

CS3102 Theory of Computation

www.cs.virginia.edu/~njb2b/cstheory/s2020

Warm up:

Is there anything a human can compute that
no “mechanical” computer ever could?
Why or why not?

Humans can compute, but machines can't

- Subjective thing:
 - Is this delicious?
- Perhaps nothing (e.g. if I could perfectly represent a human brain as a program)
- Meaning of life
 - Can humans even do this
 - 42
- Quality of art
 - Quality art looks realistic
 - Quality of art uses lots of color
- Machines can only do what we could write an algorithm for.
- You could maybe make a machine mimic an individual's preferences

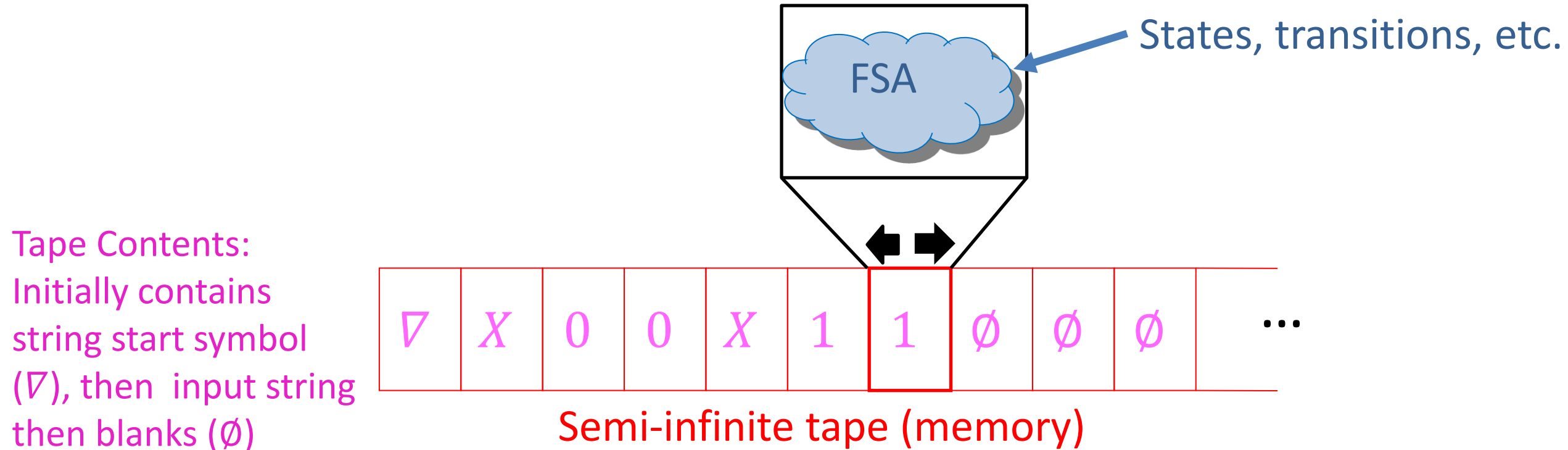
Alan Turing's contribution

- What we did this semester:
 - We have a vague understanding of what a computer is and what it does
 - We started with our idea of what a computer is, and then tried to formalize that
- What did Alan Turing do?
 - We had examples of computers: Babbage machines, electrical computers (Bombe), etc.
 - He started with the most impressive computer he could think of as the “baseline” for his definition
 - Humans

Turing Machines

- FSA:
 - Finite number of states,
 - read-once input,
 - transition using input bit and state(s)
- Turing Machine:
 - Finite number of states,
 - ~~– Read-once input,~~
 - Semi-infinite tape (memory)
 - Transition using “current symbol” on tape
 - Can overwrite current symbol, move left/right on tape

Turing Machine



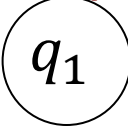
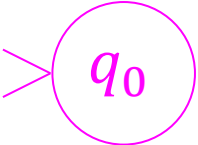
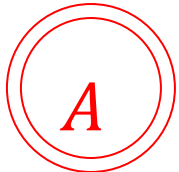
Operation:

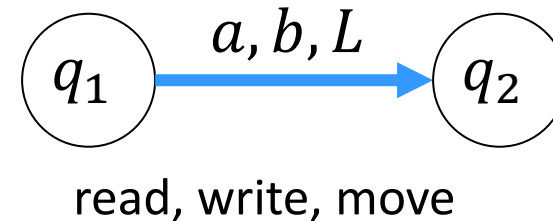
transitions outgoing from each state match on current character on the tape, when transitioning you can overwrite that character and move which cell you're reading

Return 1 if we enter the accept state, Return 0 if we enter the reject state

Turing Machine

Basic idea: a **Turing Machine** is a finite state automaton that can optionally read from/write to an infinite **tape**.

- Finite set of states: $Q = \{q_0, q_1, q_2, \dots, q_k\}$ 
- Input alphabet: Σ
- **Tape** alphabet (includes \emptyset, ∇): Γ
- **Transition** function: $\delta: Q \times \overset{\text{Read}}{\Gamma} \rightarrow Q \times \overset{\text{Write}}{\Gamma} \times \overset{\text{Move}}{\{L, R, S, H\}}$
- **Initial** state: $q_0 \in Q$ 
- **Final** states: $F \in Q$ 



Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Turing Machine Execution

- Start in the initial state with the “read head” at the start of the tape. The input string follows that, then infinitely many blanks
- Look at the current state and character under the “read head”, then transition to a new state, overwrite that character, and move the “read head”
- Continue until the movement instruction is “halt”
- Output: two models:
 - Decision problems: if you’re in a final state then return 1, else return 0
 - Function: All the contents remaining on the tape (except for ∇ and \emptyset)

What does a given machine compute?

- Say the output (string left on the tape) of Turing Machine M on input $x \in \{0,1\}^*$ is $M(x) \in \{0,1\}^*$
 - That mapping $M(x)$ is the function of that Turing machine
- Say the output of Turing Machine M on input $x \in \{0,1\}^*$ is $M(x) \in \{0,1\}$
 - The language of machine M , denoted as $L(M)$, is the set of all strings for which $M(x) = 1$

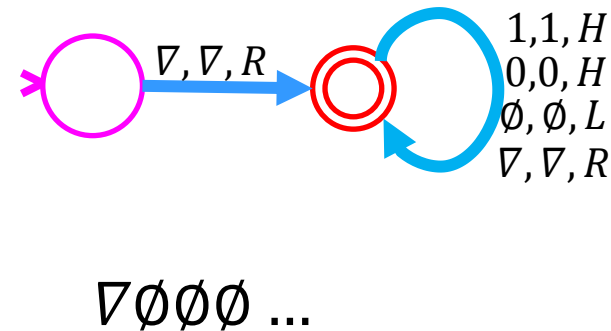
FSA vs TM

- Returns 1 when
 - FSA: you read the last input character and end up in a final state
 - TM: you take a halt transition to a final state
- Returns 0 when
 - FSA: you read the last input character and end up in a non-final state
 - TM: you take a halt transition to a non-final state
- FSA: you'll always eventually read the last character, so they always return something
- TM: You can build a machine that, for some inputs, never takes a halt transition!

Some Turing Machines never return

- In this case they run forever
- 3 behaviors
 - Return 1
 - Return 0
 - Run forever
- This is necessary for computation

```
while(true){  
    twiddle(thumbs);  
}
```



```
while(x != 1){  
    if(x%2 == 0){  
        x = x / 2;  
    }  
    else{  
        x = 3x+1;  
    }  
}
```

Running forever

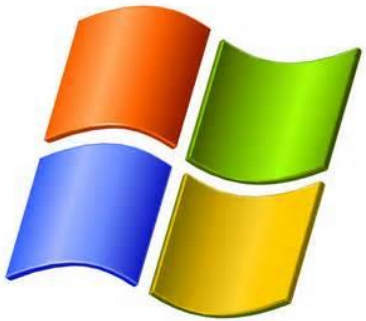
- Is it a bad thing?

Programs we want to halt:



Your 2150
Homework

Programs we want to run forever:



What is Computable?

- Definition:
 - A function/language is computable provided there is some **always-halting Turing machine** for it
 - Function: computable provided there is an **always-halting Turing machine** which, when run on a tape containing only the input, always halts with only the corresponding output on the tape
 - Language: computable provided there is an **always-halting Turing machine** which, when run on a tape containing only the input, always halts and returns 1 if that string was in the language, and 0 otherwise
- Assertion:
 - This definition is the most powerful definition of computability that is physically possible
 - Why...?

Church-Turing Thesis



Alonzo Church, 1903-1995

A Turing Machine (or
Lambda Calculus)
can simulate *any*
“mechanical
computer”.



Alan Turing, 1912-1954

Thesis?

- Axiom:
 - Something that is not proven, but is so obvious that it doesn't need to be
 - Justifiably assumed to be true
- Theorem:
 - A statement that has been proven by a sequence of axioms
- Thesis:
 - A philosophical statement, that is justified with an intuitive (yet compelling) argument
 - A statement that is too open-ended to be approached by formal mathematics

COMPUTING
DIVISION
COMPUTING
SECTION

Bonus Bureau, Computing Division, 11/24/1924



<https://www.loc.gov/pictures/item/201>

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

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also

Why is the *alphabet* finite?

will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent†. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as

17 or 9999999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of symbols). The differences from our point of view between the single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 99999999999999999 and 99999999999999999 are the same.

Why is the *alphabet* finite?

will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, *i.e.* on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we

† If we regard a symbol as literally printed on a square we may suppose that the square is $0 \leq x \leq 1, 0 \leq y \leq 1$. The symbol is defined as a set of points in this square, viz. the set occupied by printer's ink. If these sets are restricted to be measurable, we can define the "distance" between two symbols as the cost of transforming one symbol into the other if the cost of moving unit area of printer's ink unit distance is unity, and there is an infinite supply of ink at $x = 2, y = 0$. With this topology the symbols form a conditionally compact space.

is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 999999999999999999 and 999999999999999999 are the same.

Why is the *alphabet* finite?

- Recognizing/distinguishing symbols should be “quick”, it shouldn’t require its own “computation”

Why is the *number of states* finite?

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his “state of mind” at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be “arbitrarily close” and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

Why is the *number of states* finite?

- The states in the machine represent the “state of mind” of a human.
- Humans should only be able to have a finite number of “states of mind”
 - Your “state of mind” must be in your brain, and your brain has finite volume/mass/matter/stuff

Turing machine

- Before: Human with “states of mind”, typing on a typewriter with finitely many “keys”, working on a piece of paper
- Model: finitely many states, finitely many characters in an alphabet, linear tape for memory, the human can change the tape, the tape can influence the human’s state of mind

The “computable” numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means.

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

What can a Turing Machine compute?

- For sure:
 - Any Java/Python program
- If the Church-Turing Thesis is Correct:
 - Anything that a human can compute
- Some evidence that it might be correct:
 - [Simulating a nematode](#)