CSC363 Tutorial 7

Paul “sushi_enjoyer” Zhang

University of Humongus Chungus Amogus

March 3, 2021

Learning objectives this tutorial

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

- Have `turingmachinesimulator.com` bookmarked in Microsoft Edge (or whichever browser you use).
- Feel great, because you will probably never have to touch upon any computability theory again.[1]
- Understand p-reducibility and have a practical example in the back of your mind.
- Fall in love with polynomials. They are so nice!

(no readings this week cuz we didn't really go over anything new)

[1] Oh wait! You have an assignment due on computability theory. ;-; If you want ~~unauthorized aids~~ help, stay for my office hours! :D

More Turing Machines. 🗿

Task: Answer the following:

do you like optimwizing cowde? uwu

Answer: Of course you do.

More Turing Machines. 🗿

Task: Answer the following:

do you like optimwizing cowde? uwu

Answer: Of course you do.

Turing machines are back! :D

Task: Decrypt what this Turing machine is doing (i.e. find the language that this Turing machine accepts). Our alphabet is $\{0, 1\}$.

State	0	1	$\square$
q_0	$(q_1, \square, \mathrm{R})$	reject	accept
q_1	$(q_1, 0, \mathrm{R})$	$(q_1, 1, \mathrm{R})$	$(q_2, \square, \mathrm{L})$
q_2	reject	$(q_3, \square, \mathrm{L})$	reject
q_3	$(q_3, 0, \mathrm{L})$	$(q_3, 1, \mathrm{L})$	$(q_0, \square, \mathrm{R})$

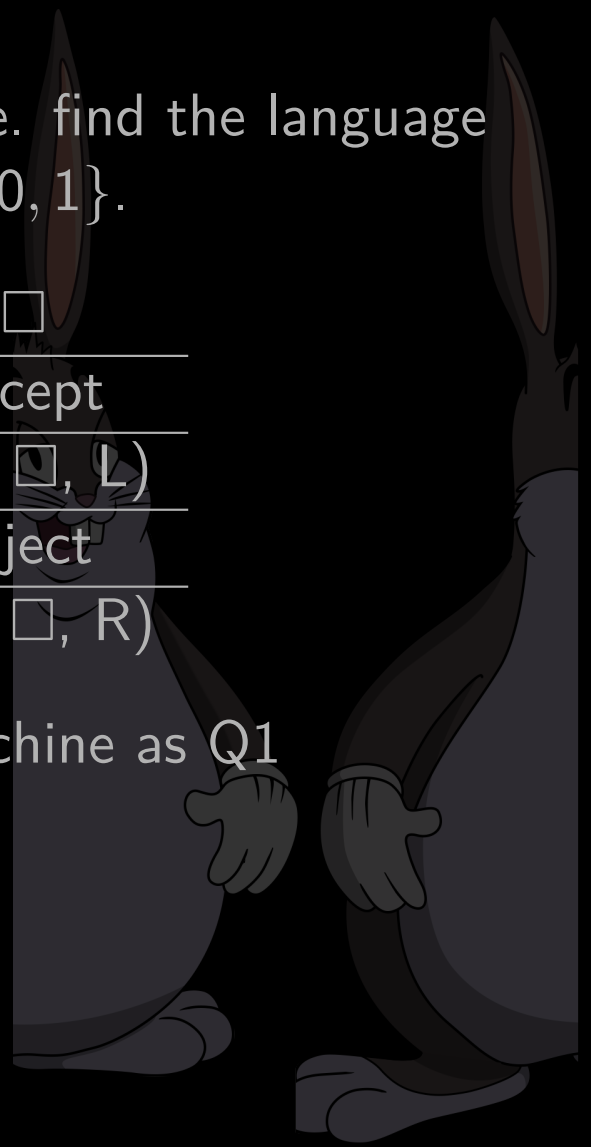
Hint:

Turing machines are back! :D

Task: Decrypt what this Turing machine is doing (i.e. find the language that this Turing machine accepts). Our alphabet is $\{0, 1\}$.

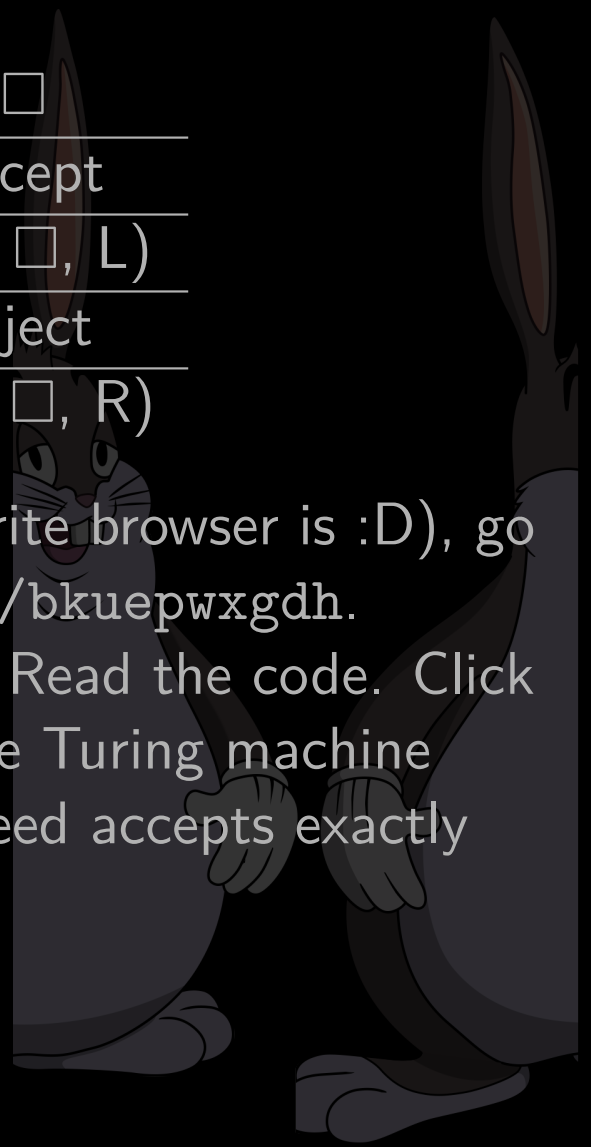
State	0	1	$\square$
q_0	$(q_1, \square, \mathrm{R})$	reject	accept
q_1	$(q_1, 0, \mathrm{R})$	$(q_1, 1, \mathrm{R})$	$(q_2, \square, \mathrm{L})$
q_2	reject	$(q_3, \square, \mathrm{L})$	reject
q_3	$(q_3, 0, \mathrm{L})$	$(q_3, 1, \mathrm{L})$	$(q_0, \square, \mathrm{R})$

Answer: $\{0^n 1^n : n \in \mathbb{N}\}$. (this is the same turing machine as Q1 Assignment 1)



Turing machines are back! :D

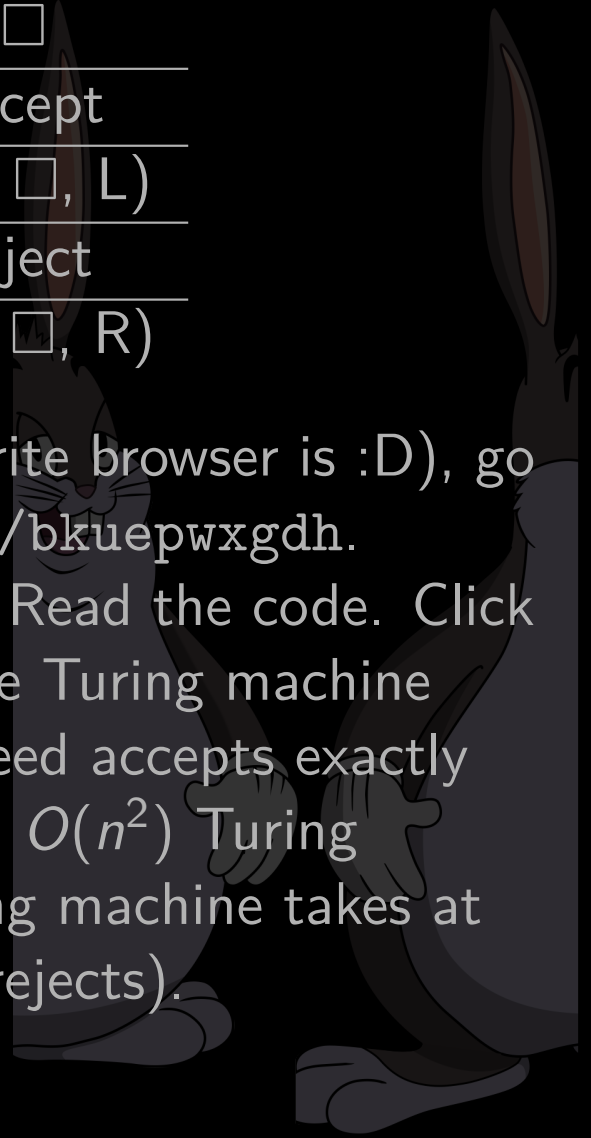
State	0	1	$\square$
q_0	$(q_1, \square, \mathrm{R})$	reject	accept
q_1	$(q_1, 0, \mathrm{R})$	$(q_1, 1, \mathrm{R})$	$(q_2, \square, \mathrm{L})$
q_2	reject	$(q_3, \square, \mathrm{L})$	reject
q_3	$(q_3, 0, \mathrm{L})$	$(q_3, 1, \mathrm{L})$	$(q_0, \square, \mathrm{R})$



Task: Open Microsoft Edge (or whatever your favourite browser is :D), go to `http://turingmachinesimulator.com/shared/bkuepwxgdh`. Bookmark the website. Put on your favourite music. Read the code. Click "Compile". Try a few different inputs and see how the Turing machine runs. Convince yourself that this Turing machine indeed accepts exactly $\{0^n 1^n : n \in \mathbb{N}\}$. Go grab a snack in the meantime.

Turing machines are back! :D

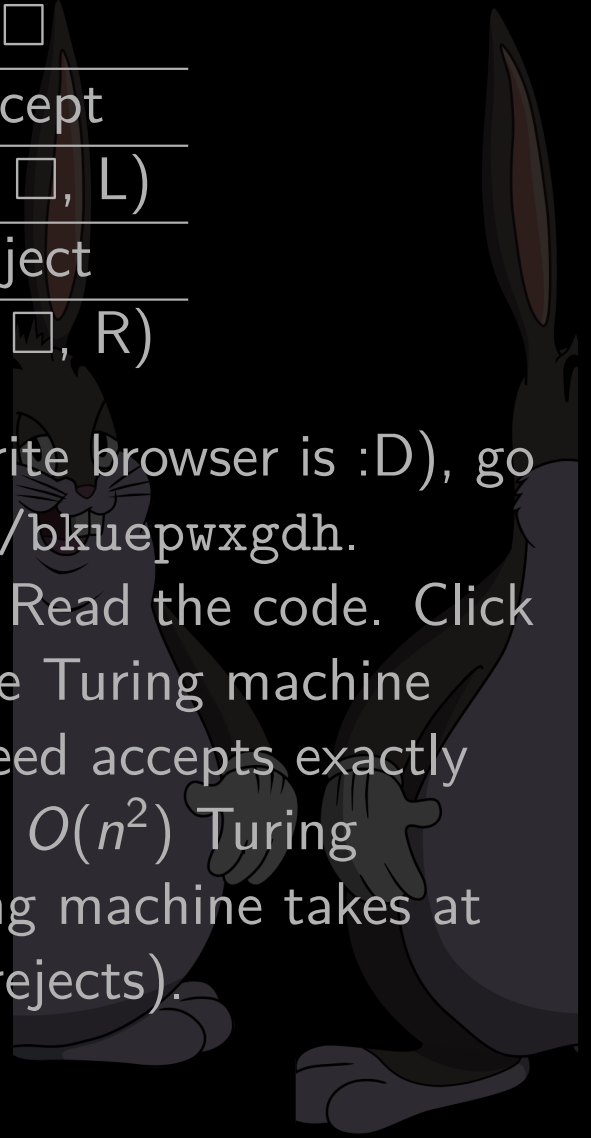
State	0	1	$\square$
q_0	$(q_1, \square, \mathrm{R})$	reject	accept
q_1	$(q_1, 0, \mathrm{R})$	$(q_1, 1, \mathrm{R})$	$(q_2, \square, \mathrm{L})$
q_2	reject	$(q_3, \square, \mathrm{L})$	reject
q_3	$(q_3, 0, \mathrm{L})$	$(q_3, 1, \mathrm{L})$	$(q_0, \square, \mathrm{R})$



Task: Open Microsoft Edge (or whatever your favourite browser is :D), go to `http://turingmachinesimulator.com/shared/bkuepwxgdh`. Bookmark the website. Put on your favourite music. Read the code. Click "Compile". Try a few different inputs and see how the Turing machine runs. Convince yourself that this Turing machine indeed accepts exactly $\{0^n 1^n : n \in \mathbb{N}\}$. Also convince yourself that this is an $O(n^2)$ Turing machine: given an input string of length n, this Turing machine takes at most $O(n^2)$ steps to halt (no matter if it accepts or rejects).

Turing machines are back! :D

State	0	1	□
q_0	$(q_1, \square, \mathrm{R})$	reject	accept
q_1	$(q_1, 0, \mathrm{R})$	$(q_1, 1, \mathrm{R})$	$(q_2, \square, \mathrm{L})$
q_2	reject	$(q_3, \square, \mathrm{L})$	reject
q_3	$(q_3, 0, \mathrm{L})$	$(q_3, 1, \mathrm{L})$	$(q_0, \square, \mathrm{R})$



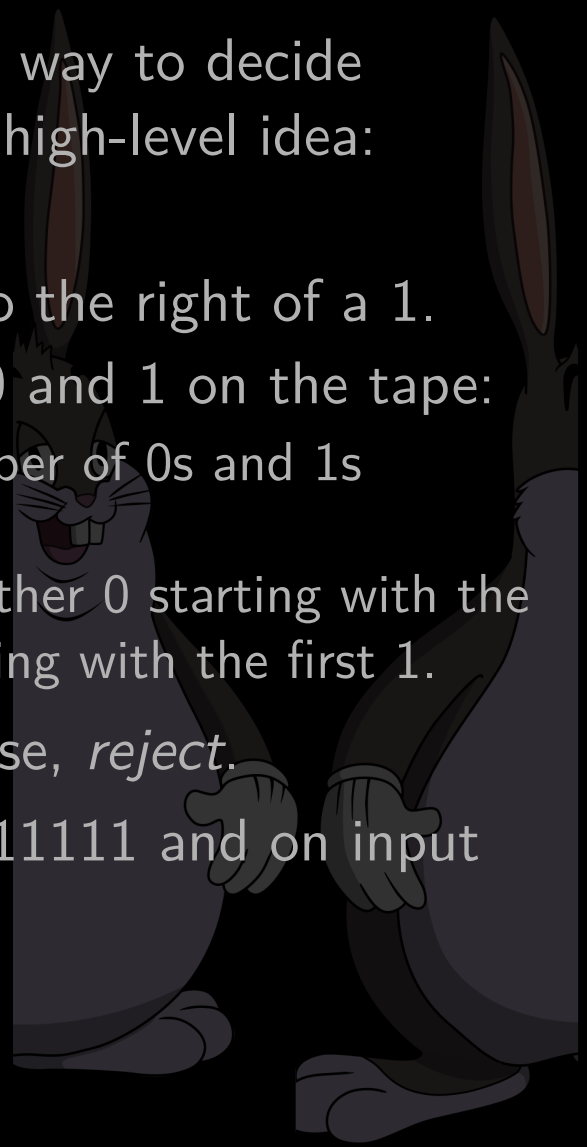
Task: Open Microsoft Edge (or whatever your favourite browser is :D), go to `http://turingmachinesimulator.com/shared/bkuepwxgdh`. Bookmark the website. Put on your favourite music. Read the code. Click "Compile". Try a few different inputs and see how the Turing machine runs. Convince yourself that this Turing machine indeed accepts exactly $\{0^n 1^n : n \in \mathbb{N}\}$. Also convince yourself that this is an $O(n^2)$ Turing machine: given an input string of length n, this Turing machine takes at most $O(n^2)$ steps to halt (no matter if it accepts or rejects).

Turing machines are back! :D

But we can do better, in fact! There is an $O(n \log n)$ way to decide whether a given input is in $0^n1^n : n \in \mathbb{N}$. Here is the high-level idea: On input w, we do the following:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following as long as there is both a 0 and 1 on the tape:
   - 2.1 Scan across the tape, checking if the total number of 0s and 1s remaining is odd. If it is odd, *reject*.
   - 2.2 Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
3. If the tape doesn't have any 0s or 1s, *accept*. Else, *reject*.

Task: Try this procedure (by hand) on input 0000011111 and on input 000001111.

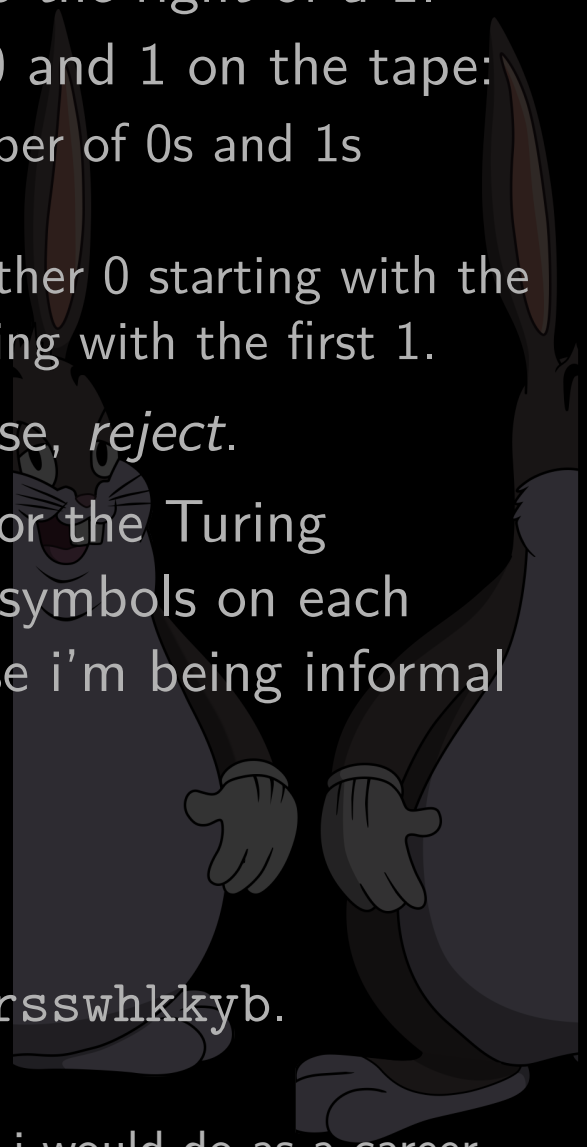


Turing machines are back! :D

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following as long as there is both a 0 and 1 on the tape:
   - 2.1 Scan across the tape, checking if the total number of 0s and 1s remaining is odd. If it is odd, *reject*.
   - 2.2 Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
3. If the tape doesn't have any 0s or 1s, *accept*. Else, *reject*.

The above procedure actually takes $O(n \log n)$ time for the Turing machine, because we cross off at least half of all the symbols on each iteration of 2, so 2 runs at most $\log n$ times (of course i'm being informal here).
But either way, it's more efficient now! yay[2]

You may try it out at
`http://turingmachinesimulator.com/shared/prsswhkkyb`.



[2] yea optimizing turing machine code is definitely something i would do as a career.

i dunno... break time before we get into P-reducibility?

hmm... i'm not sure if youse prefer watching Turing machines execute, over playing with computable and c.e. sets![3]

Trivia time!

What ingredients are in this "drink"?[a]

Answer: i dunno, i haven't yet made the drink when making these slides.

[a] i dunno if this classifies as a drink...

[3] i don't think you want to ever see the symbol φ again.

i dunno... break time before we get into P-reducibility?

hmm... i'm not sure if youse prefer watching Turing machines execute, over playing with computable and c.e. sets![3]

Trivia time!

What ingredients are in this "drink"?[a]

Answer: i dunno, i haven't yet made the drink when making these slides.

[a] i dunno if this classifies as a drink...

[3] i don't think you want to ever see the symbol φ again.

some preface as we're transitioning into a different part of the course, probably

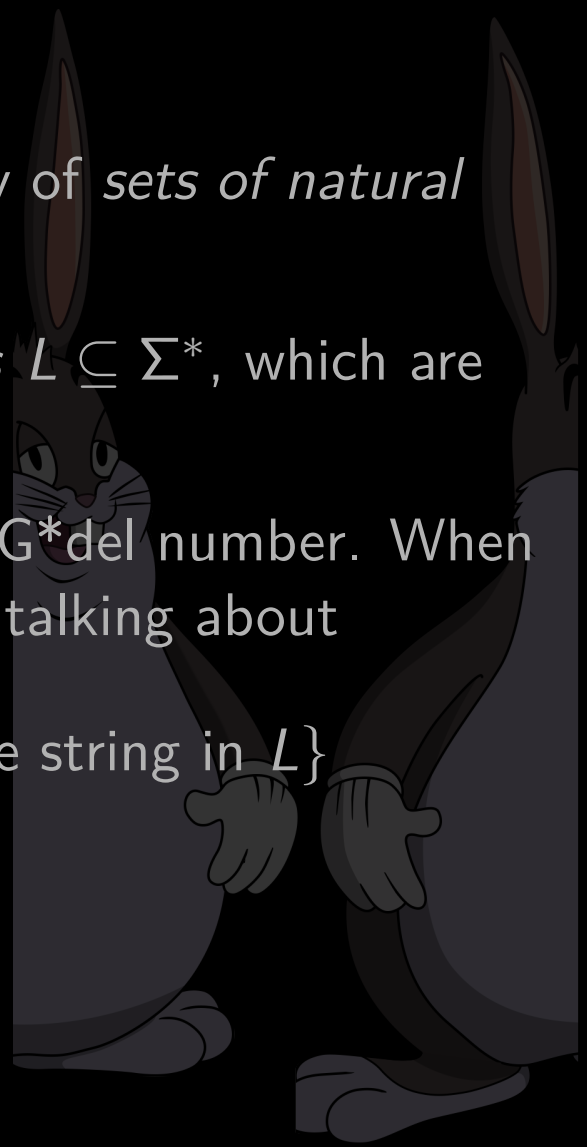
Remember when we were talking about computability of *sets of natural numbers* $A \subseteq \mathbb{N}$?

Now we are talking about computability of *languages* $L \subseteq \Sigma^*$, which are sets of strings.

But really, they're the same thing! Each string has a G*del number. When we talk about a language $L \subseteq \Sigma^*$, we can instead be talking about

$$A = \{e \in \mathbb{N} : e \text{ is the G*dot number for some string in } L\}$$

and vice versa.



some preface as we're transitioning into a different part of the course, probably

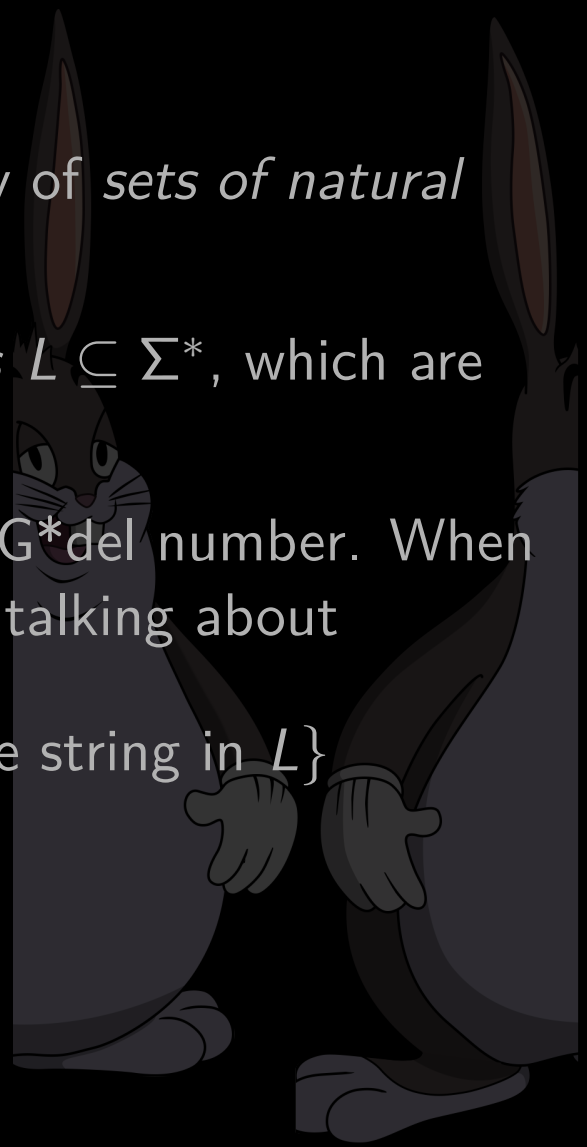
Remember when we were talking about computability of *sets of natural numbers* $A \subseteq \mathbb{N}$?

Now we are talking about computability of *languages* $L \subseteq \Sigma^*$, which are sets of strings.

But really, they're the same thing! Each string has a G*del number. When we talk about a language $L \subseteq \Sigma^*$, we can instead be talking about

$$A = \{e \in \mathbb{N} : e \text{ is the G*dot number for some string in } L\}$$

and vice versa.



P-reducibility :/

Task: What does the P in "P-reducibility" stand for? **Answer:** ~~Psushi~~ paulinomial.

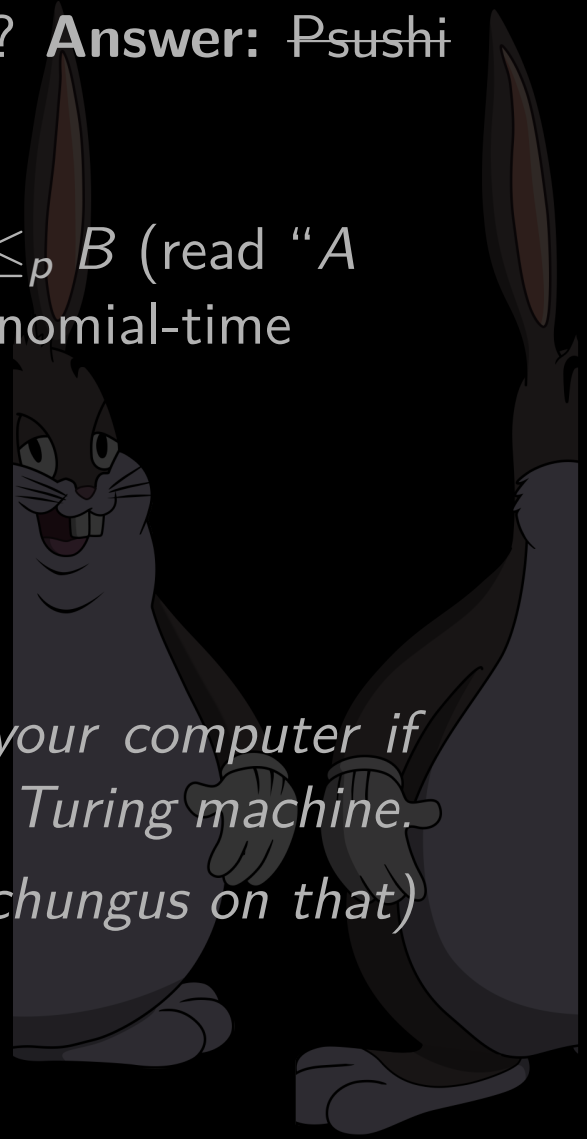
Definition: Let $A, B \subseteq \Sigma^*$ be languages. We say $A \leq_p B$ (read "A polynomially reduces to B") when there exists a polynomial-time computable $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \Leftrightarrow f(x) \in B.$$

Fact:

A language is polynomial-time computable on your computer if and only if it is polynomial-time computable by a Turing machine.

- Chungus, 2077 (don't quote the chungus on that)



P-reducibility :/

Definition: Let $A, B \subseteq \Sigma^*$ be languages. We say $A \leq_p B$ (read "A polynomially reduces to B") when there exists a polynomial-time computable $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \Leftrightarrow f(x) \in B.$$

Theorem

If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.

Task: Prove this.

P-reducibility :/

Definition: Let $A, B \subseteq \Sigma^*$ be languages. We say $A \leq_p B$ (read "A polynomially reduces to B") when there exists a polynomial-time computable $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \Leftrightarrow f(x) \in B.$$

Theorem

If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.

Proof.

Since $A \leq_p B$, let f be a polynomial time computable function such that $x \in A \Leftrightarrow f(x) \in B$. Since $B \leq_p C$, let g be a polynomial time computable function such that $y \in B \Leftrightarrow g(y) \in C$. Then $g \circ f$ is polynomial-time computable (why?), and

$$x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C.$$

By definition, this means $A \leq_p C$. □

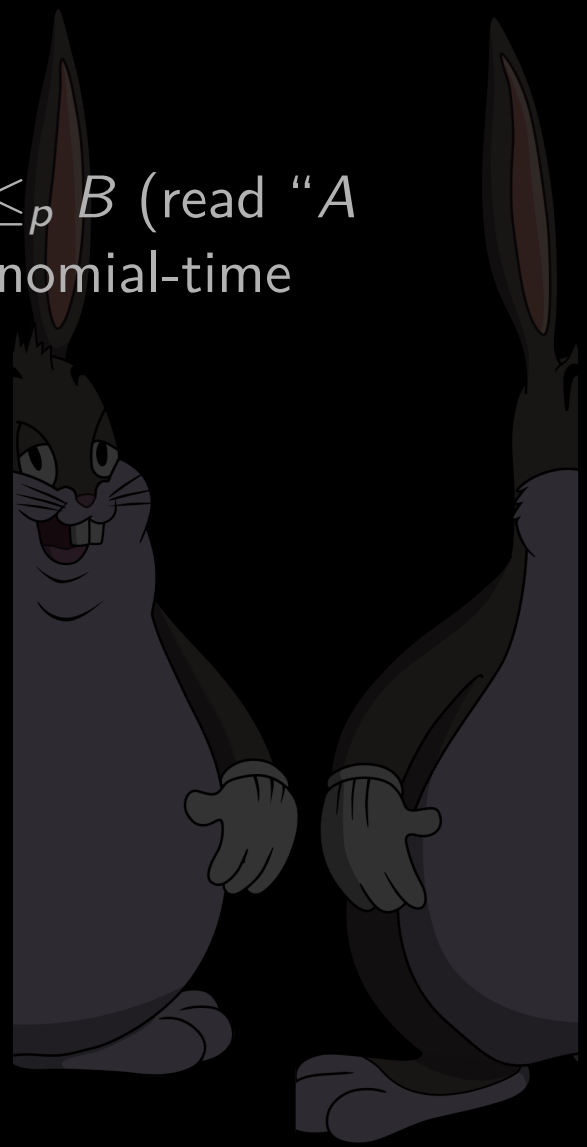
P-reducibility :/

Definition: Let $A, B \subseteq \Sigma^*$ be languages. We say $A \leq_p B$ (read "A polynomially reduces to B") when there exists a polynomial-time computable $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \Leftrightarrow f(x) \in B.$$

Task: Guess what $A \equiv_p B$ means!

Answer: $A \equiv_p B$ when $A \leq_p B$ and $B \leq_p A$.

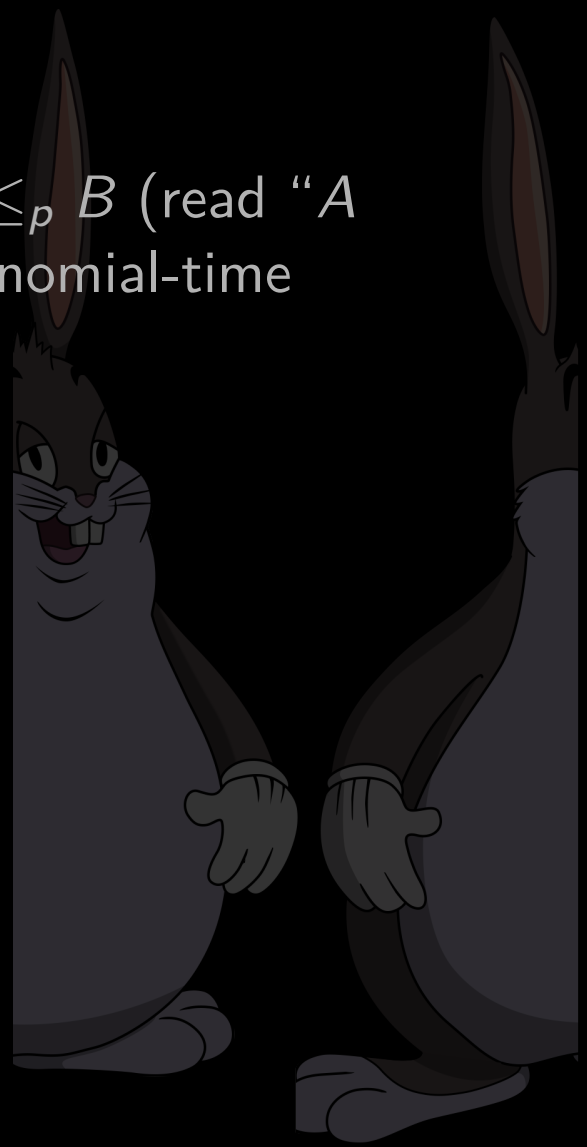


P-reducibility :/

Definition: Let $A, B \subseteq \Sigma^*$ be languages. We say $A \leq_p B$ (read "A polynomially reduces to B") when there exists a polynomial-time computable $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \Leftrightarrow f(x) \in B.$$

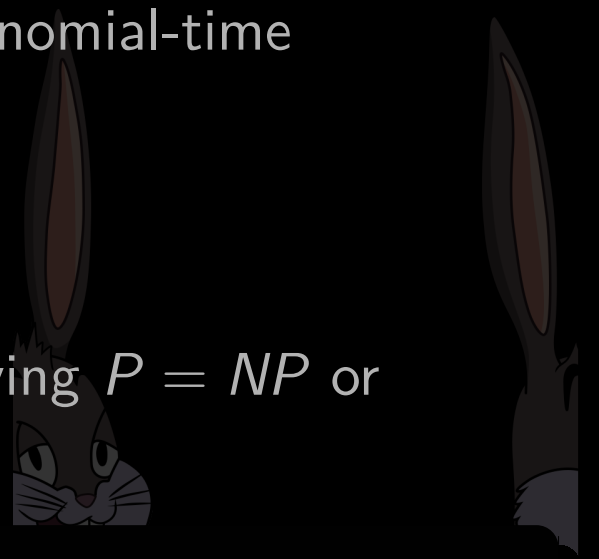
Task: Guess what $A \equiv_p B$ means!
Answer: $A \equiv_p B$ when $A \leq_p B$ and $B \leq_p A$.



P-reducibility :/

Definition: Let $A, B \subseteq \Sigma^*$ be languages. We say $A \leq_p B$ (read "A polynomially reduces to B") when there exists a polynomial-time computable $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \Leftrightarrow f(x) \in B.$$



Task: Prove the following theorem, and then try solving $P = NP$ or something:

Theorem

Let A, B, C be languages. Then

1. $A \equiv_p A$.
2. $A \equiv_p B$ *implies* $B \equiv_p A$.
3. $A \equiv_p B$ *and* $B \equiv_p C$ *implies* $A \equiv_p C$

So $\equiv_p$ is an equivalence relation on the set of languages.

P-reducibility :/

Theorem

Let A, B, C be languages. Then

1. $A \equiv_p A$.
2. $A \equiv_p B$ *implies* $B \equiv_p A$.
3. $A \equiv_p B$ *and* $B \equiv_p C$ *implies* $A \equiv_p C$

So $\equiv_p$ is an equivalence relation on the set of languages.

Proof.

1. The function $f(x) = x$ is very polynomial-time computable, and $x \in A \Leftrightarrow f(x) \in A$, so $A \leq_p A$. Bruh. $A \leq_p A$, so $A \equiv_p A$.
2. $A \equiv_p B$ means $A \leq_p B$ and $B \leq_p A$. Bruh. $B \leq_p A$ and $A \leq_p B$ means $B \equiv_p A$.
3. We have $A \leq_p B$, $B \leq_p A$, $B \leq_p C$, and $C \leq_p B$. By the theorem we've proven earlier, $A \leq_p B$ and $B \leq_p C$ imply $A \leq_p C$. Similarly $C \leq_p B$ and $B \leq_p A$ imply $C \leq_p A$. So $A \equiv_p C$.

□

Homework cheating time!

I'm sure you hate the letter p by now. Why don't we also throw in the letter P?

Theorem

If $A \leq_p B$ and $B \in P$, then $A \in P$.

Task: Prove this. Then submit your proof for this under Question 5 of Assignment 3.
(or you know, the proof is actually in the textbook! page 251.)