CS3102 Theory of Computation

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

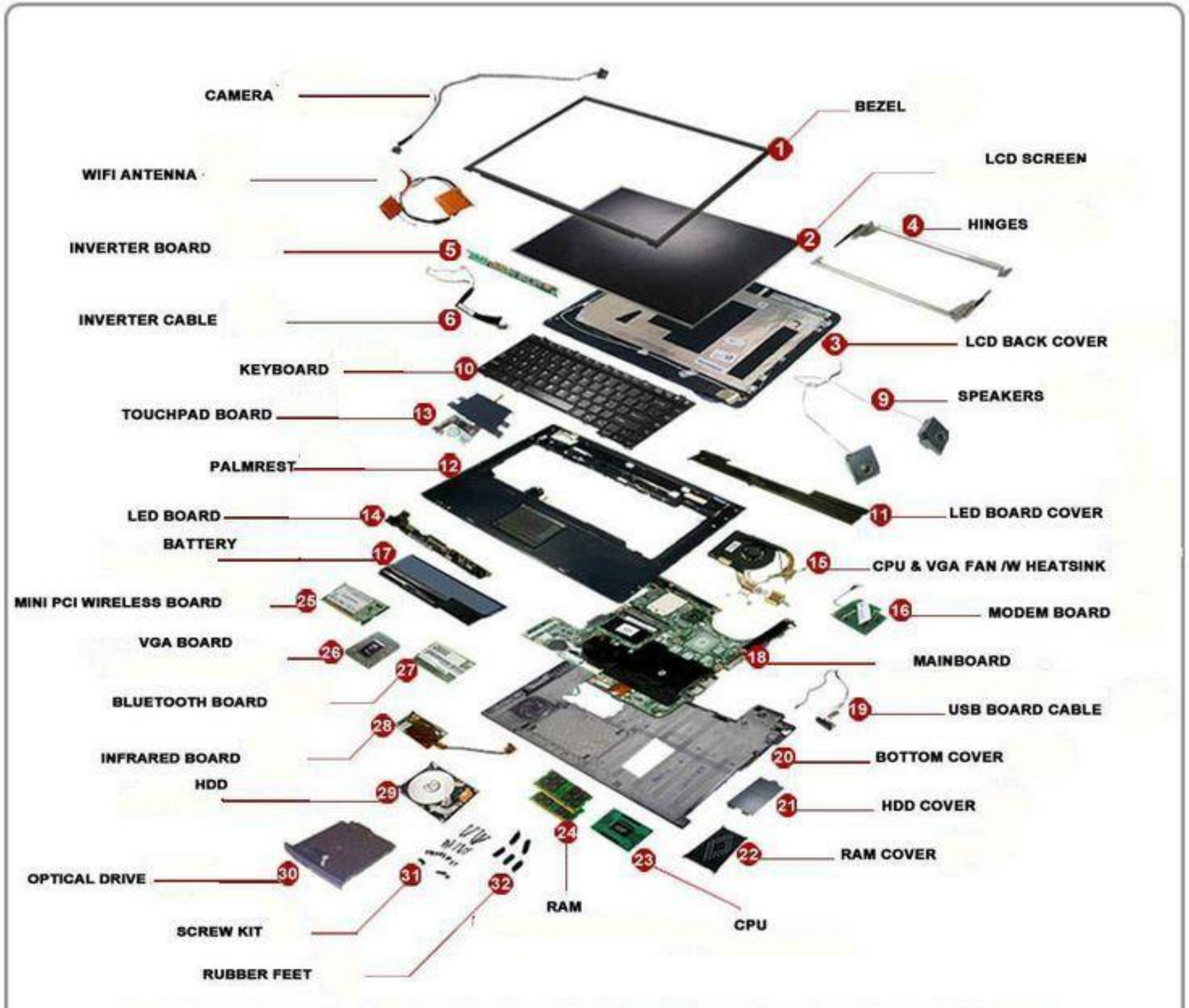
Warm up:

How are you?

Going Online Logistics

- Lecture
  - Important Zoom features:
    - Go faster
    - Go slower
    - Raise hand
    - Yes/no
- Office Hours
  - Office hours queue
  - Services
- Exams

“Dissecting” a Computer

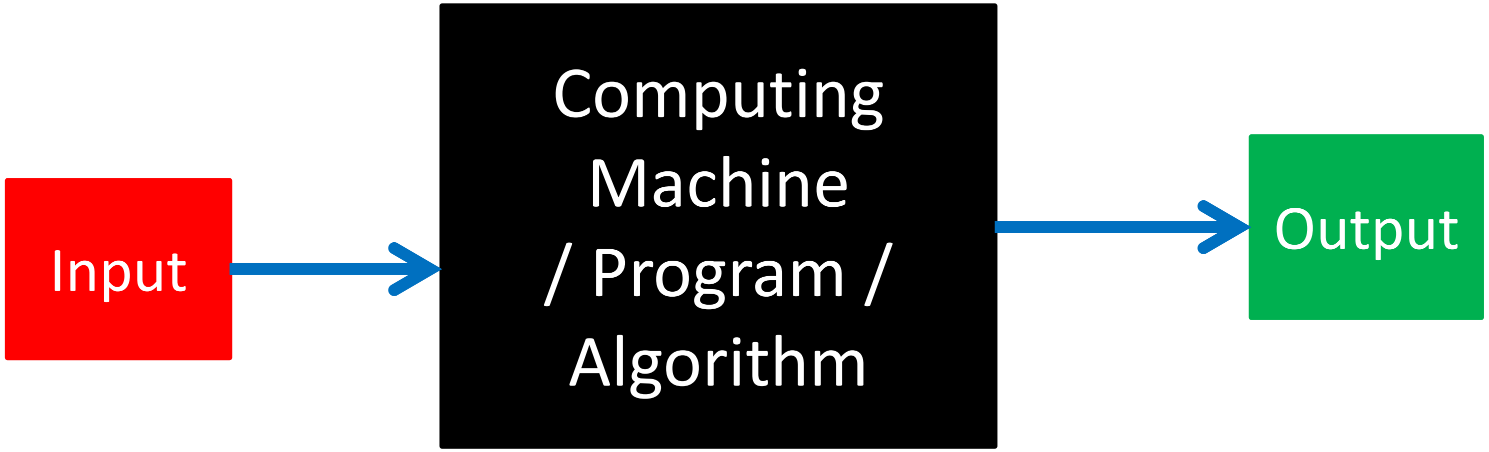


Most important parts (according to Nate)

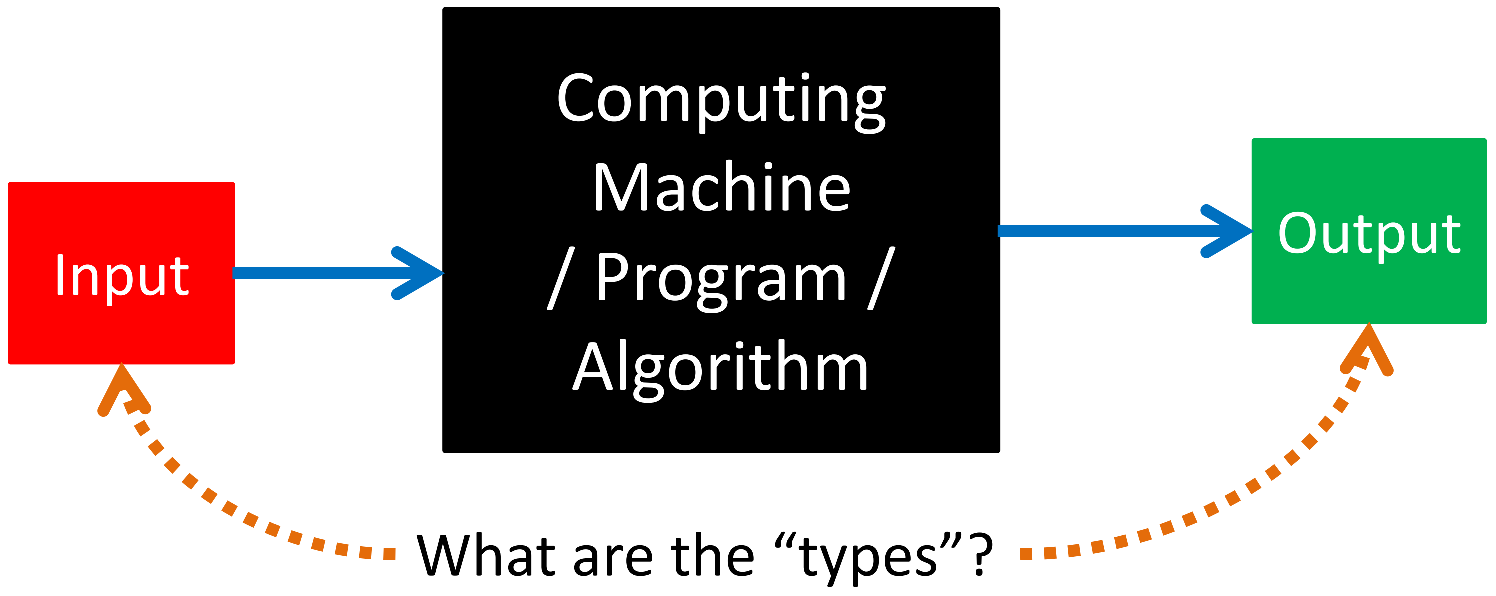
- CPU
  - Circuits of transistors
- RAM
  - Limited memory
- HDD/SSD
  - Large memory

What does it mean to compute?

- We’ll discuss several ideas this semester
- Several “models” of computing
- Vague idea: take and input and produce an output



Defining Our Input/Output



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

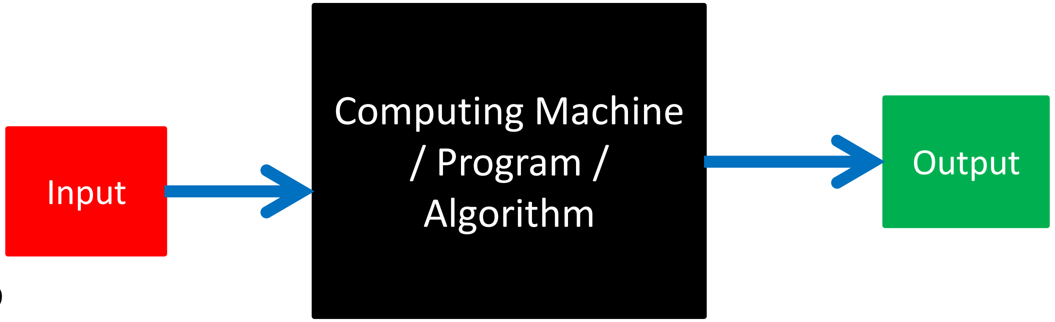
What we compute on: representations of things (e.g. numbers)

What do we compute on?

- **String**: an ordered sequence of characters
- Is a representation of something
- Characters come from an alphabet

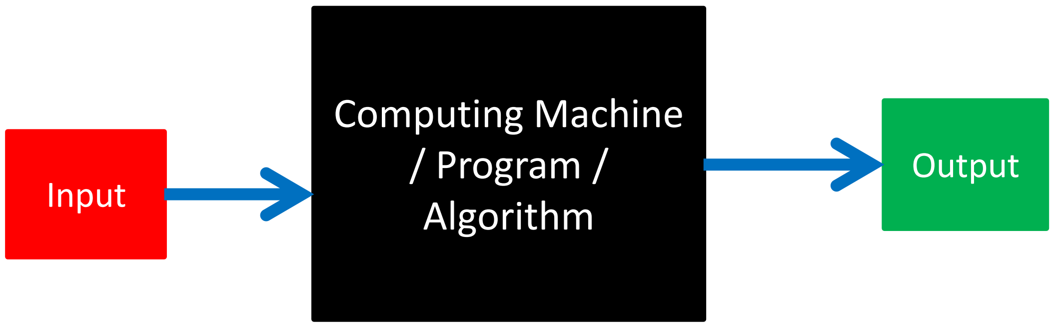
- Let's formally define them

What do we compute, then?



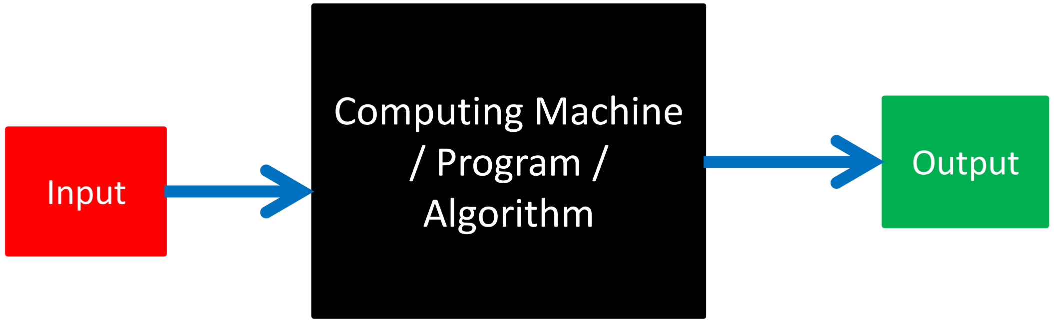
- Input and output are strings
- Black box is an implementation
- What are we implementing?
  - Functions
  - Languages

Computing a Function



- A function f is computable under a computing model if:
- That model allows for an implementation (way of filling in the black box) such that,
  - For any input $x \in D$ (string representing an element from the domain of f)
  - The implementation "produces" the correct output

Computing a Language



- A Language L is computable under a computing model if:
- That model allows for an implementation (way of filling in the black box) such that,
  - For any input $x \in \Sigma^*$
  - The implementation returns 1 if and only if $x \in L$

Function vs Decision vs Language

Name	Decision Problem	Function	Language
XOR	Are there an odd number of 1's?	$f(b) = \begin{cases} 0 & \text{number of 1s is even} \\ 1 & \text{number of 1s is } odd \end{cases}$	$\{b \in \Sigma^* \| b \text{ has and even number of 1s}\}$
Majority	Are there more 1s than 0s?	$f(b) = \begin{cases} 0 \text{ more 0s than 1s} \\ 1 \text{ more 1s than 0s} \end{cases}$	$\{b \in \Sigma^* \| b \text{ has more 1s than 0s}\}$

$|\{0,1\}^\infty| > |\mathbb{N}|$

- Idea:
  - show there is no way to "list" all finited binary strings
  - Any list of binary strings we could ever try will be leaving out elements of $\{0,1\}^\infty$

Differences

Hardware (CPU)

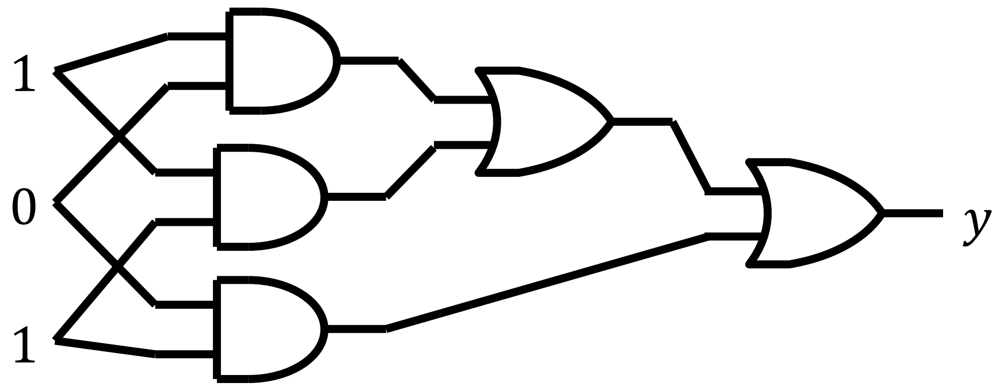
- Concrete
- Fixed
- Simpler (each unit of computation does “less”)
  - Computation has smaller steps
- Doesn’t ever need to be software
- Everything is always doing physics

Software (Java)

- “idealized”, “abstract”
- Reconfigurable
- Transportable
- Each “step” is bigger
- Needs to “become” hardware
- Needs to be translated
- Sequential (limited parallel)

Defining the AON circuit model

- Define how to represent a computation
  - And/Or/Not circuit:
    - Number of inputs
    - Number of outputs
    - Gates and their labels
    - Wires connecting the above



- Define how to perform an execution
  - For each component, find its value once all its inputs are defined
  - Inputs start of with their value defined
  - Things labelled as output are the result

A circuit-like programming language

- Define how to represent a computation
  - Inputs as positional arguments
  - Outputs as return statements
  - Variable assignments using boolean operators AND/OR/NOT
- Define how to perform an execution
  - Evaluate each variable assignment sequentially

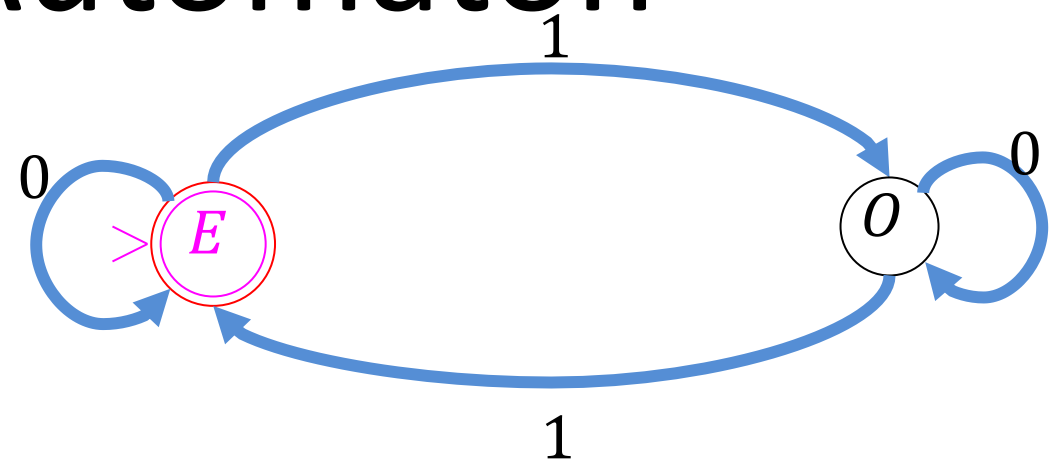
AON-Straightline

```
def MAJ(a, b, c):
    first_two = AND(a,b)
    last_two = AND(b,c)
    first_last = AND(a,c)
    temp = OR(first_two, first_last)
    return OR(temp, last_two)
```

Issues and Solutions

- What were the limitations of circuits?
  – No loops: meaning only finite functions
  – Fixed input sizes
- How can we overcome those?
  – Finite state automata (the execution definition allowed for infinite)
  – Iterated: do some work, update “state”, do more work, until no more input

Finite State Automaton



- Implementation:
  - Finite number of states
  - One start state
  - "Final" states
  - Transitions (function mapping state-character pairs to states)
- Execution:
  - Start in the initial "state"
  - Read each character once, in order (no looking back)
  - Transition to a new state once per character (based on current state and character)
  - Give output depending on which state you end in

"Pieces" of a Regex

- Empty String:
  - Matches just the string of length 0
  - Notation: ε or ""
- Literal Character
  - Matches a specific string of length 1
  - Example: the regex a will match just the string a
- Alternation/Union
  - Matches strings that match at least one of the two parts
  - Example: the regex $a|b$ will match a and b
- Concatenation
  - Matches strings that can be dividing into 2 parts to match the things concatenated
  - Example: the regex $(a|b)c$ will match the strings ac and bc
- Kleene Star
  - Matches strings that are 0 or more copies of the thing starred
  - Example: $(a|b)c^*$ will match a, b, or either followed by any number of c's

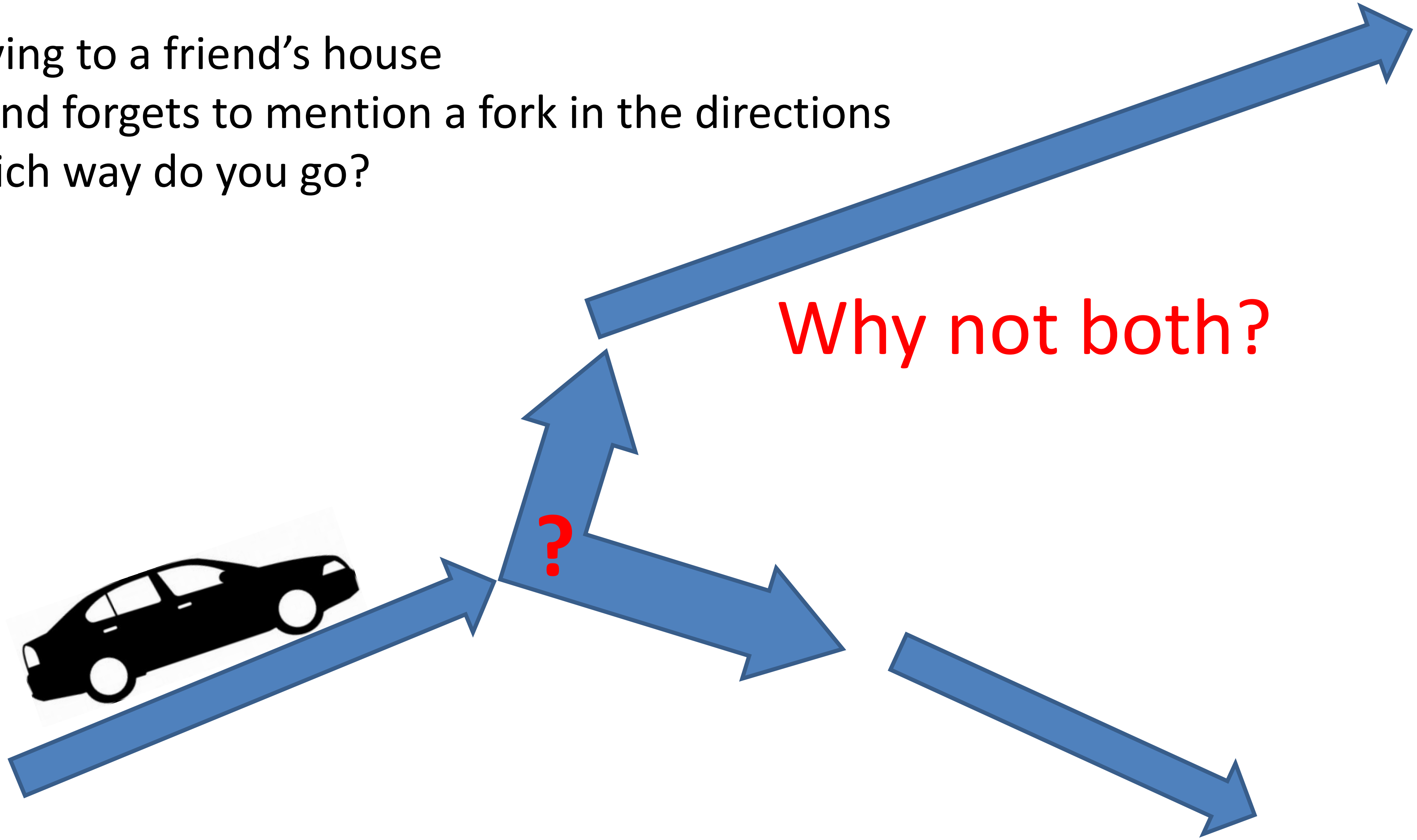
Note: The compents here are the minimal necessary. In practice, regexes have other components as well, those are just "syntactic sugar".

Nondeterminism

Driving to a friend’s house

Friend forgets to mention a fork in the directions

Which way do you go?



Issues and Solutions

- What were the limitations of circuits?
  - ~~Actually infinite inputs (not relevant to us)~~
  - No looking back!
  - Change the machine mid-process
  - Limited storage, bigger inputs require more memory for some functions
  - Larger output space (only 0 or 1)
  - Non-determinism: no communication among parallel paths
    - Outside the scope of this semester
    - Alternation
- How can we overcome those?
  - You can look backwards!
  - Lots of / Plentiful / enough memory: infinite!
  - Make machines that can play the roll of another machine, compute machines (macros)
  - Execution model that allows for long strings to be outputs

Characterizing What’s computable

- Things that are computable by FSA:
  - Functions that don’t need “memory”
  - Languages expressible as Regular Expressions
- Things that aren’t computable by FSA:
  - Things that require more than finitely many states
  - Intuitive example: Majority

Majority with FSA?

- Consider an inputs with lots of 0s

000...0000	111...1111	000...0000	111...1111	000...0000	111...1111
×49,999	×50,000	×50,000	×50,000	×50,000	×50,001

- Recall: we read 1 bit at a time, no going back!
- To count to 50,000, we'll need 50,000 states!