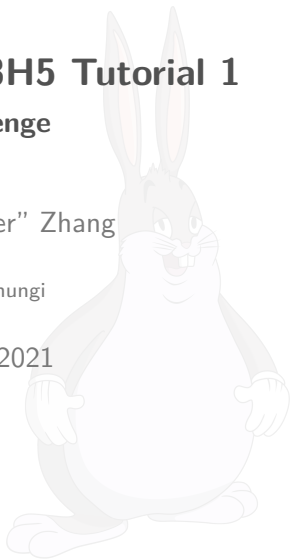# ~~CSC363H5~~ CSC258H5 Tutorial 1
## Paul's Revenge

Paul "sushi_enjoyer" Zhang

University of Chungi

January 13, 2021

## Learning objectives this tutorial

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

▶ Be able to name the TA and his favourite food. This is where your tuition is going.

▶ Have become good friends with everyone in this tutorial session.

▶ Be able to describe what a URM is, and relate it to your knowledge of assembly programming (if you've done that before).

▶ Be able to create URMs that perform operations such as addition.

▶ Grasp the idea of a Turing Machine at a very informal level.

▶ Appreciate the computational power of the personal computer, but realize that it only can do the things that a Turing machine can, which can only do things that a URM can.

▶ See helo_fish.jpg in your dreams.

▶ Have a proof for $P = NP$, the Riemann hypothesis, and have attained nirvana.
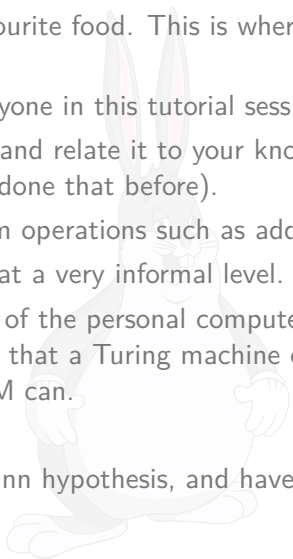
## helo!

Welcome to CSC363! Here's `helo_fish.jpg`.
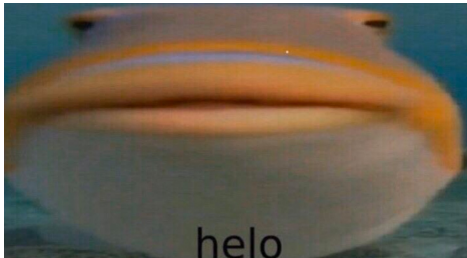


`helo_fish.jpg` shall revisit my tutorial from time to time. Today, `helo_fish.jpg` sends her greetings and hopes she will not haunt your dreams.

`helo_fish.jpg` will say goodbye for now. Say your farewells!

**Protip...**

`helo_fish.jpg` wants you to make friends in this tutorial session! Say hi to the people sitting right beside you.

## a bit about myself, i guess?

Hai! I'm paul (he/him), B.Sc., M.Sc, Ph.D, D.P.H, S.Sc.D, and others (in minecraft). Address me however you'd like!

I'm doing a mathematics specialist (i'm done my cs minor).
My favourite ice cream flavour is coffee.

Current status: sad and alone in downtown toronto with nae pals ;-; kinda lonely from time to time, wanna chat with me? owo

- ▶ office hours: Wed 5-6pm (after this tut), same zoom link
- ▶ email: `pol.zhang@mail.utoronto.ca`
- ▶ discord: `sjorv#0943`
- ▶ website: `https://sjorv.github.io/` (not finished yet sowwy)
- ▶ reddit: guess lol
- ▶ facebook/instagram/twitter/whatsapp/tiktok/snapchat/ vkonkakte/line/amazon/quercus/tinder/chungustalk: nope, sorry :((
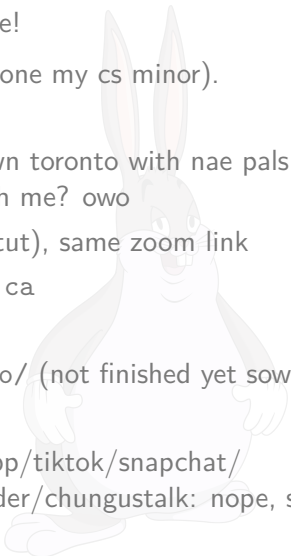
## Oh yea, i just realized something...

When making those slides, I realized that for people enrolled in the Friday lecture, you will be attending this tutorial before you have learned any course content.

I've tried to make the slides as self-contained as possible, so you don't need knowledge from the lecture. But if I have any gaps (or you have questions), please don't hesi**taste** to ask! owo

Also, we are doing some really **low level** "programming" stuff today. Knowledge of assembly language is not required by any means, but if you've done assembly programming before it might help to relate today's content to it.
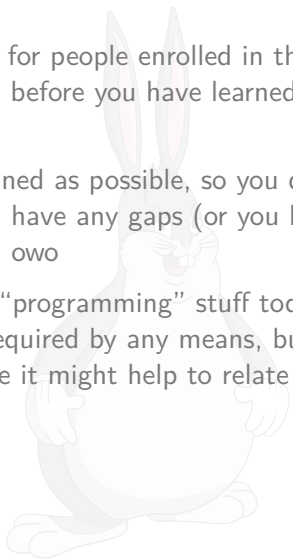
# What's a URM?

If you recall from class, we wanted to mathematically define what it means for a function to be "computable". For simplicity, we only consider functions $f : \mathbb{N}^k \to \mathbb{N}$ for now.

A **U**niform **R**esource **M**achine is one way to formally define computability of functions.
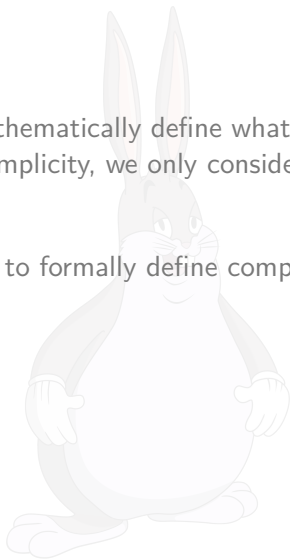
## What's a URM?

Consider a "tape" that extends infinitely in one direction. This tape can be thought of as memory in a computer (except we have infinite memory).

We will label the positions on the tape (or *registers*) $R_1, R_2, \ldots$. Each register can store a **natural** number (including 0).

| $R_1$ | $R_2$ | $R_3$ | $R_4$ | $\cdots$ | |
|---|---|---|---|---|---|

We will manipulate the values on the tape according a series of instructions, called a *URM program*. The instructions are very similar to assembly programming if you have done it before.

## What's a URM?

A URM program consists of a series of *basic instructions*. The instructions are numbered $1, 2, \ldots, m$, where $m$ is the number of instructions the URM program has.

There are four basic instructions:

▶ The zero instruction $Z(n)$: replace the number in $R_n$ (the $n$th position on the tape) with 0.

▶ The successor instruction $S(n)$: Add 1 to the number in $R_n$ (i.e. increment it).

▶ The copy instruction $C(m, n)$: Replace the number in $R_n$ with the number in $R_m$. This does not affect $R_m$.

▶ The jump instruction $J(m, n, q)$: If the numbers in $R_m$ and $R_n$ are equal, go to instruction $q$ (otherwise go to the next instruction).[1]

---

[1] If $q > m$ (so the $q$th instruction doesn't exist), halt.

## What's a URM?

The URM takes in a $k$-tuple $(a_1, a_2, \ldots, a_k) \in \mathbb{N}^k$ as input.

We start with values $a_1$ in $R_1$, $a_2$ in $R_2$, and so on, until $a_k$ in $R_k$. For $i > k$, $R_i$ starts with 0 (so all the other registers start as 0).

| $a_1$ | $a_2$ | $\cdots$ | $a_k$ | 0 | 0 | $\cdots$ |
|---|---|---|---|---|---|---|

Then the URM executes its instructions in order.

The output of the URM is a natural number: whatever number remains in $R_1$ at the end of execution is the output.

Note that some URMs don't terminate (in the case of a loop).
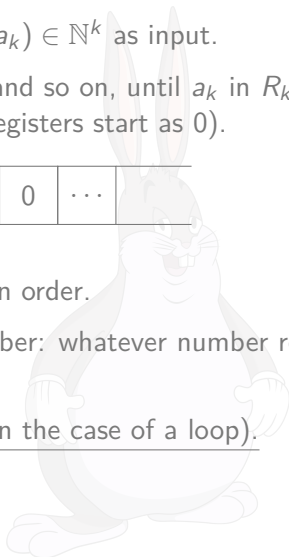
## What's a URM?

Then, we can say a function $f : \mathbb{N}^k \to \mathbb{N}$ is *computable* if there exists a URM $M$ such that for any input $(a_1, \ldots, a_k) \in N^k$, $M$ terminates on input $(a_1, \ldots, a_k)$ and outputs $f(a_1, \ldots, a_k)$.
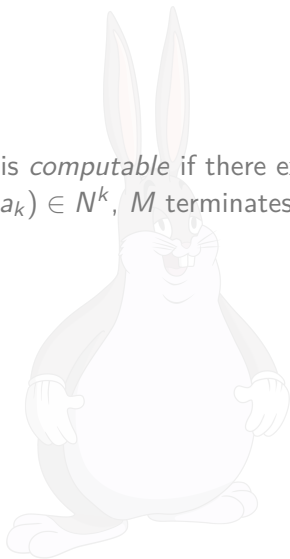
## Example

Let $f$ be the function that adds two natural numbers:

$$f : \mathbb{N}^2 \to \mathbb{N}, f(a, b) = a + b.$$

We construct a URM that computes $f$.

URM instructions:
  1: $J(2, 3, 69420)$
  2: $S(1)$
  3: $S(3)$
  4: $J(1, 1, 1)$

Note if the numbers in $R_2$ and $R_3$ are equal, $J(2, 3, 69420)$ points to a nonexistent instruction and therefore halts execution. $J(1, 1, 1)$ jumps to instruction 1 unconditionally.
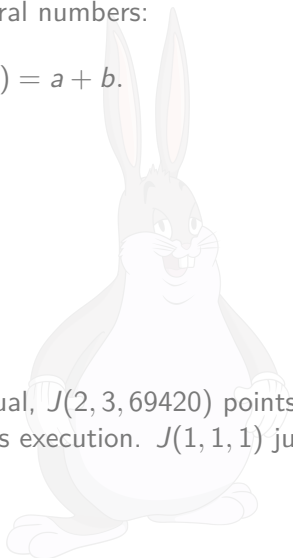
## Example

1: $J(2, 3, 69420)$
2: $S(1)$
3: $S(3)$
4: $J(1, 1, 1)$

On input $(3, 2)$, we start with the following tape:

| 3 | 2 | 0 | 0 | $\cdots$ | |

We start executing from line 1: $J(2, 3, 69420)$. Nothing happens since $2 \neq 0$.

| 3 | 2 | 0 | 0 | $\cdots$ | |

# Example

1: $J(2, 3, 69420)$
2: $S(1)$
3: $S(3)$
4: $J(1, 1, 1)$

| 3 | 2 | 0 | 0 | $\cdots$ | |
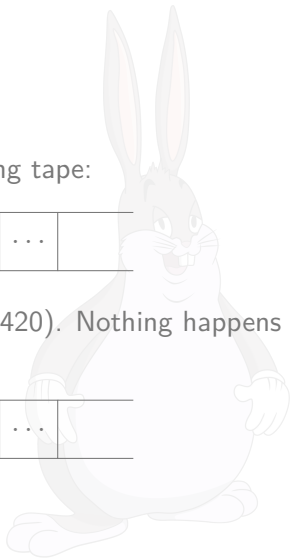
2: $S(1)$

| 4 | 2 | 0 | 0 | $\cdots$ | |

3: $S(3)$

| 4 | 2 | 1 | 0 | $\cdots$ | |

# Example

1: $J(2, 3, 69420)$
2: $S(1)$
3: $S(3)$
4: $J(1, 1, 1)$

| 4 | 2 | 1 | 0 | $\cdots$ | |

4: $J(1, 1, 1)$
1: $J(2, 3, 69420)$
2: $S(1)$

| 5 | 2 | 1 | 0 | $\cdots$ | |

3: $S(3)$

| 5 | 2 | 2 | 0 | $\cdots$ | |

## Example

1: $J(2, 3, 69420)$
2: $S(1)$
3: $S(3)$
4: $J(1, 1, 1)$

| 5 | 2 | 2 | 0 | $\cdots$ | |

4: $J(1, 1, 1)$
1: $J(2, 3, 69420)$
We now halt since the numbers in $R_2$ and $R_3$ are equal (both 2). We output 5, which is indeed $f(3, 2)$.
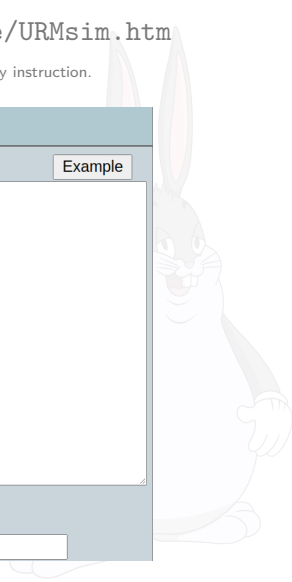
# A useful tool

Try it out yourself!

https://sites.oxy.edu/rnaimi/home/URMsim.htm
(Sorry Firefox users, it's time to whip out Microsoft Edge.)
Note: The website uses $T(m, n)$ instead of $C(m, n)$ to denote the copy instruction.

```
                    INPUT

URM program:                        [ Example ]

1: J(2, 3, 69420)
2: S(1)
3: S(3)
4: J(1, 1, 1)




Initial register values:

3, 2
```

## Now it's your turn!

Let $f : \mathbb{N} \to \mathbb{N}$, $f(a) = 3a$. Build a URM to compute this function.

If you're done, try building a URM for this function:

$$f : \mathbb{N} \to \mathbb{N}, f(a) = \begin{cases} 0 & a \text{ even} \\ 1 & a \text{ odd}. \end{cases}$$
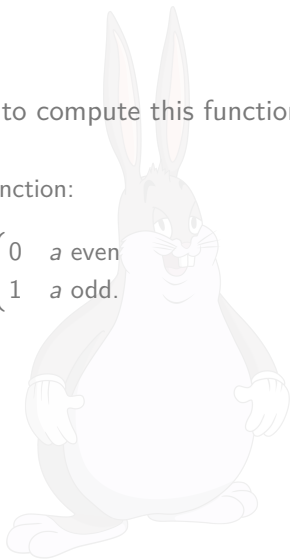
## Solution

There are multiple different URMs that compute $f : \mathbb{N} \to \mathbb{N}$, $f(a) = 3a$.
Example:

1: $S(5)$
2: $S(5)$
3: $C(1, 2)$
4: $J(2, 3, 8)$
5: $S(1)$
6: $S(3)$
7: $J(1, 1, 4)$
8: $Z(3)$
9: $S(4)$
10: $J(4, 5, 2020)$
11: $J(1, 1, 4)$
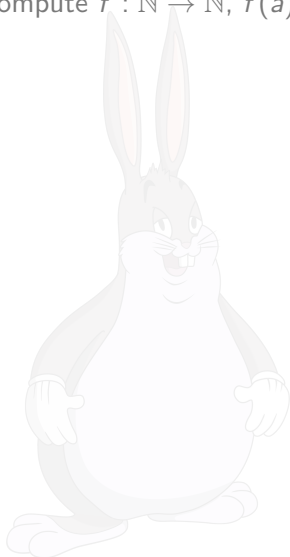
# Turing machines!

~~Now that you have the URM in the back of your mind, I can introduce Turing machines!~~
actually nvm it would take way too much time. Here's the informal idea of a Turing machine (don't worry if you don't understand everything here):[2]

You have a tape, as before, but this tape extends infinitely in both directions. You also have a *read-write head*, pointing to a position on the tape.

Each position on the tape can store a symbol in a specified *alphabet* (remember CSC236!). For example, our alphabet could be $\{0, 1\}$, or it could be the set of all ASCII characters. There is a specially designated *empty symbol*, usually denoted by 0, that the tape is filled with.

---
[2]There are many equivalent ways of defining the Turing machine. This may differ from what is defined in this class later.

# Turing machines!

The Turing machine takes in a string input, say "sushi". The execution starts with "sushi" written on the tape (and 0 everywhere else), and the read-write head pointing to the first letter of "sushi".

| | | head | | | | | | |

| ··· | 0 | s | u | s | h | i | 0 | ··· |
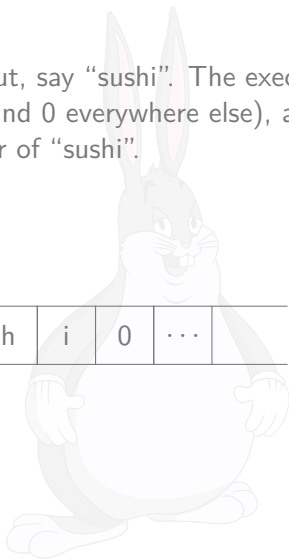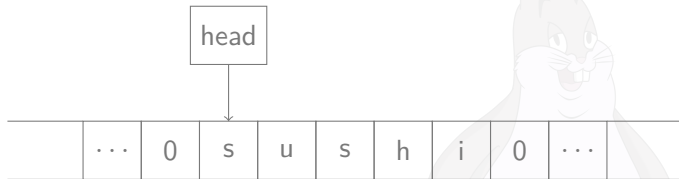
# Turing machines!

The Turing machine also has a set of predetermined *states*, which can be encoded in a DFA (from CSC236). Each state dictates the current behaviour of the Turing machine.

In each iteration of the Turing machine, we do the following:

1. Read the symbol, from the read-write head.
2. Inquire the current state: what should I do if I have read in this symbol? The current state will give us the following:
   ▶ A symbol to write back. (This symbol might be the same as the symbol we read in.)
   ▶ A new state to transition to (which might be the same state).
   ▶ A direction to move the read-write head in by one, either left or right (we **must** move the read-write head).

In addition, the Turing machine specifies a stating state, and a list of halting states, just like in a DFA. After halting, the Turing machine outputs whatever is left on the tape.
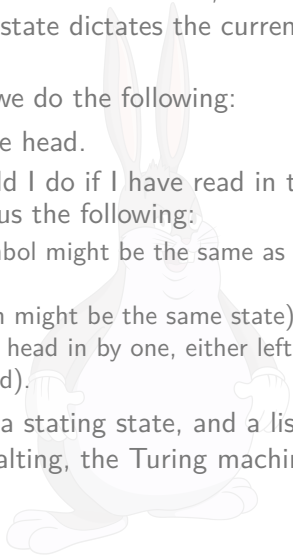
# Turing machines!

That was very informal. Here's the formal definition from Wikipedia:



Greek letters scare people so that's why I didn't feel like giving a full overview of the Turing machine today, sowwy owo

The catchline is that although Turing machines are more complicated than URMs, whatever Turing machines can compute, URMs can also compute, and vice versa. So in some sense their "computational power" is equivalent! In fact, theoretically you could simulate a Turing machine inside a URM, and vice versa.

# Example of Turing machine!

Let's make sushi! owo

Our alphabet will be $\left\{ 0, 🐟, 🍚, 🍣 \right\}$. The states in the DFA for the Turing machine will be $\{empty, rice, fish, \textbf{finish}, \textbf{fail}\}$ with *empty* being the starting state and bold states being halting states.

Remember, each state takes in a symbol from the alphabet and outputs three things: a symbol to write back, a new state to transition to, and a direction to move the read-write head.
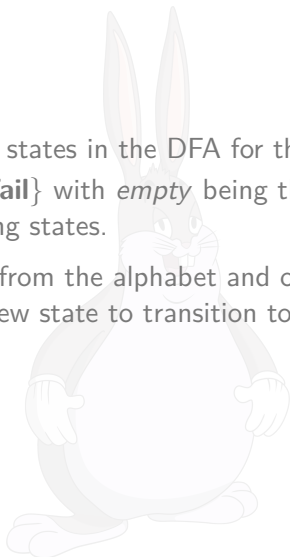
# Example of Turing machine!
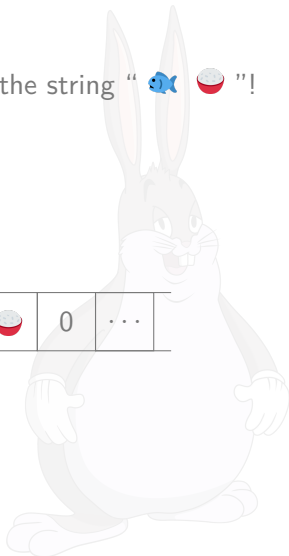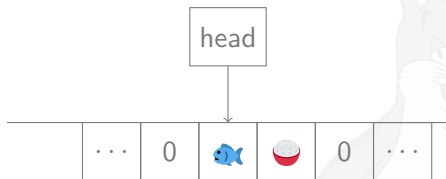
Consider the following table:

| State | 0 | 🐟 | 🍥 | 🍣 |
|---|---|---|---|---|
| *empty* | (0, **fail**, R) | (0, fish, R) | (0, **fail**, R) | - |
| fish | (0, **fail**, R) | (0, **fail**, R) | ( 🍣 , **finish**, R) | - |
| rice | (0, **fail**, R) | ( 🍣 , **finish**, R) | (0, **fail**, R) | - |
| **finish** | - | - | - | - |
| **fail** | - | - | - | - |

where − denotes that we will never encounter this symbol while in this state (so we can put anything we want there)

# Example of Turing machine!

Let's try executing our sushi machine on the string " 🐟 🍥 "!
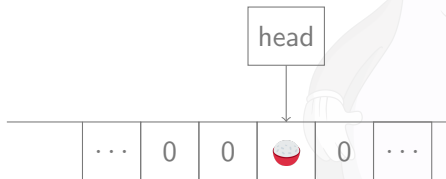
Current State: *empty*

# Example of Turing machine!

We read in 🐟 , and consult our current state *empty*.

| State | 0 | 🐟 | 🍣 | 🍱 |
|---|---|---|---|---|
| *empty* | (0, **fail**, R) | (0, fish, R) | (0, **fail**, R) | - |

Our current state returns (0, fish, R). So we write back 0, go to the fish state, and move the head right.
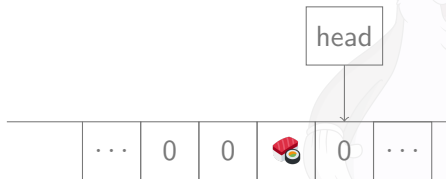
Current State: fish

# Example of Turing machine!

We read in 🍥 , and consult our current state fish.

| State | 0 | 🐟 | 🍥 | 🍣 |
|---|---|---|---|---|
| fish | (0, **fail**, R) | (0, **fail**, R) | ( 🍣 , **finish**, R) | - |

Our current state returns ( 🍣 , **finish**, R). So we write back 🍣 , go to the **finish** state, and move the head right.

Current State: **finish**



Then we halt since **finish** is a halting state, and return 🍣 !

# bye!

hope you learned something today!
i'll probably post the slides later, and set up a pateron to fund my sushi
addiction.

# License

**Get the source of this theme and the demo presentation from**

http://github.com/famuvie/beamerthemesimple