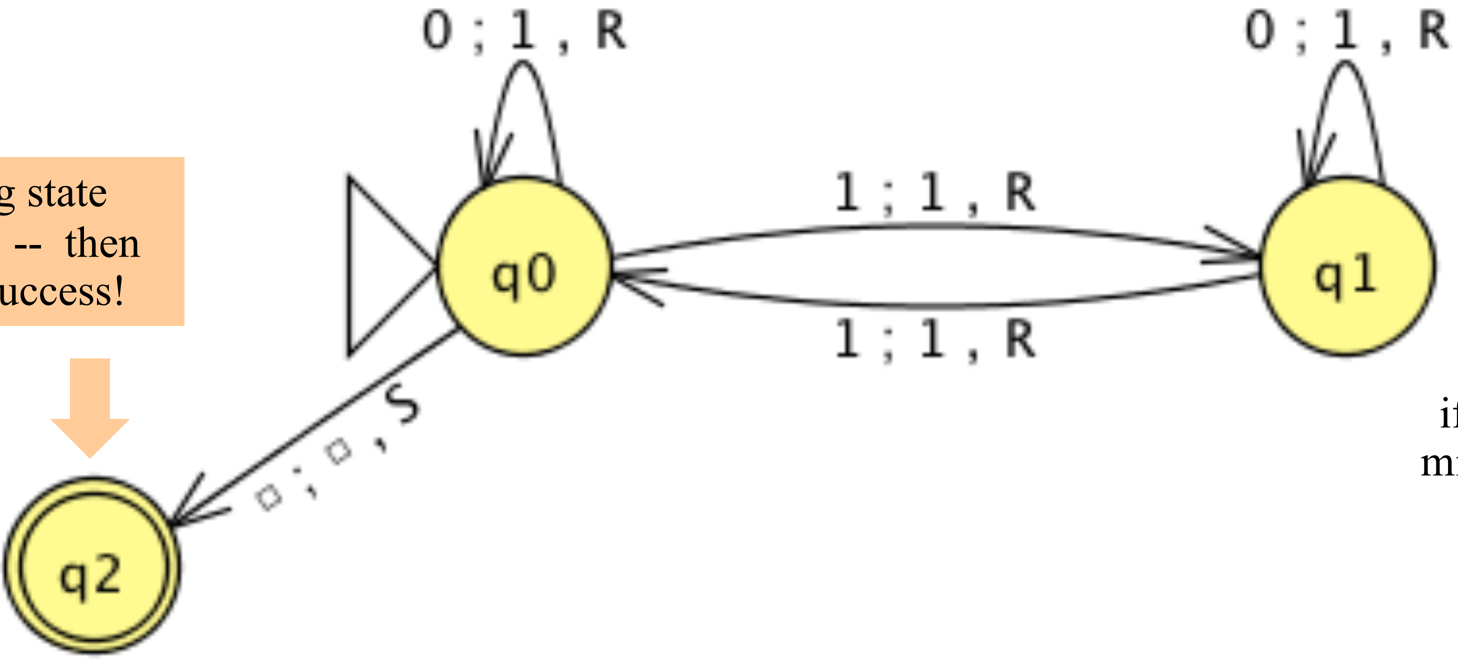


the input

the tape

R/W head

"blanks" are everywhere else



an accepting state **always halts** -- then basks in its success!

if a transition is missing, the input FAILS!

a Turing Machine rule: 0 ; 1 , R

READ	WRITE	MOTION
0	1	R

try it in JFLAP...

CS 5 spokesperson of the day!

The Imitation Game (2014)

30

PG-13 114 min - Biography | Drama | Thriller -

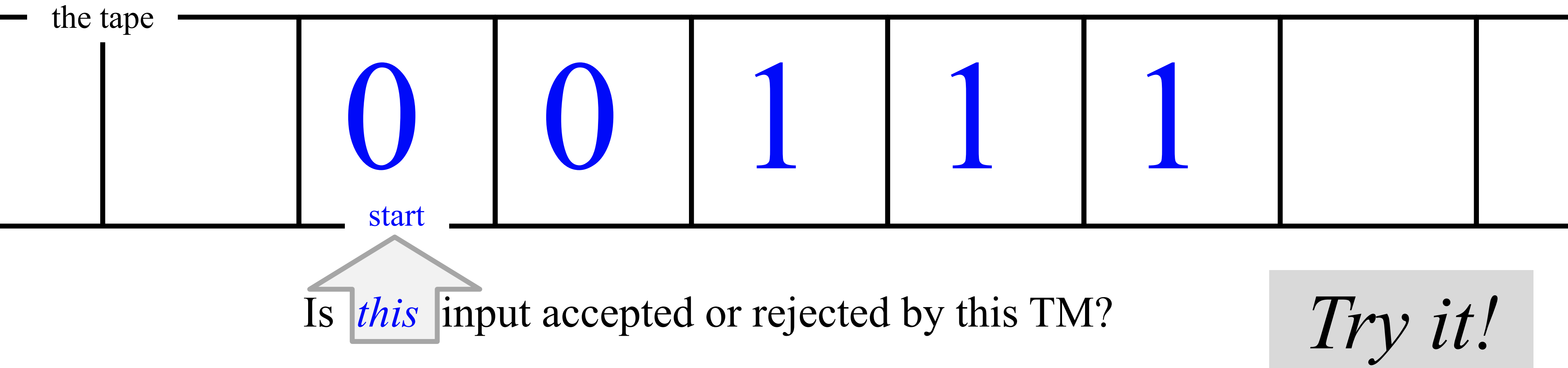
25 December 2014 (USA)

Your rating: ★★★★★★★★★★ -/10

8.4

Ratings: 8.4/10 from 8,470 users Metascore: 71/100

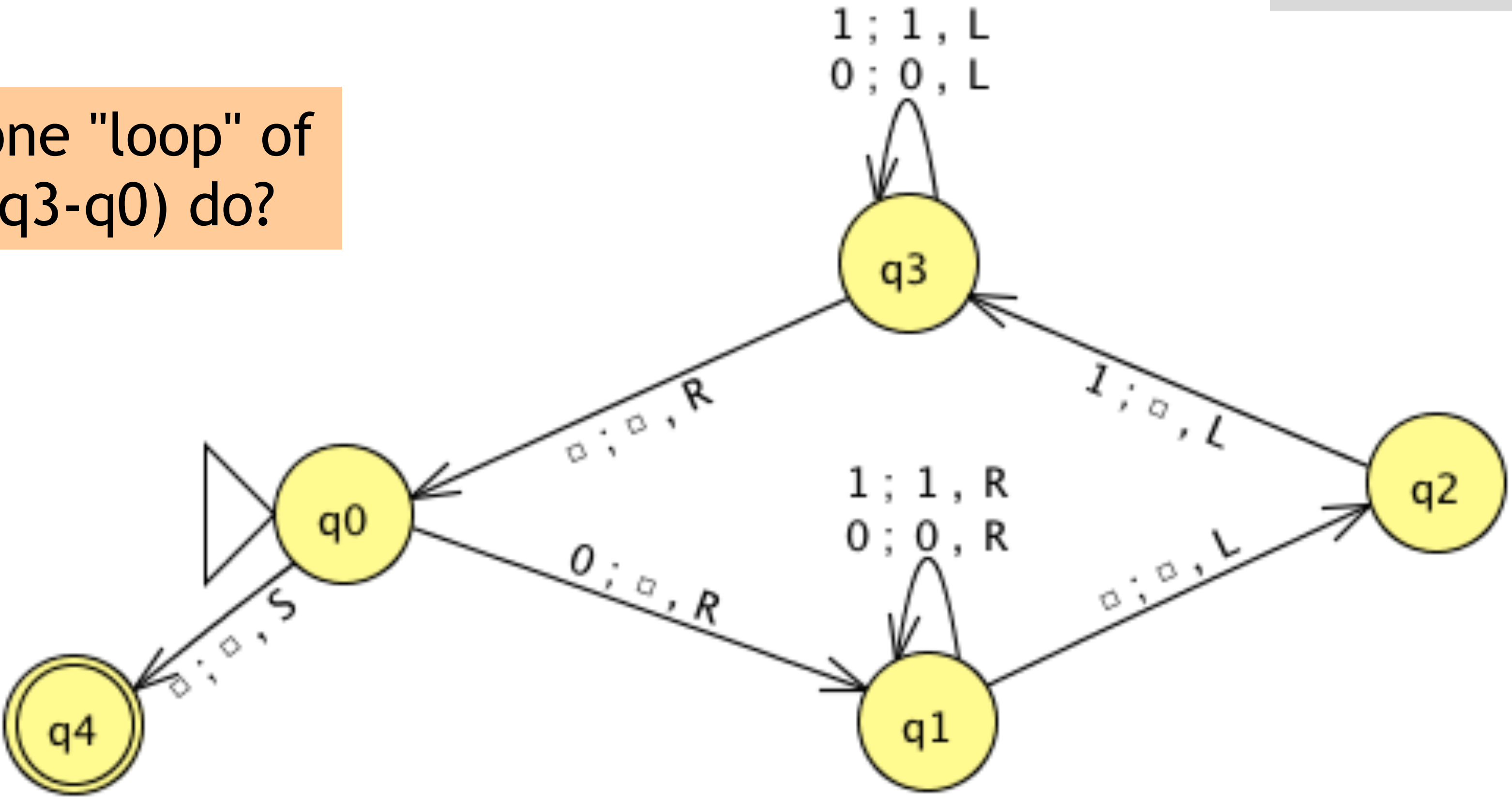
Reviews: 46 user | 108 critic | 31 from Metacritic.com



Is *this* input accepted or rejected by this TM?

Try it!

What does one "loop" of (q0-q1-q2-q3-q0) do?



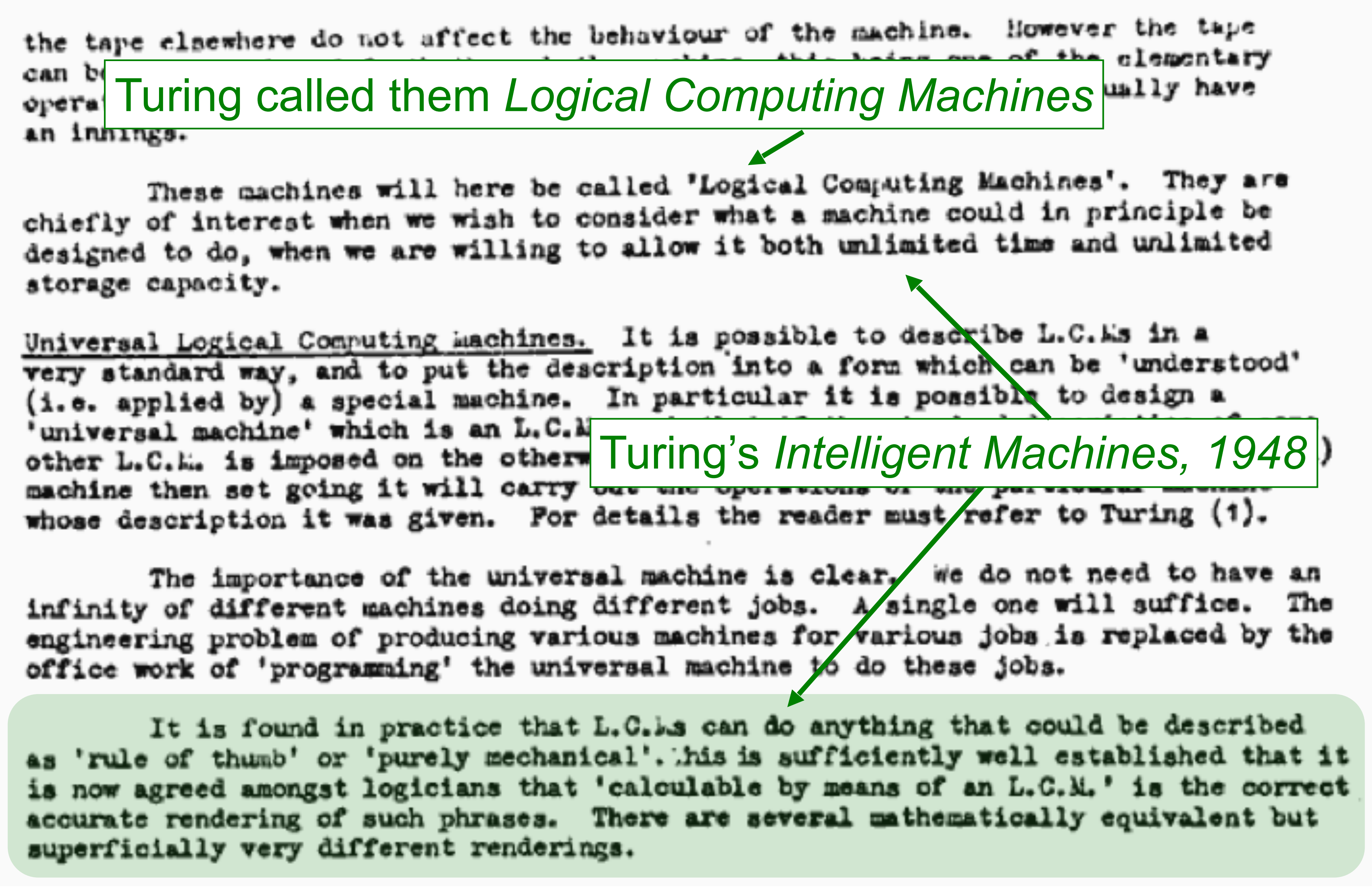
What inputs are accepted *in general*?

Extra: How could you change this TM to accept *palindromes?*

(thought experiment and ex. cr.)

Can TMs compute *everything*?

Alan Turing says **yes**...



the tape elsewhere do not affect the behaviour of the machine. However the tape
can b elementary
opera ually have
an innings.

These machines will here be called 'Logical Computing Machines'. They are chiefly of interest when we wish to consider what a machine could in principle be designed to do, when we are willing to allow it both unlimited time and unlimited storage capacity.

Universal Logical Computing Machines. It is possible to describe L.C.Ms in a very standard way, and to put the description into a form which can be 'understood' (i.e. applied by) a special machine. In particular it is possible to design a
'universal machine' which is an L.C.M
other L.C.M. is imposed on the otherw)
machine then set going it will carry
whose description it was given. For details the reader must refer to Turing (1).

The importance of the universal machine is clear. We do not need to have an infinity of different machines doing different jobs. A single one will suffice. The engineering problem of producing various machines for various jobs is replaced by the office work of 'programming' the universal machine to do these jobs.

It is found in practice that L.C.Ms can do anything that could be described as 'rule of thumb' or 'purely mechanical'. This is sufficiently well established that it is now agreed amongst logicians that 'calculable by means of an L.C.M.' is the correct accurate rendering of such phrases. There are several mathematically equivalent but superficially very different renderings.

Turing called them *Logical Computing Machines*

Turing's *Intelligent Machines, 1948*

http://www.alanturing.net/turing_archive/archive/l/l32/L32-005.html

So far, all known computational devices can compute only what Turing Machines can...

some are faster than others…

Quantum computation

http://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf

Molecular computation

http://www.arstechnica.com/reviews/2q00/dna/dna-1.html

Parallel computers

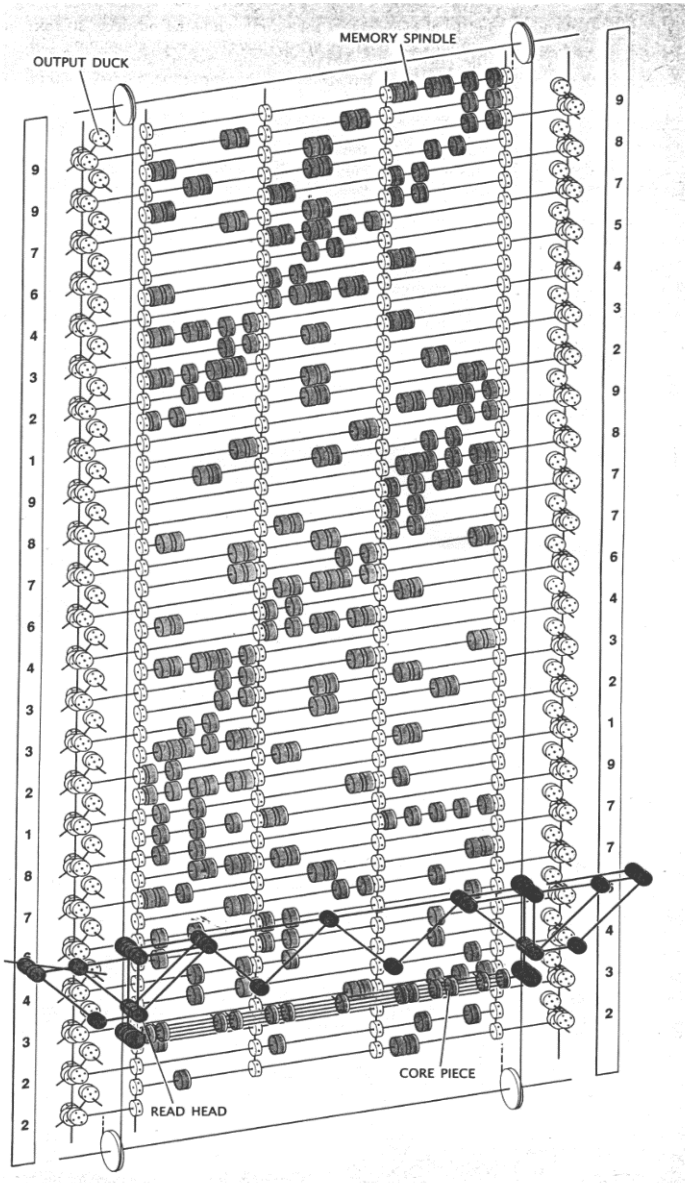
Integrated circuits

Electromechanical computation

Water-based computation

Tinkertoy computation

Turing machine

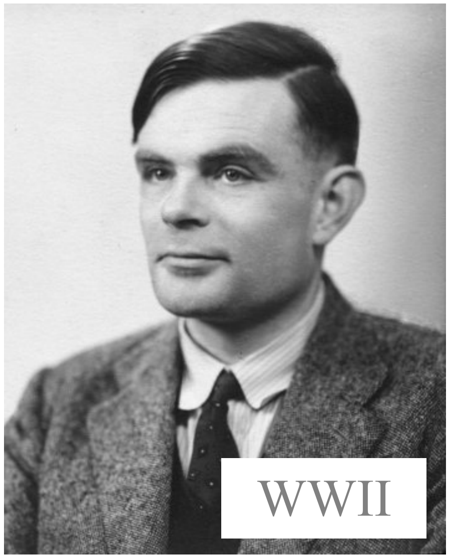


The Tinkertoy computer: ready for a game of tic-tac-toe

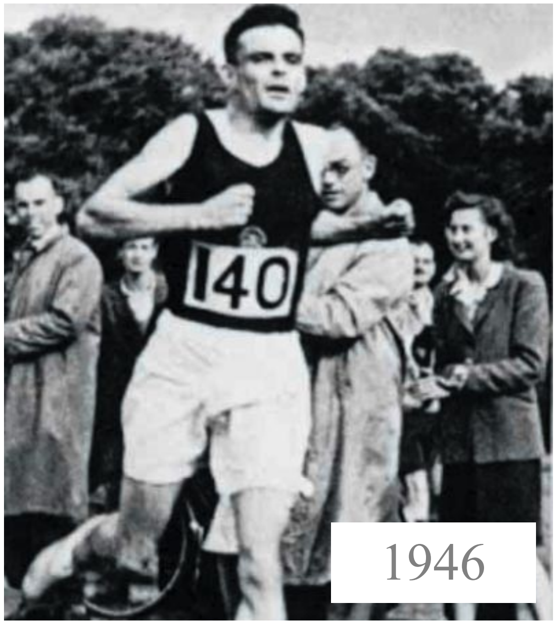
Alan Turing

1912-1954

AI!


WWII

Enigma machine ~ The axis's encryption engine


1946

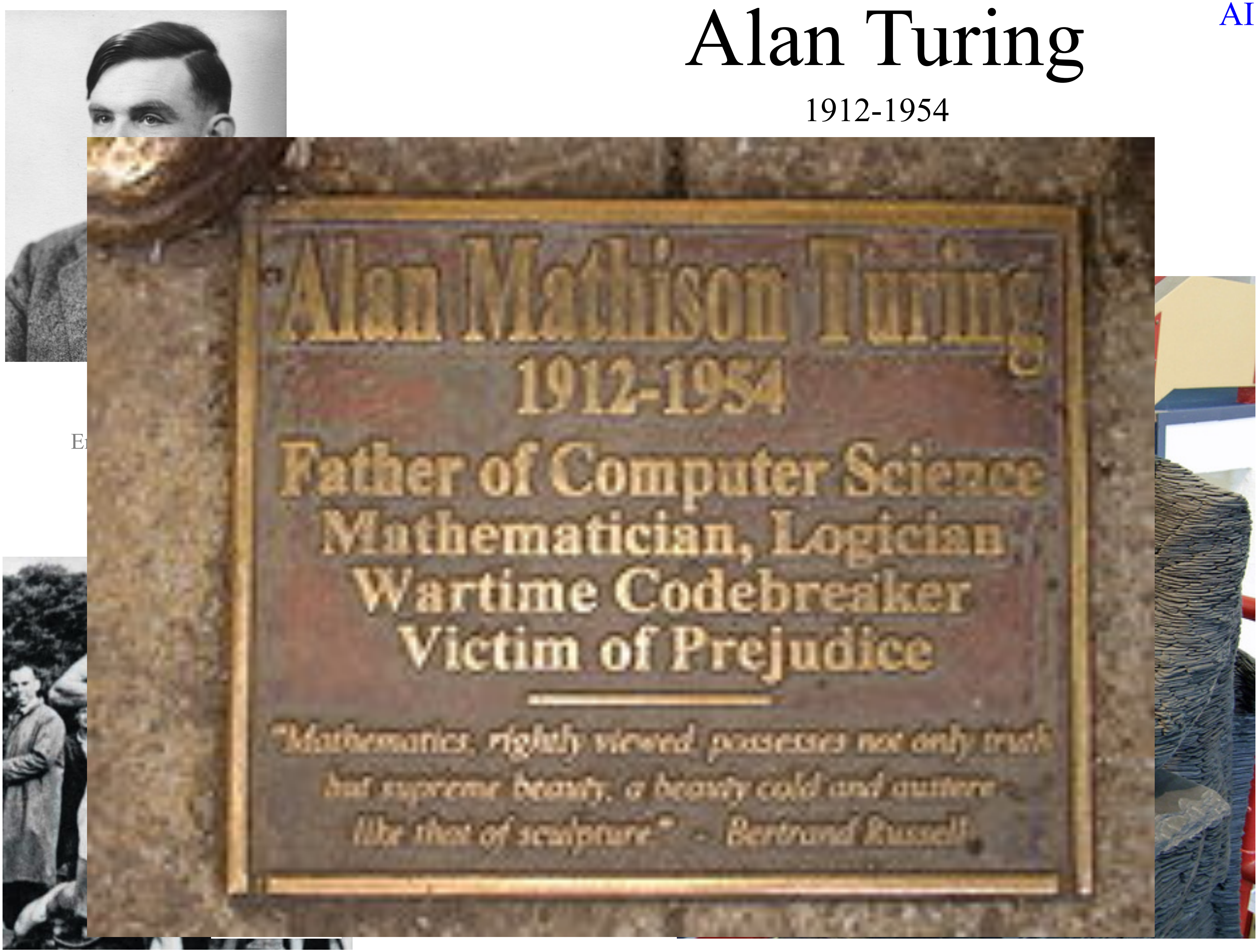

2007

Bletchley Park

Alan Turing

1912-1954

AI!



Alan Mathison Turing

1912-1954

Father of Computer Science

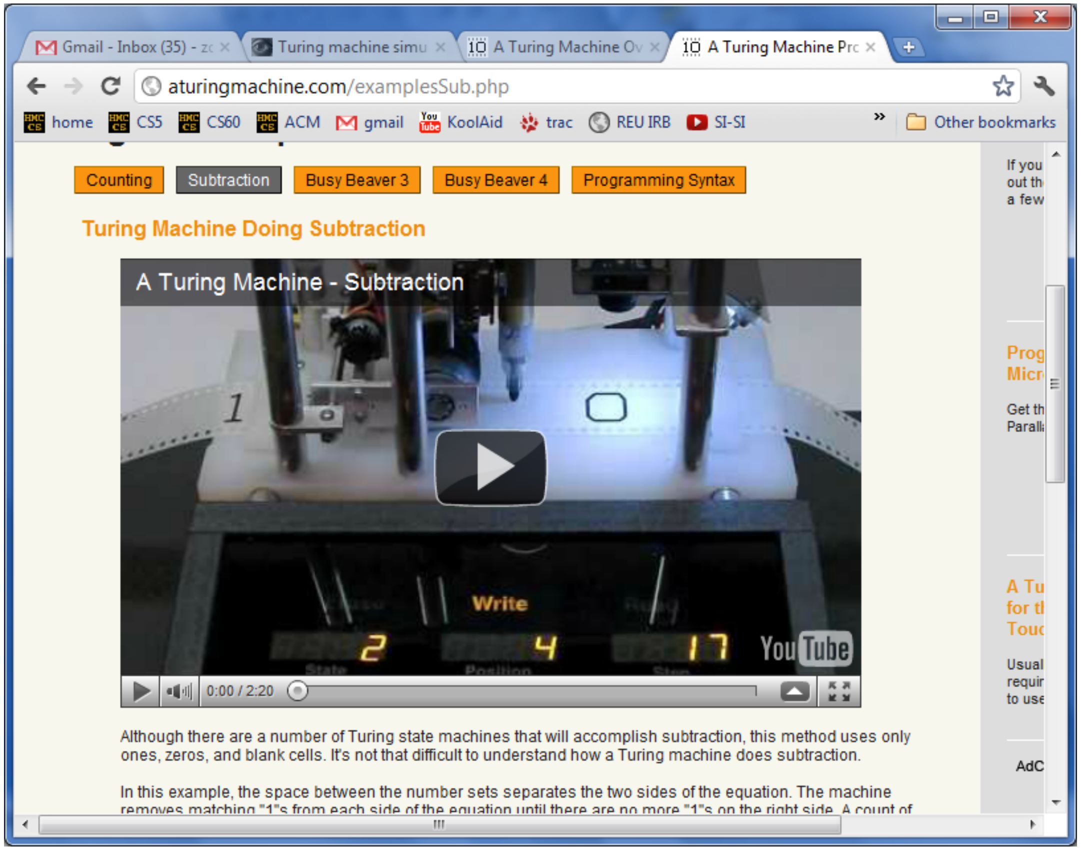
Mathematician, Logician

Wartime Codebreaker

Victim of Prejudice

"Mathematics, rightly viewed, possesses not only truth
but supreme beauty, a beauty cold and austere
like that of sculpture." - Bertrand Russell

2012: *Turing Celebration*

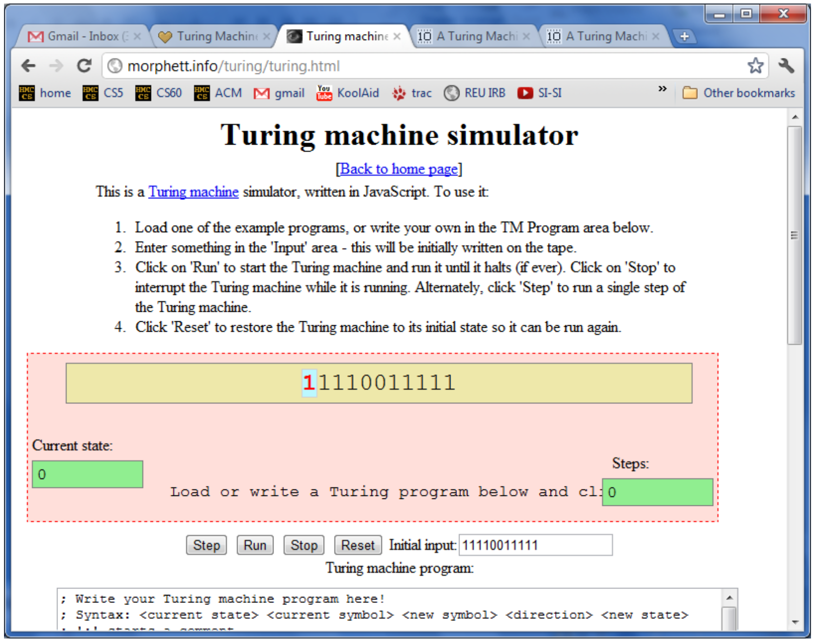


Turing Machine Doing Subtraction

Although there are a number of Turing state machines that will accomplish subtraction, this method uses only ones, zeros, and blank cells. It's not that difficult to understand how a Turing machine does subtraction.

In this example, the space between the number sets separates the two sides of the equation. The machine removes matching "1"s from each side of the equation until there are no more "1"s on the right side. A count of

http://aturingmachine.com/examplesSub.php



Turing machine simulator

[Back to home page]

This is a Turing machine simulator, written in JavaScript. To use it:

1. Load one of the example programs, or write your own in the TM Program area below.
2. Enter something in the 'Input' area - this will be initially written on the tape.
3. Click on 'Run' to start the Turing machine and run it until it halts (if ever). Click on 'Stop' to interrupt the Turing machine while it is running. Alternately, click 'Step' to run a single step of the Turing machine.
4. Click 'Reset' to restore the Turing machine to its initial state so it can be run again.

11110011111

Current state: 0

Load or write a Turing program below and cl

Steps: 0

Initial input: 11110011111

Turing machine program:

```
; Write your Turing machine program here!
; Syntax: <current state> <current symbol> <new symbol> <direction> <new state>
```

https://www.youtube.com/watch?v=aBToqFJLrl4

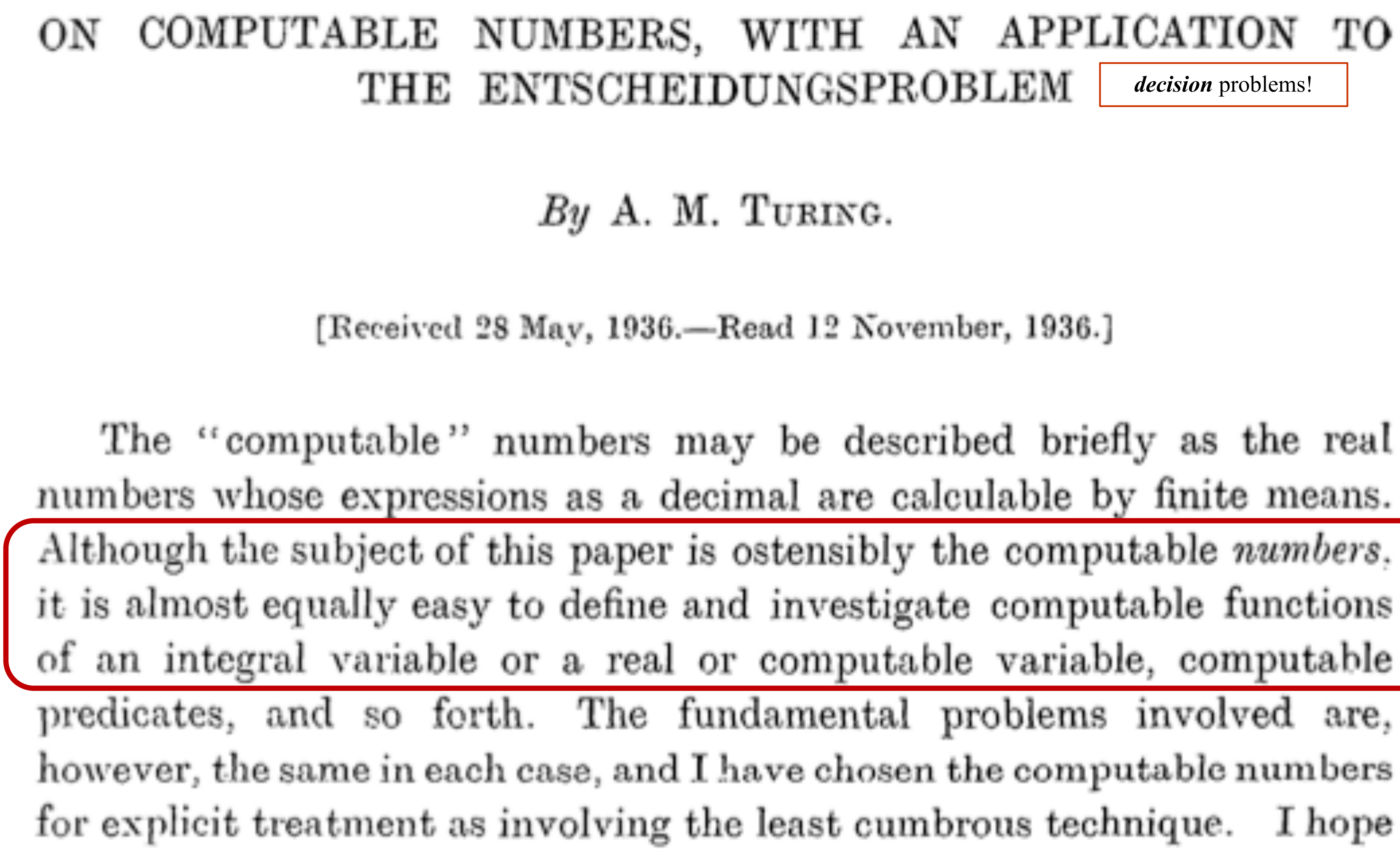
Stretch Break!

TURING TEST

PERMANENT LINK TO THIS COMIC: HTTPS://XKCD.COM/329/

Can ~~TMs~~ computers compute everything?

Alan Turing says **No!**



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

decision problems!

By A. M. TURING.

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

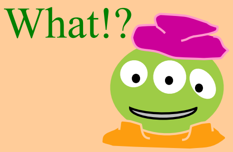
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable 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

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

There are many problems computers can't solve at all !

Perhaps this is not that surprising…

- rising sea levels
- disbelief in aliens
- losing to your own Connect4 (at 0 ply!)
- towel folding! (well, ***fast*** towel folding…)



Unprogrammable *functions?*

There are

well-defined
mathematical
functions

that no

computer
program

or TM

can compute

even with ***any***
amount of memory!

functions

`programs`

$$f(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

```
def prog1(x):
   return x%2
```

Unprogrammable *functions?*

There are <u>many</u> more of these …

mathematical functions

than these!

programs

There ar hat no

co pr

memory!

functions

programs

$$f(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

```
def prog1(x):
  return x%2
```

functions **int - bool**

$$f_A(x) = 1$$

$$f_B(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

$$f_C(x) = \begin{cases} 1 & \text{if x is 0, 1, or 2} \\ 0 & \text{otherwise} \end{cases}$$

These are ***"int-bool"*** mathematical functions.

They're also sometimes called *"integer predicates."*

- Input is an integer, x >= 0
- Output is 0/1 (*boolean or bit*)

...even if you restrict your functions to be totally discrete!

int - bool `programs`

Example programs

- Input is one integer, **`x >= 0`**
- Output is **0**/**1** (*boolean or bit*)

If programs <u>look different</u> they are different – *even if they compute the same function!*

```
def prog1(x):
   return x%2
```

all int inputs:
`x >= 0`

```
def prog2(x):
   return x<3
```

```
def prog3(x):
   return 1
```

```
def prog4(x):
   return len(str(x+42))>1
```

... and allow ANY programs at all ~ even syntax errors

Let's match!

functions

$$f_A(x) = 1$$

$$f_B(x) = \begin{cases} 1 & \text{if x is odd} \\ 0 & \text{if x is even} \end{cases}$$

$$f_C(x) = \begin{cases} 1 & \text{if x is 0, 1, or 2} \\ 0 & \text{otherwise} \end{cases}$$

1. Match each program with the function it computes.
2. There are three different functions on the left side -- how many *different programs* are in the right side?

*How - or why - is the set of **all** **functions** larger than the set of **all** **programs** ?*

programs

```
def prog1(x):
  return x%2
```

all int inputs:
`x >= 0`

```
def prog2(x):
  return x<3
```

```
def prog3(x):
  return 1
```

```
def prog4(x):
  return len(str(x+42))>1
```

```
def prog5(x):
  return x in [0,1,2]
```

```
def prog6(x):
   if x<2: return x
   else: return prog6(x-2)
```

Worksheet!

functions

programs

$f_A(x) =$

$f_B(x) =$

$$f_C(x) = \begin{cases} 1 & \text{if x is 0, 1, or 2} \\ 0 & \text{otherwise} \end{cases}$$

There are *many* more of these …

mathematical functions

```
def p
   ret
```

```
def p
   retu
```

```
def prog3(x):
   return 1
```

```
def prog4(x):
   return len(str(x+42))>1
```

```
def prog5(x):
   return x in [0,1,2]
```

```
def prog6(x):
   if x<2: return x
   else: return prog6(x-2)
```

than these!

programs

1. Match each program with the function it computes.
2. There are three different functions on the left side -- how many *different programs* are in the right side?

*How - or why - is the set of **all functions** larger than the set of **all programs** ?*

Quiz! Name(s): ______________________

Uncomputable *functions*?

There are

well-defined mathematical functions

that no

computer program

or TM™

can compute

*even with **any** amount of memory!*

Why?

There are many more of these …

mathematical functions

than these!

programs

$\mathbb{R}$ ***vs.*** $\mathbb{N}$

Programs are "like" integers…

```
def alien( x ):
  if x == 42:
    return True
  else:
    return not \
alien(x+1)
```

`programs`

For each program, there is an integer.

For each integer, there is a "program."

This is true:

but how?

at least a string!

some have syntax errors...

Programs are integers (and vice-versa)

```
def alien( x ):
  if x == 42:
    return True
  else:
    return not \
alien(x+1)
```

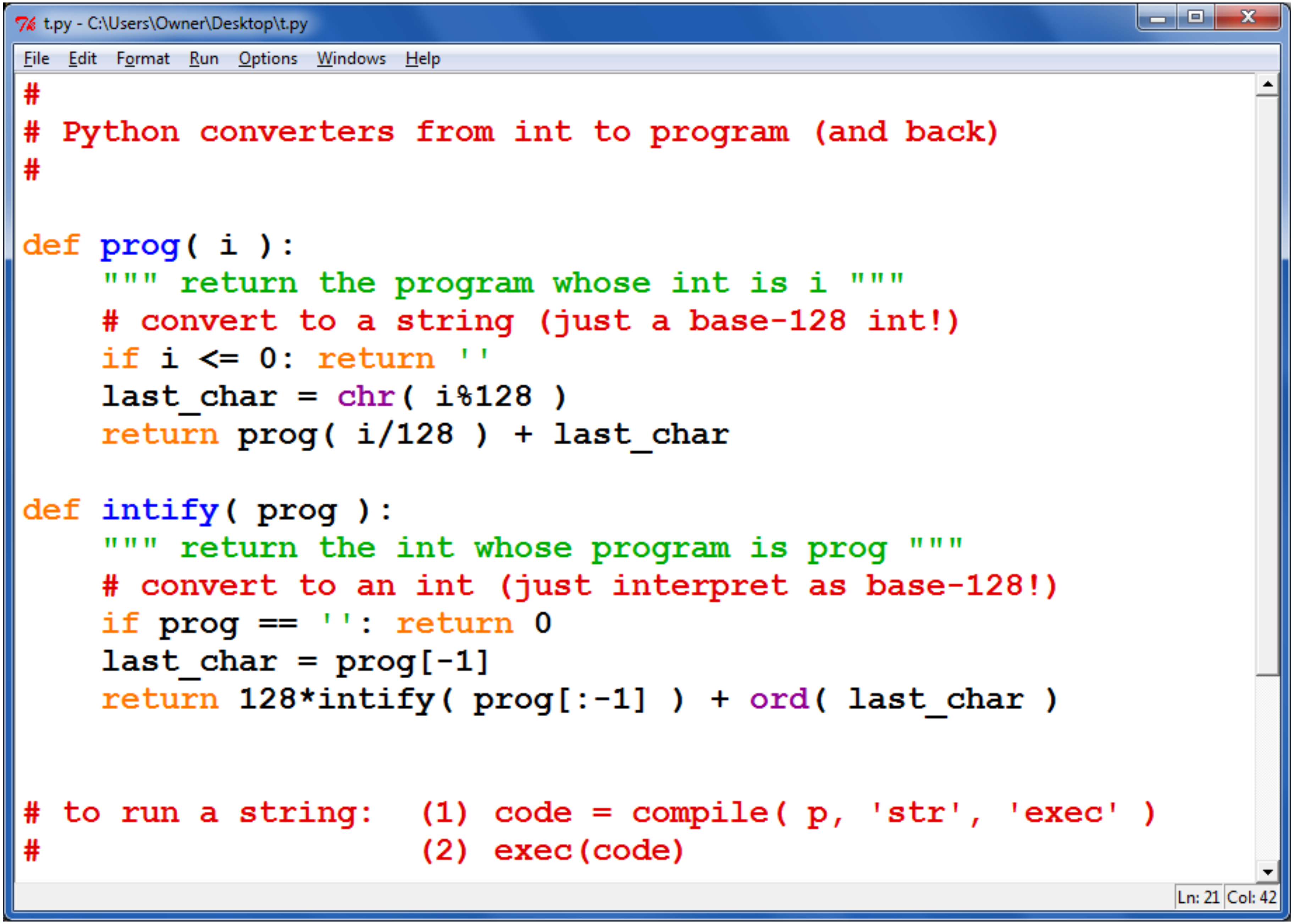
`programs` = ℕ Positive integers

Every program is a **string**.

Every string is just a sequence of **bits**

Every sequence of bits is also an **int**!

from **prog**s to **int**s ~ and *back*...



```
#
# Python converters from int to program (and back)
#

def prog( i ):
    """ return the program whose int is i """
    # convert to a string (just a base-128 int!)
    if i <= 0: return ''
    last_char = chr( i%128 )
    return prog( i/128 ) + last_char

def intify( prog ):
    """ return the int whose program is prog """
    # convert to an int (just interpret as base-128!)
    if prog == '': return 0
    last_char = prog[-1]
    return 128*intify( prog[:-1] ) + ord( last_char )


# to run a string:   (1) code = compile( p, 'str', 'exec' )
#                    (2) exec(code)
```