Exams

- Why do exams exist?
  - Proctored, solo, and timed evaluation of your knowledge
  - Everyone takes the exam on their own, and limits resources for themselves
    - Doesn’t work well
  - Everyone takes the exam during the normal class time
    - What happens if you have internet connectivity issues?
    - What will be the format of the exam so that you can take it, and have a readable digital version to submit?
    - Can I make the exam take longer than 75 minutes?
- What does the sudden move to online do to a class/students/professors/etc?
  - It stresses everybody out
  - Best I could hope for with the online format is that ya’ll learn stuff
  - Exams are super stressful and they are “forcing factor” for learning rather than an exercise in learning
- What I’m going to do:
  - I’m going to assume that you want to learn the stuff
  - Trust that exercises and quizzes are sufficient opportunity for you to learn the stuff
  - Exam 2 and the final exam will be optional for all students

CS3102 Theory of Computation

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

Warm up:

What did “Universality” mean in the context of Circuits?

What might “Universality” mean in the context of Turing Machines?

Turing Universality

- Your thoughts:
  - It solves all infininite functions
    - Fewer implementations than infinite functions
    - Countable number of implementations, but uncountably many infinite functions
    - Not possible
  - It can compute anything that’s computable
    - Some infinite functions are computable
    - We used Turing machines to define computability
      - A thing is computable if an always-halting Turing machine can implement it

Last Time

- Church-Turing Thesis
  - Why are Turing Machines the “goto” model of computing?

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

Today

- What can't Turing Machines do?

Circuit Universality

- A set of gates is universal if they can be used to compute any finite function
- Conquence: A circuit to evaluate other Circuits:

Defining EVAL

$EVAL_{s,n,m}: \{0,1\}^{S(s)+n} \rightarrow \{0,1\}^m$

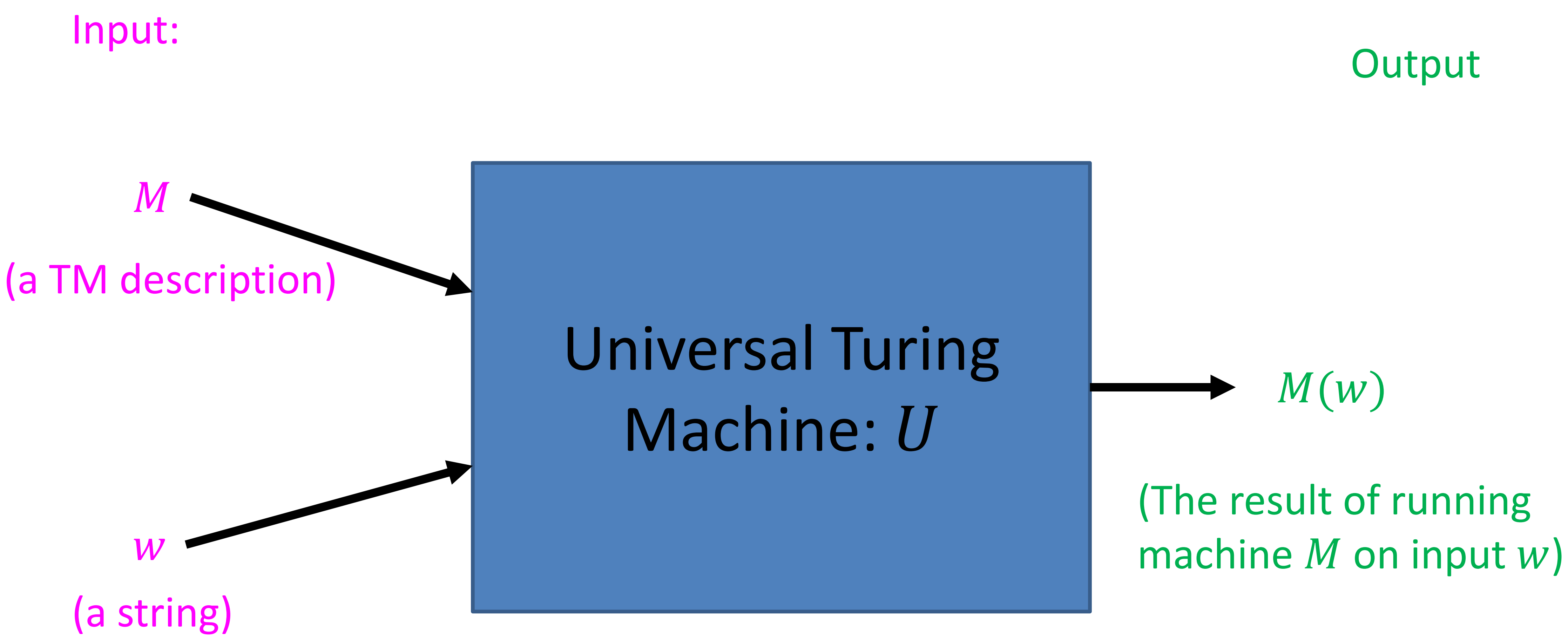
Input: bit string representing a program (first $S(s)$ bits) plus input values (remaining n bits)

Output: the result of running the represented program on the provided input, or m 0's if there's a "compile error"

Turing Universality

- Turing Machines are “Universal” in the sense that you can have a Turing Machine which can “simulate” any other Turing Machine
- Universal Turing Machine:
  - Input: The “description” of a machine and an input for that machine
  - Output: The same as the output the described machine would give for its input

Universal Turing Machine



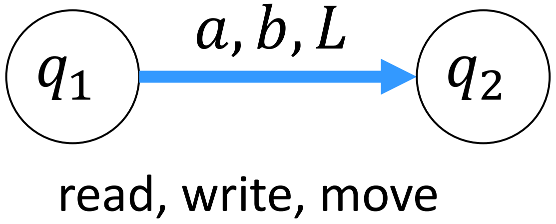
What does it need?

- What does a Universal Turing Machine need to have?
  - Memory in order to maintain the configuration of the machine you're simulating
    - Tape contents
    - Finite state "controller"
    - Current state
    - Current position on the tape
  - A way to take a transition
  - A way to keep going

Turing Machine

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

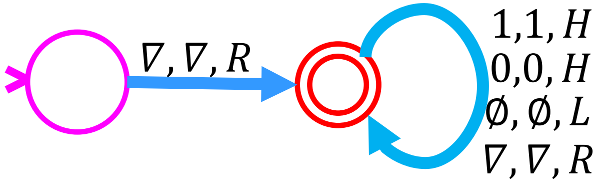
- Finite set of states: $Q = \{q_0, q_1, q_2, \ldots, q_k\}$ q_1
- Input alphabet: Σ
- Tape alphabet (includes $\emptyset$, ∇): Γ
- Transition function: $\delta: Q \times \overset{\text{Read}}{\Gamma} \to Q \times \overset{\text{Write}}{\Gamma} \times \overset{\text{Move}}{\{L, R, S, H\}}$
- Initial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



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

Some Turing Machines never return

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



$\nabla\emptyset\emptyset\emptyset$ …

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

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

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
    - Langauge: 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 langauge, and 0 otherwise
- Assertion:
  - This definition is the most powerful definition of computability that is physically possible
  - Why…?

What can’t be computed?

- Turing machines are really powerful
  - They can do complicated functions
- Turing machines are so powerful, you can use them to describe “nonsense”
  - Nonsense- paradox
- “colorful green ideas sleep furiously”
- “this statement is false” <- build a TM that says exactly this

The $ACCEPTS$ function

- "Reject" = Returns 0
- "Accept" = Returns 1
- $M(x) =$ the TM described by "source code" string x

- $ACCEPTS(x, w) = \begin{cases} 1 \text{ if } x \text{ running on } w \text{ returns } 1 \\ 0 \quad\quad\quad \text{otherwise} \end{cases}$
  - Situations in which we return 0:
    - When x doesn't halt
    - When x returns 0
  - What we have to do:
    - Recognize when a machine is in an infinite loop

Self-Rejecting Function

- $SelfReject(x) = \begin{cases} 1 \text{ when x } is\ a\ TM\ source\ code\ which\ rejects\ its\ own\ input \\ 0\ otherise \end{cases}$

- $SelfReject = \{w \in \{0,1\}^* | w \text{ represents a TM, and } w \notin L\big(M(w)\big)\}$
  - The set of all Turing machine source codes such that the described machine rejects is own description.
- X is the source code of a machine, SelfReject will accept x provided that x running on x rejects

Implementing $SelfReject$ With $ACCEPTS$

- Idea: run $ACCEPTS$ and flip the output
- Pseudocode for $SelfReject(w)$:
  1) Let $a = ACCEPTS(w, w)$
  2) If $a = 1$:

     Return 0
  3) Else:

     Return 1

What’s the problem?

- $SelfReject$ says “reject anything that accepts itself”, “accept anything that rejects itself”
- Let w_{SR} be the description of $SelfReject$
  - What is $SelfReject(w_{SR})$?
- Option1: $SelfReject(w_{SR}) = 1$
  - In words, w_{SR} accepted itself, and so by definition of SR, it should have been that SR(w_sr) = 0
- Option 2: $SelfReject(w_{SR}) = 0$
  - In other words w_{SR} is rejected by itself, and so by definition of SR, we conclude that it should be that $SelfReject(w_{SR}) = 1$
- Conclusion is, that any implementation of $SelfReject$ can’t produce an output that makes sense, therefore any implementation of $Self\ Reject$ must not be able to provide an output for w_{SR}

$w_{SR} \in SelfReject$?

Option 1: $\boldsymbol{w_{SR} \in SelfReject}$

Option 2: $\boldsymbol{w_{SR} \notin SelfReject}$