~~CSC363~~ CSC369 Tutorial 6

hope tomorrow's paul is feeling better!
(paul, having a headache and sick, on feb 23, while preparing slides)

Paul "sushi_enjoyer" Zhang

University of Amogus

February 24, 2021

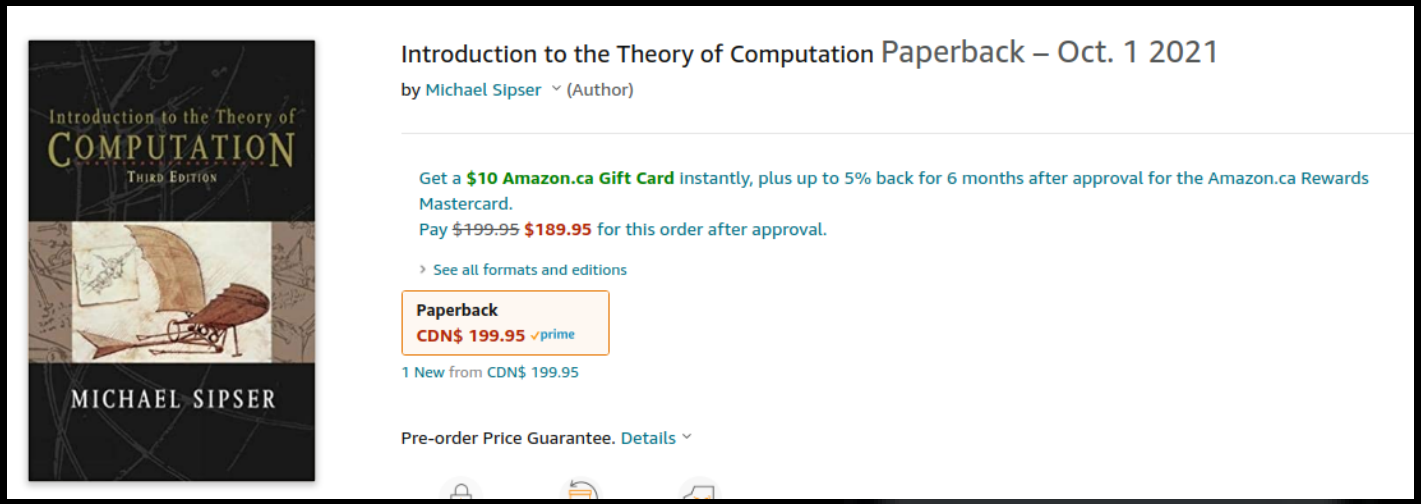
Learning objectives this tutorial

By the end of this tutorial, you should...

- Once again have CSIS (or whatever law enforcement for copyright violations in your jurisdiction) come to your house due to [REDACTED].
- Have recalled the formal definition of a Turing machine, and have built one in Minecraft (or any other Turing-complete game)
- Understand the formal definition of a multi-tape Turing machine, and have built one again in Minecraft (or any other Turing-complete game).
- Understand the formal definition of a nondeterministic Turing machine.
- Be infuriated by any further mentioning of computability, because we are completely done with that chapter of our lives.[1]

[1] If you hate computability, don't open assignment 3! (too bad you're forced to ;-;)

More unaffordable textbooks! yay :/



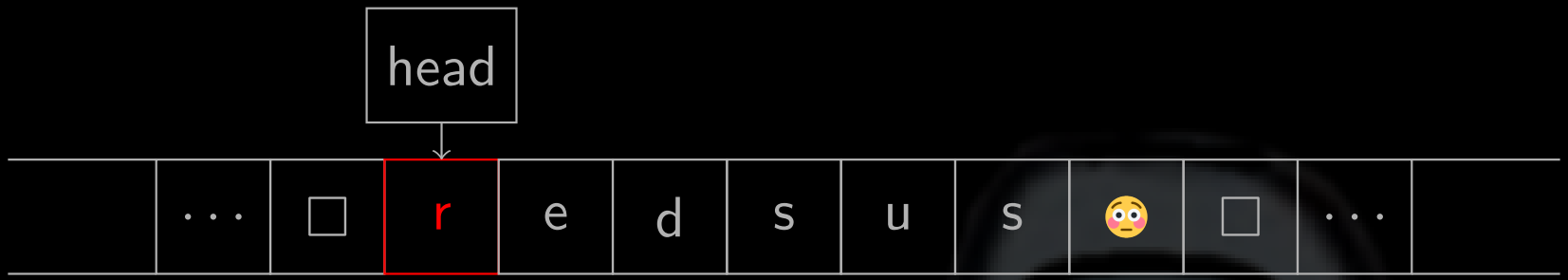
i'm broke from spending money on sushi. i can't afford this textbook.

Again, it's worth a read! you'll go over cool stuff like

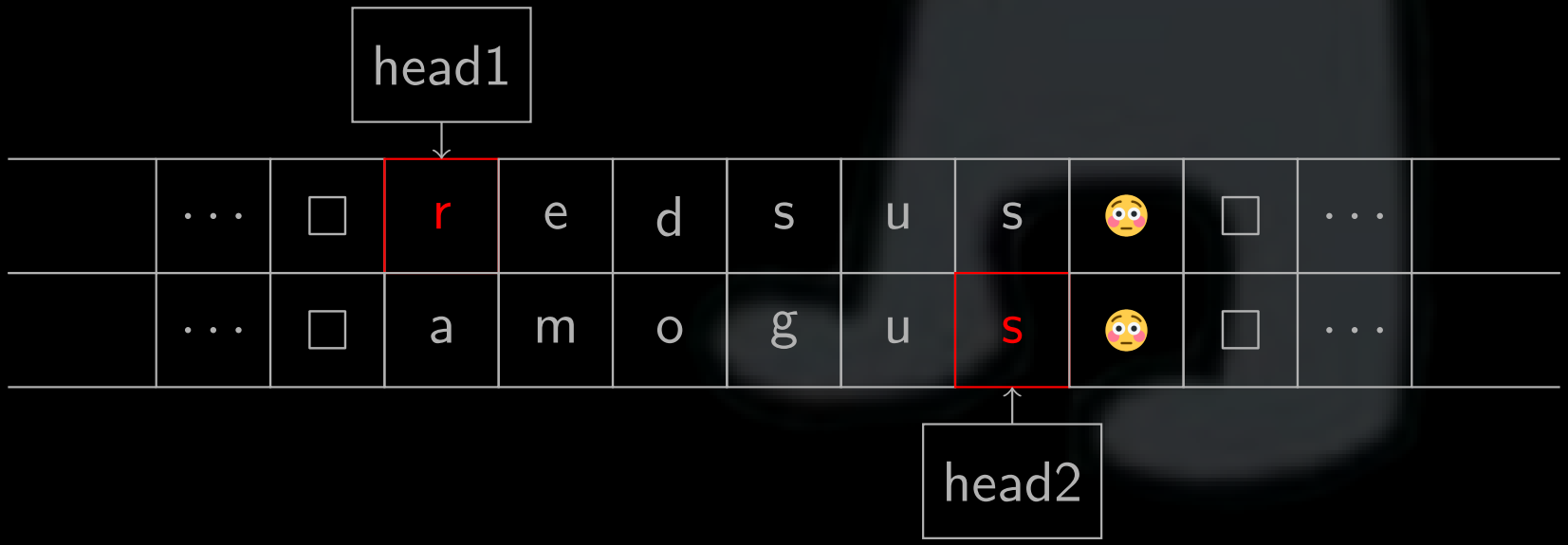
- What really is a Turing machine? (3.1)
- What are multi-tape and nondeterministic Turing machines? Why are they equivalent to Turing machines? (3.2)
- What does $f(n) = O(g(n))$ mean again? What does P mean? (7.1, 7.2)
- Why didn't we use this textbook earlier? (69.420)

Turing machines are back! :D

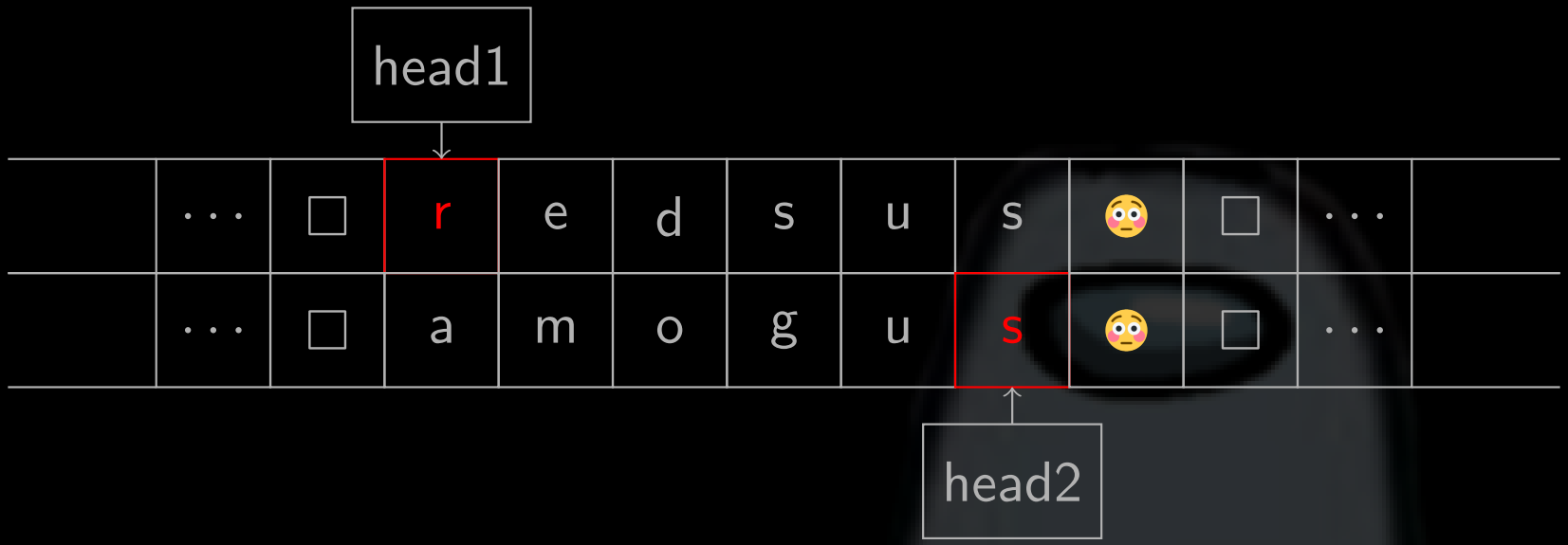
Hope you remember what a Turing machine is!



This Turing machine is sus 😳 it only has one tape. what if we had multiple tapes?



Turing machines are back! :D



This 2-tape Turing machine will read in 2 symbols at once (as a 2-tuple), consult the current state, then output the following:

1. Next state to transition towards;
2. Symbol to write back via head1, and direction to move head1;
3. Symbol to write back via head2, and direction to move head2.

Greek letters are back! D:

Do you like formal definitions? If you don't, too bad :(
Task: Recall that a Turing machine (the mathematical object) is an 8-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}, \square)$ where

- Q is the (finite) set of states;
- Σ is the input alphabet not containing $\square$;
- Γ is the tape alphabet, and $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function;
- $q_0 \in Q$ is the starting state;
- q_{accept} is the accept state;
- q_{reject} is the reject state (and $q_{\text{reject}} \neq q_{\text{accept}}$);
- $\square$ is the blank symbol.

Familiarize yourself with this definition. Ask any questions you have!

Greek letters are back! D:

Do you like formal definitions? If you don't, too bad :(
Task: The definition of a k-tape Turing machine (the mathematical object) is an 8-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}, \square)$ where

- Q is the (finite) set of states;
- Σ is the input alphabet not containing $\square$;
- Γ is the tape alphabet, and $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$;
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, -\}^k$ is the transition function;[2]
- $q_0 \in Q$ is the starting state;
- q_{accept} is the accept state;
- q_{reject} is the reject state (and $q_{\text{reject}} \neq q_{\text{accept}}$);
- $\square$ is the blank symbol.

Compare this with the previous definition. Ask any questions you have!

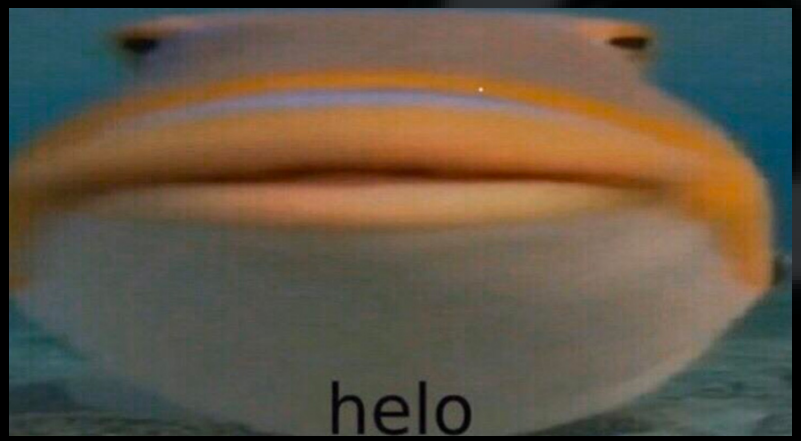
[2] $-$ is used to not move the head at all. This additional feature is not in the textbook definition of a k-tape Turing machine, but I added it as a user story in my spare time. Don't worry, the computational power is still equivalent.

Greek letters are back! D:

But how does the k-tape Turing machine execute?

Here are the differences between a k-tape Turing machine and a regular Turing machine:

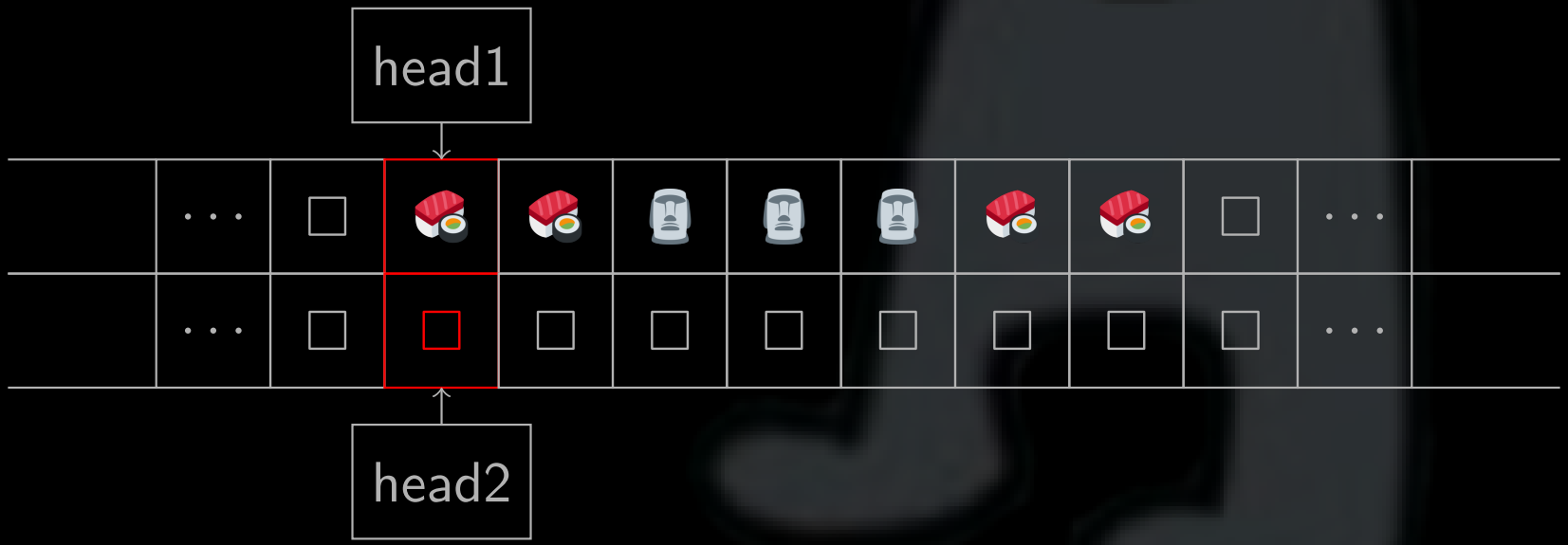
1. In a regular Turing machine, the string input is written on the (only) tape at the start of execution. In a k-tape Turing machine, the string input is always written on the first tape.
2. In a k-tape Turing machine, we read in k symbols at once, and we output k symbols and k directions from $\{L, R, -\}$, to specify the symbol to write and direction to move for each of the k tapes (where $-$ doesn't move it at all).
3. `helo_fish.jpg` likes multi-tape Turing machines, unlike regular Turing machines.



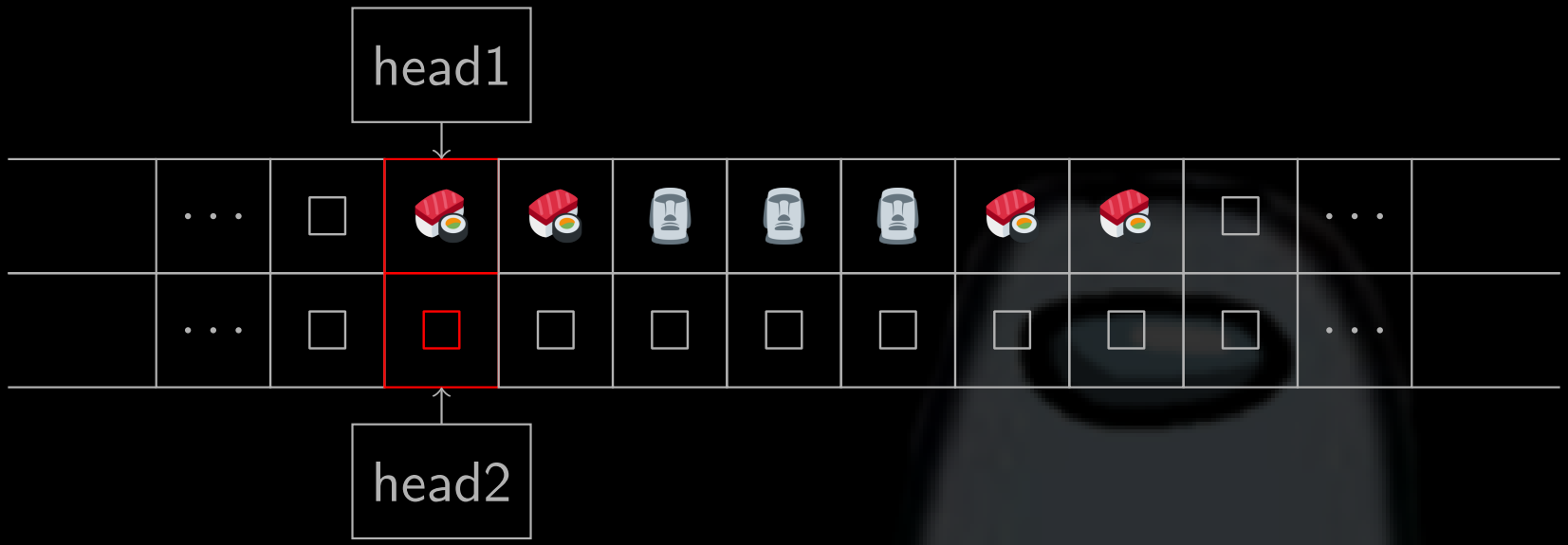
Let's try an example!

I want to construct a 2-tape Turing machine will check if a given binary string is a palindrome. For example, 🍣🗿🍣🍣🍣🍣🗿🍣 is accepted, while 🍣🗿🗿🗿🍣🗿🍣🗿 is not.

Say we are given input 🍣🍣🗿🗿🗿🍣🍣. The Turing machine would start out like so:

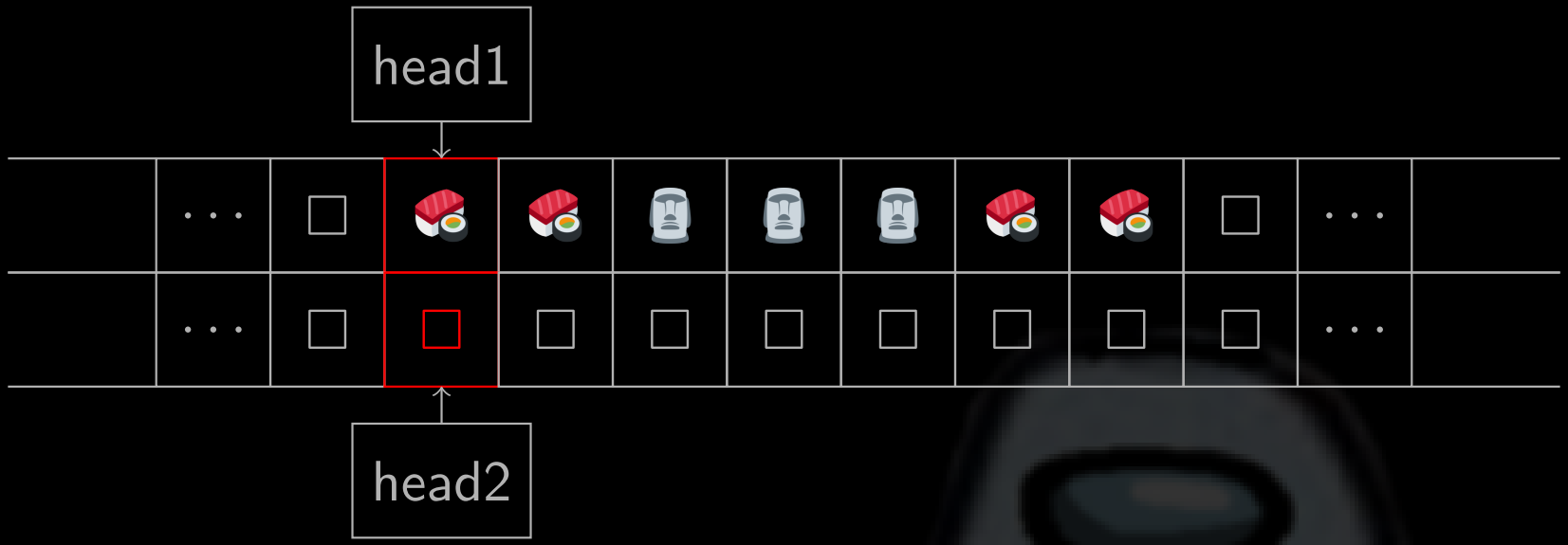


Let's try an example!

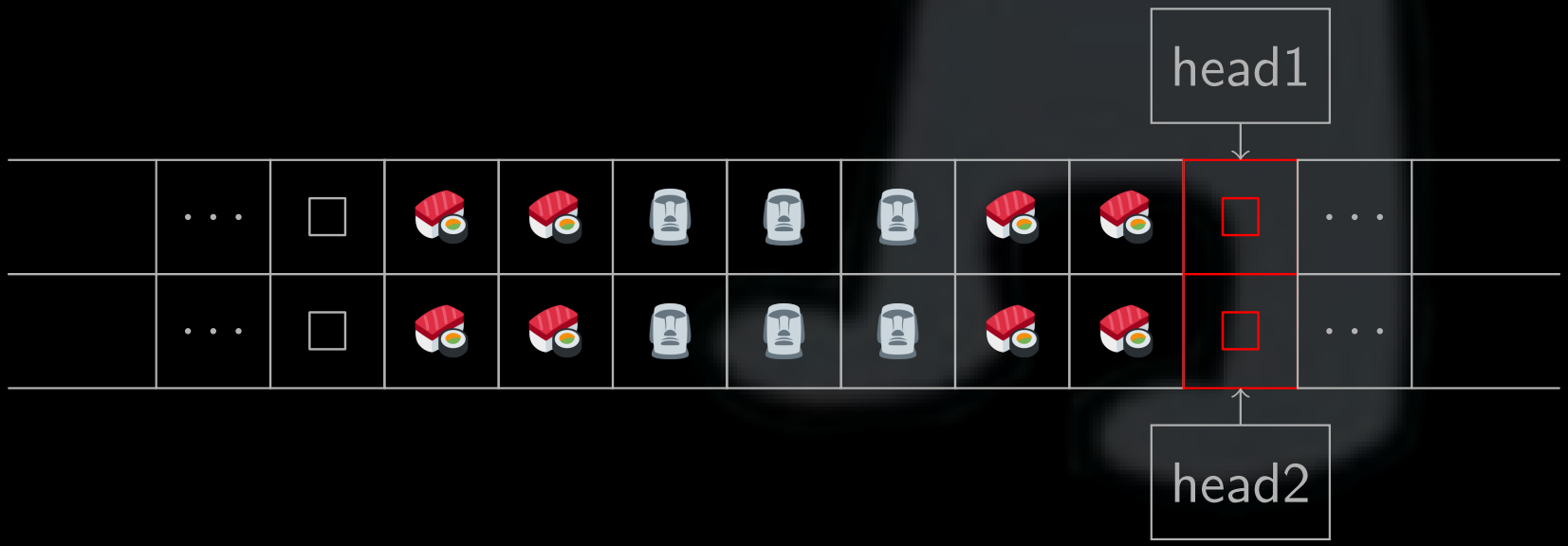


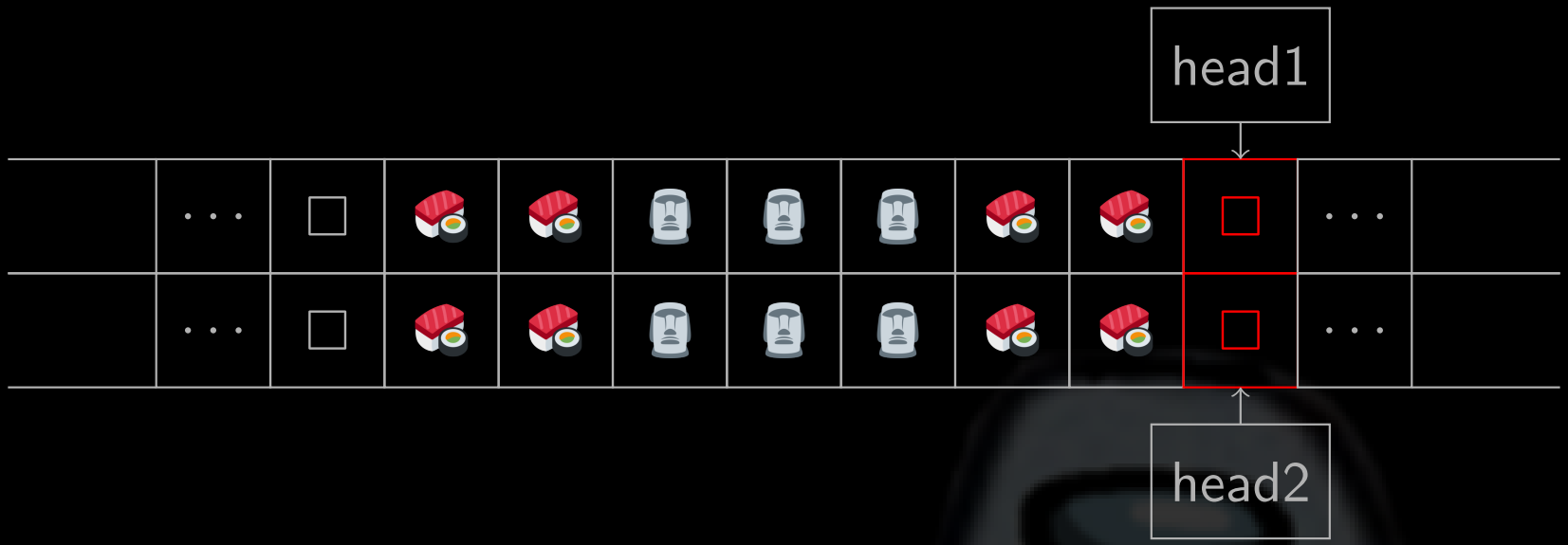
We'll program the Turing machine to do the following, at a high level:

1. Copy the string on the first tape to the second tape.
2. Move head1 to the beginning of the first tape, and head2 to the last character on the second tape.
3. Compare the symbols at head1 and head2. If they are not equal, reject. Else move head1 to the right and head2 to the left.

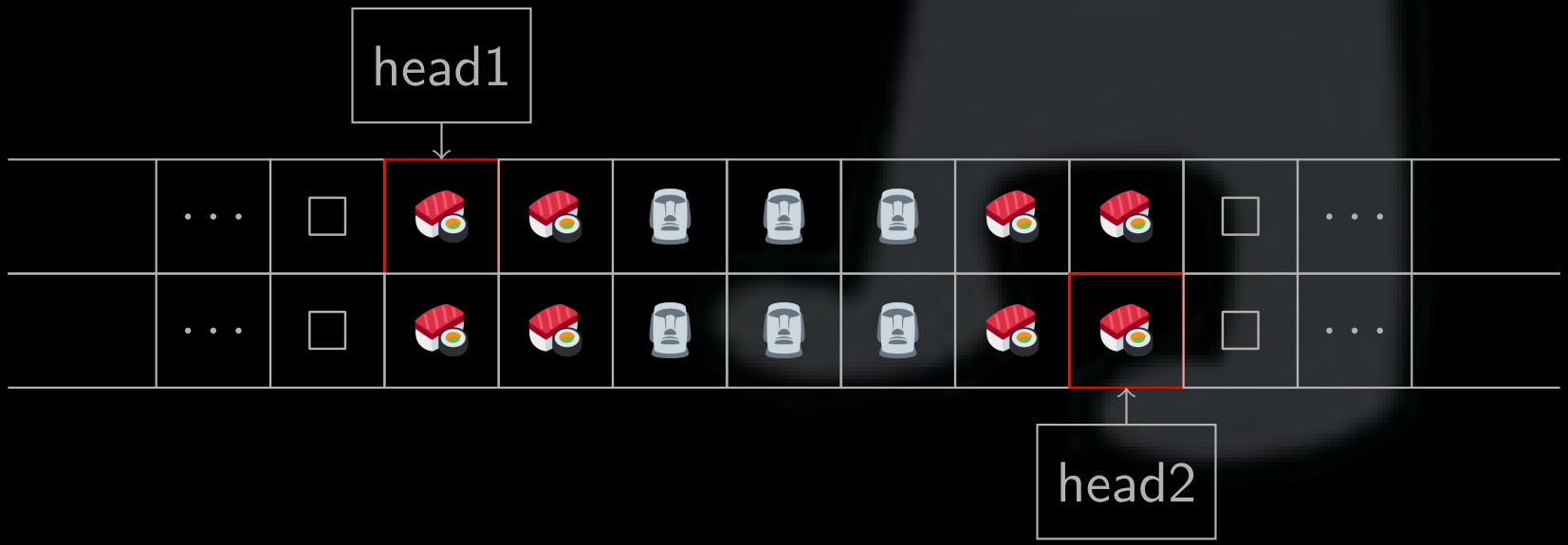


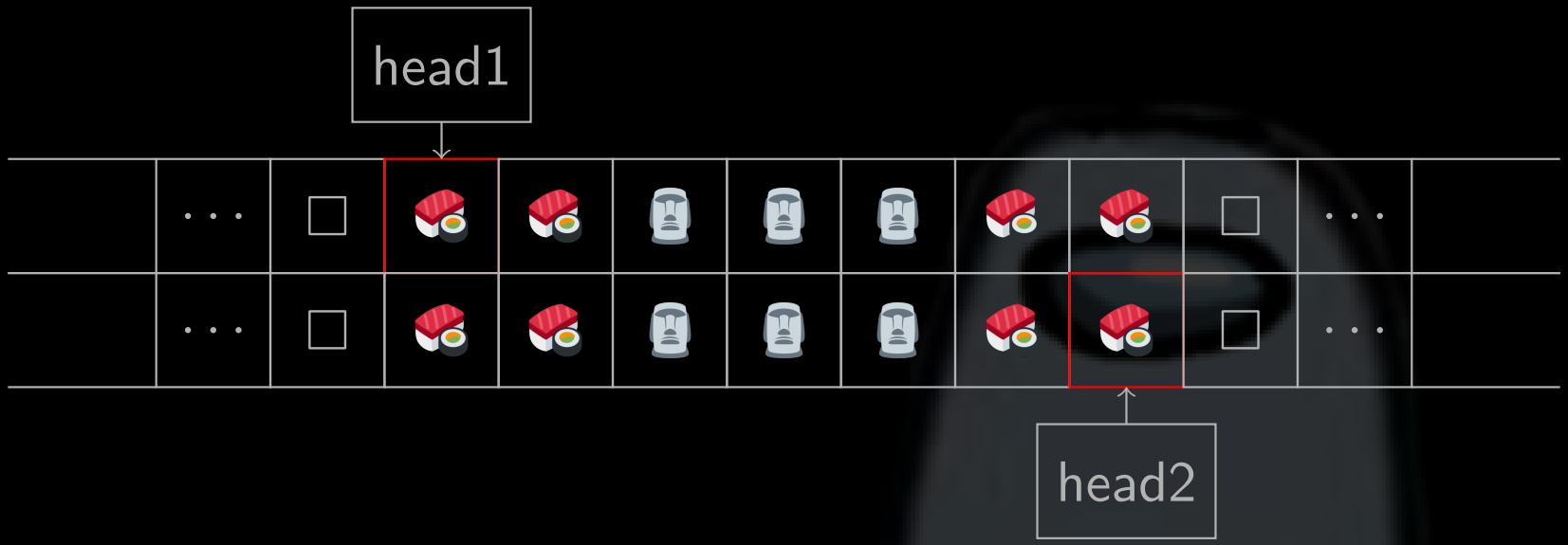
1. Copy the string on the first tape to the second tape.





2. Move head1 to the beginning of the first tape, and head2 to the last character on the second tape.





Compare the symbols at head1 and head2. If they are not equal, reject. Else move head1 to the right and head2 to the left.

I want to construct a 2-tape Turing machine will check if a given binary string is a palindrome. For example, 🍣🗿🍣🍣🍣🍣🗿🍣 is accepted, while 🍣🗿🗿🗿🍣🗿🍣🗿 is not.

We'll program the Turing machine to do the following, at a high level:

1. Copy the string on the first tape to the second tape.
2. Move head1 to the beginning of the first tape, and head2 to the last character on the second tape.
3. Compare the symbols at head1 and head2. If they are not equal, reject. Else move head1 to the right and head2 to the left.

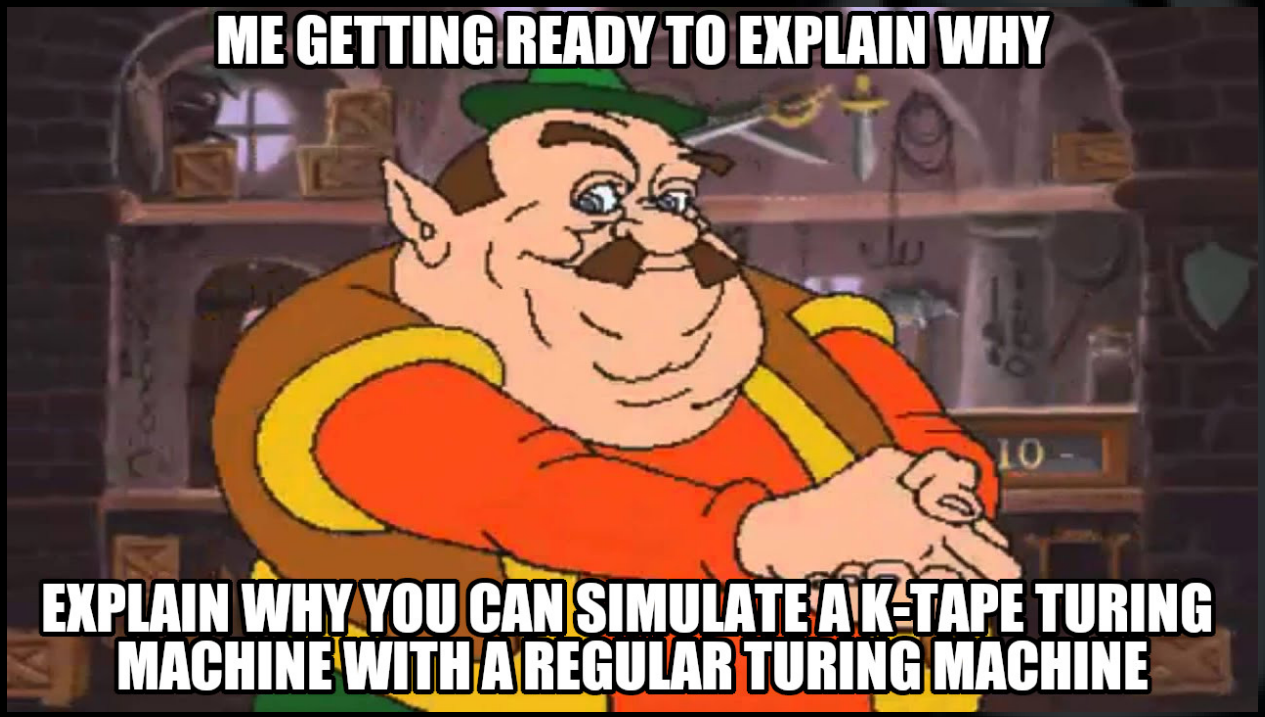
Task: ~~Formally write out this Turing machine.~~
Go to `https://turingmachinesimulator.com`, load the "Fast binary palindrome" example under "Examples".[3] Try to understand what is going on here! Ask questions!

[3] Or just click here: `http://turingmachinesimulator.com/shared/pxgabhhejv`

Why is this relevant?

~~Cuz I was told to cover this in tutorial~~ Because they are actually equivalent in computational power! (not exactly, but they are kinda equivalent)

By that we mean something is "computable" by a regular Turing machine if and only if it is "computable" by a multi-tape Turing machine, whatever reasonable definition of "computable" you use.



(not actually gonna do it, it's in the textbook though!)

Why is this relevant?

They aren't necessarily equivalent in terms of *computational complexity* though.[4]

Our 2-tape Turing machine takes $O(n)$ steps to determine whether a string of length n is a palindrome or not. In a regular Turing machine, we're probably gonna need $O(n^2)$ steps. So in some sense, multi-tape Turing machines are more efficient at computation!

[4] By computational complexity, I mean the *number of steps* it takes to halt. Refer to CSC236/CSC263 or whatever, lol.

Why is this relevant?

there's a nice theorem in the textbook! (chapter 7, theorem 7.8)

Theorem (kinda)

If something is computable in $t(n)$ time by a multi-tape Turing machine, then it is also computable in $O((t(n))^2)$ time by a regular Turing machine.

you don't need to fully understand what this means! informally it's just saying whatever a multi-tape Turing machine can do, a regular Turing machine can do as well, but possibly slower!

break!

pls dont copyright strike me, kero kero bonito sama uwu
https://www.youtube.com/watch?v=PEmqJBcQ2lg

non-deterministic turing machines!

because sometimes, just like humans, turing machines can't make up their minds ;-;

Remember DFAs and NFAs from CSC236? Me neither. But there's a good analogy between

Turing Machine : Nondeterministic Turing Machine

and

DFA: NFA

In a Nondeterministic Turing machine, there are multiple possible execution paths!

non-deterministic turing machines!

because sometimes, just like humans, turing machines can't make up their minds ;-;

Remember DFAs and NFAs from CSC236? Me neither. But there's a good analogy between

Turing Machine : Nondeterministic Turing Machine

and

DFA: NFA

In a Nondeterministic Turing machine, there are multiple possible execution paths!

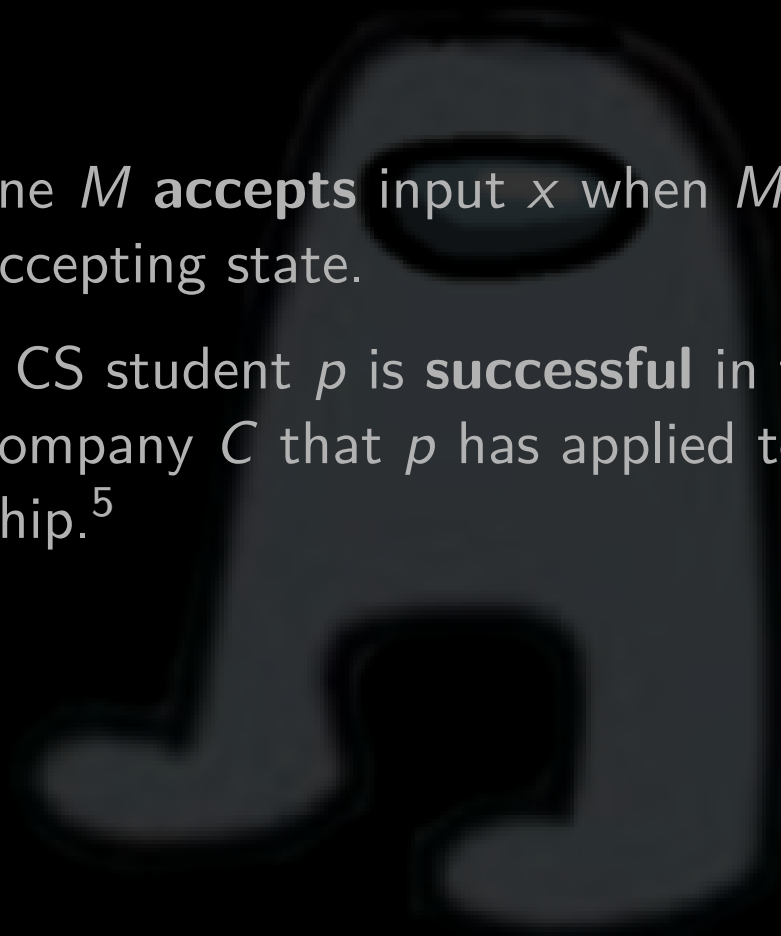
Greek letters are back, once again!

Do you like formal definitions? If you don't, too bad :(
Task: The definition of a nondeterministic Turing machine (the mathematical object) is an 8-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}, \square)$ where

- Q is the (finite) set of states;
- Σ is the input alphabet not containing $\square$;
- Γ is the tape alphabet, and $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$;
- $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ is the transition **relation**;
- $q_0 \in Q$ is the starting state;
- q_{accept} is the accept state;
- q_{reject} is the reject state (and $q_{\text{reject}} \neq q_{\text{accept}}$);
- $\square$ is the blank symbol.

Take a moment to understand why $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ instead of $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$. Ask questions!

How does a nondeterministic Turing machine accept some input?



A nondeterministic Turing machine M **accepts** input x when M has *some* execution path that halts in an accepting state.

An analogy: We say an awkward CS student p is **successful** in their job application when there is *some* company C that p has applied to, for which C has offered p an internship.[5]

[5] Even if C is underpaying p, which happens all the time, unfortunately :(

The *POWER* of nondeterministic Turing machines (at a high level)

unfortunately i won't be going over any low-level examples of nondeterministic Turing machines today :(

Subset Sum Problem: Given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$ and a *target sum* T, write a program (a Turing machine) to determine if S has a subset $S' \subseteq S$ such that the sum of S', $\sum S'$, is equal to T.

For example, say we are given $S = \{1, 2, 3, 4, 5, 6\}$, and $T = 11$. Then the program should say "yes", because there exists a subset $S' \subseteq S$ such that $\sum S' = T$: if we take $S' = \{1, 2, 3, 5\}$ then $\sum S' = 11$.

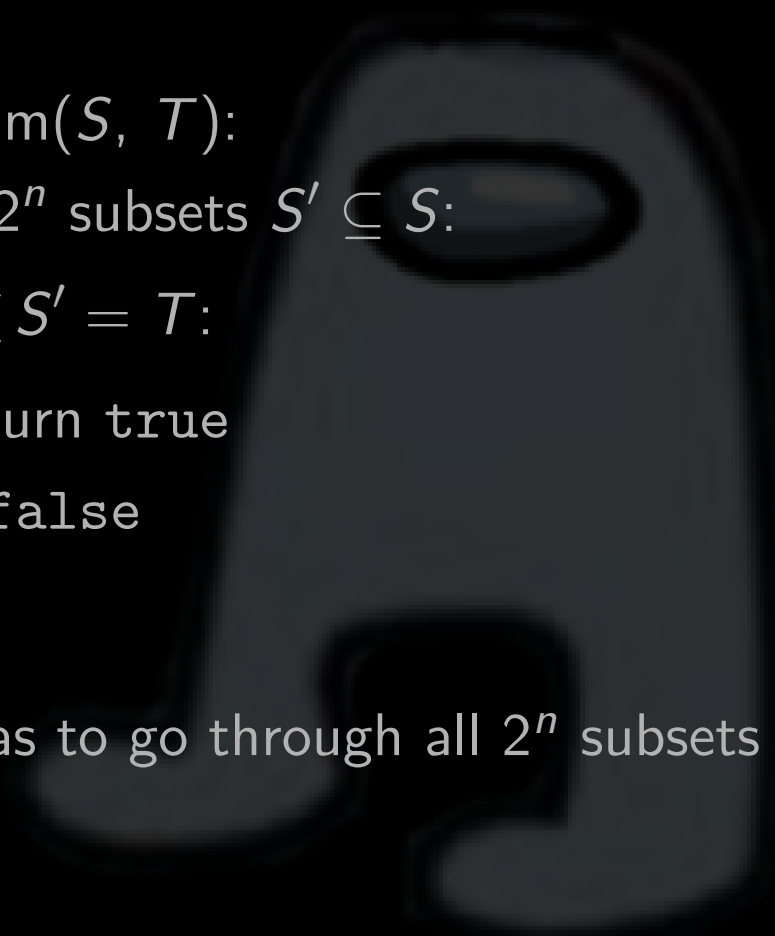
For example, say we are given $S = \{5, 5, 5, 6, 6, 6\}$, and $T = 13$. Then the program should say "no", because there does not exist a subset $S' \subseteq S$ such that $\sum S' = T$. You can convince yourself by *checking every subset of* S.

Task: Write a program (a Turing machine) that solves the Subset Sum Problem (i.e. given S and T, return `true` iff S has a subset S' such that $\sum S' = T$). Use Pseudocode!

The *POWER* of nondeterministic Turing machines (at a high level)

Answer: something like

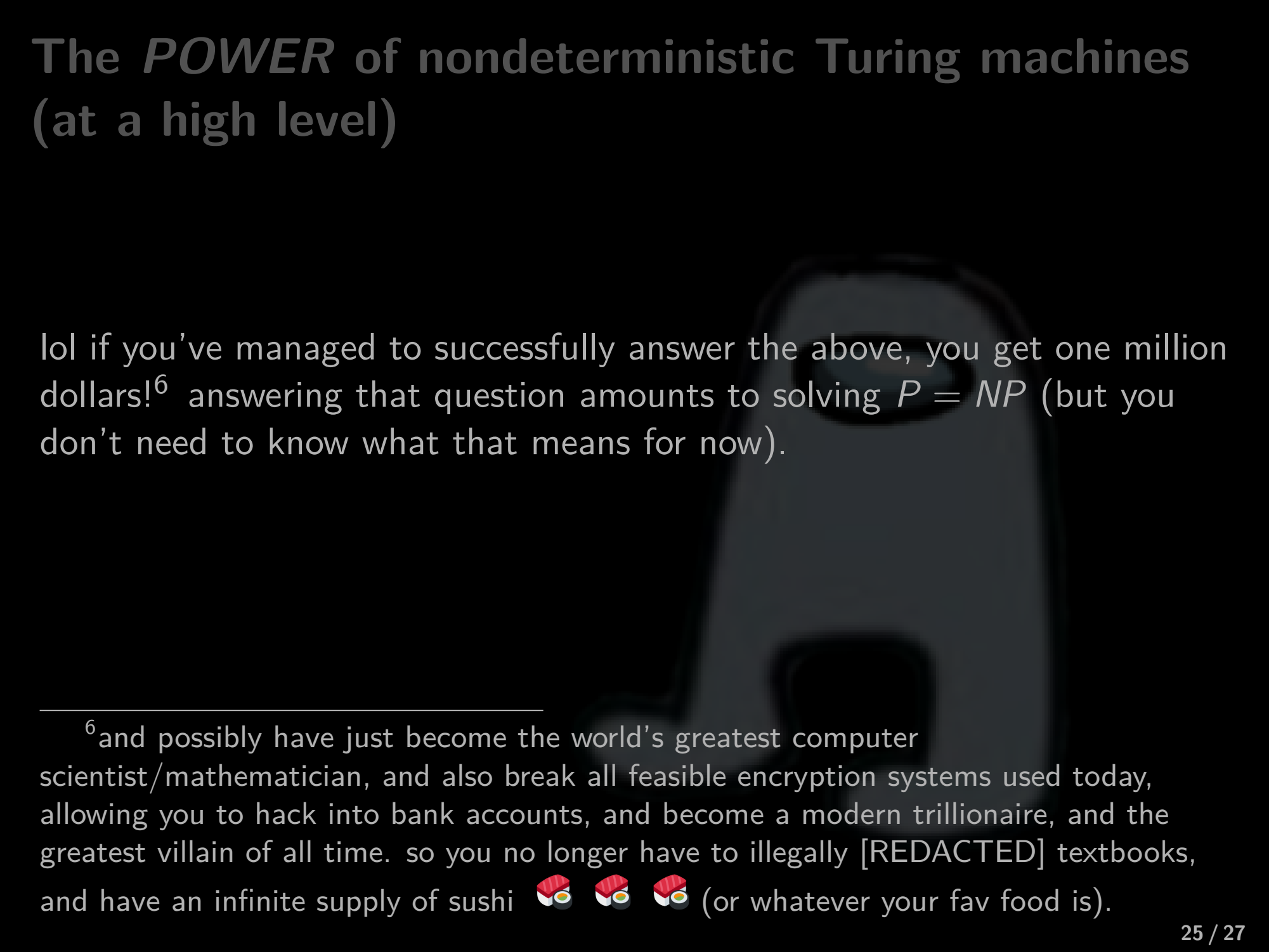
Subset-Sum(S, T):
  For all 2^n subsets $S' \subseteq S$:
    if $\sum S' = T$:
      return `true`
  return `false`



This code is very inefficient: it has to go through all 2^n subsets of S! Can we do any better?

Task: Answer the above.

The *POWER* of nondeterministic Turing machines (at a high level)



lol if you've managed to successfully answer the above, you get one million dollars![6] answering that question amounts to solving $P = NP$ (but you don't need to know what that means for now).

[6] and possibly have just become the world's greatest computer scientist/mathematician, and also break all feasible encryption systems used today, allowing you to hack into bank accounts, and become a modern trillionaire, and the greatest villain of all time. so you no longer have to illegally [REDACTED] textbooks, and have an infinite supply of sushi 🍣 🍣 🍣 (or whatever your fav food is).

The *POWER* of nondeterministic Turing machines (at a high level)

We can do better with nondeterministic Turing machines! [7]

Define the nondeterministic Turing machine M using the following pseudocode:

Subset-Sum(S, T):

  Choose a particular subset $S' \subseteq S$:

  if $\sum S' = T$:

    return `true`

  return `false`

Task: Answer the following:

1. Why is the above code is allowed (as a high-level description of a nondeterministic Turing machine)?
2. Why is it that M accepts (S, T) if and only if (S, T) satisfies the subset sum problem? Recall what it means for a nondeterministic M to accept an input.

[7] unfortunately this isn't really practical in real life, so no infinite sushi supply yet :(

end i think?

hey, uh, sorry if today's tutorial seemed pretty rushed! you don't have to fully understand what a multi-tape turing machine or a nondeterministic turing machine is, as long as you get the high level concepts:

- multi-tape turing machines read and write multiple symbols at once
- whatever a multi-tape turing machine can compute, a regular turing machine can compute as well (possibly taking more time to compute)
- nondeterministic turing machines can have multiple different execution paths (think of it like alternate universes!), and are more efficient than regular turing machines, but aren't really practical in real life unfortunately :/
- big chungus is gone, i am sick, and i don't know what i'm writing for these slides anymore