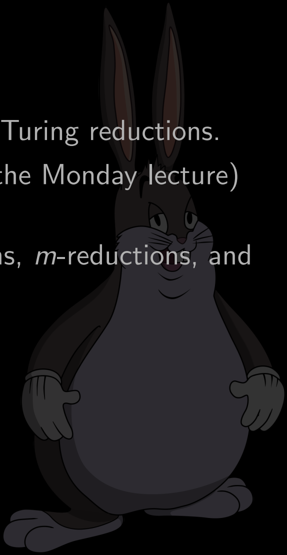
CSC363 Tutorial #4

Turing reductions! (and some assignment feedback)

February 09, 2022

Learning objectives this tutorial

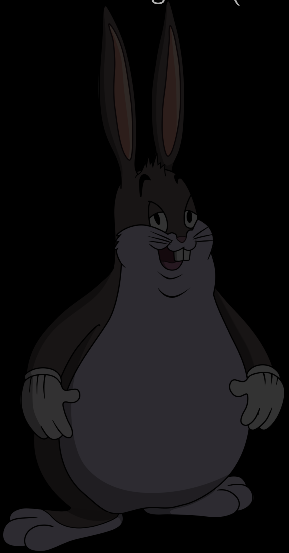
- Review (hopefully, if you remember) Turing reductions.
- Learn (or review, if you've attended the Monday lecture) m-reductions and 1-reductions.
- Distinguish between Turing reductions, m-reductions, and 1-reductions.



Assignment 1 stuff

Assignment 1 feedback has been posted to Pizza. 🍕

Please read through it! Some common mistakes throughout (also appearing on Assignment 2):

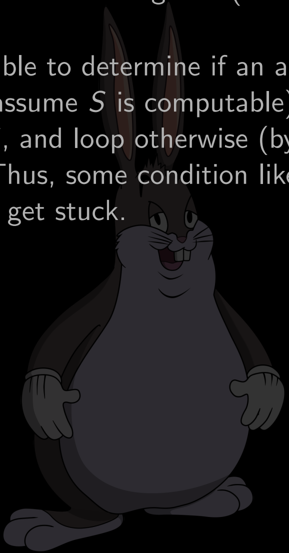


Assignment 1 stuff

Assignment 1 feedback has been posted to Pizza. 🍕

Please read through it! Some common mistakes throughout (also appearing on Assignment 2):

- Given a CE set S, we might not be able to determine if an arbitrary $x \in \mathbb{N}$ is in S or not (unless we can assume S is computable). The best we can do is confirm that $x \in S$, and loop otherwise (by printing out elements of S until we find x). Thus, some condition like "if $x \in S$" might cause your program to get stuck.

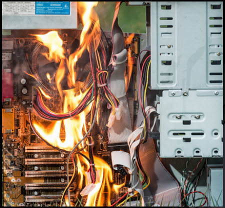


Assignment 1 stuff

Assignment 1 feedback has been posted to Pizza.

Please read through it! Some common mistakes throughout (also appearing on Assignment 2):

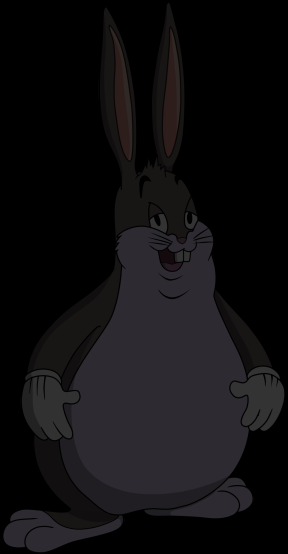
- Given a CE set S, we might not be able to determine if an arbitrary $x \in \mathbb{N}$ is in S or not (unless we can assume S is computable). The best we can do is confirm that $x \in S$, and loop otherwise (by printing out elements of S until we find x). Thus, some condition like "if $x \in S$" might cause your program to get stuck.



- Sometimes, the solution proves a completely different (and maybe more trivial) statement. Make sure to reiterate what you are trying to prove, so that you don't lose track!

Turing reductions!

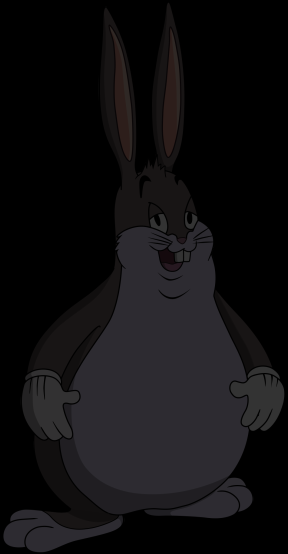
Task: Show that $K = \{x : \varphi_x(x) \text{ halts}\}$ is computable!



Turing reductions!

~~**Task:** Show that $K = \{x : \varphi_x(x) \text{ halts}\}$ is computable!~~

Ans: That's kinda impossible...



Turing reductions!

~~**Task:** Show that $K = \{x : \varphi_x(x) \text{ halts}\}$ is computable!~~

Ans: That's kinda impossible...

But what if some person comes along and gives you this **black box** (or **oracle**) that tells you whether something is in K? This would probably break some law of the universe, but still

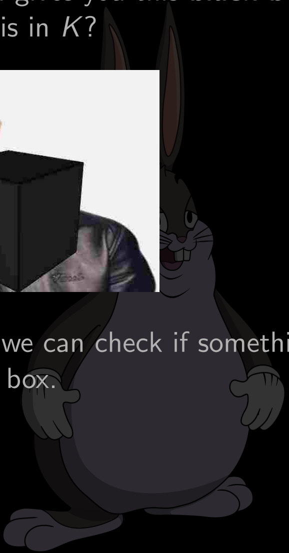
Turing reductions!

But what if some person comes along and gives you this **black box** (or **oracle**) that tells you whether something is in K?

Is K now computable?

Turing reductions!

But what if some person comes along and gives you this **black box** (or **oracle**) that tells you whether something is in K?



Is K now computable? Yes, because now we can check if something is in K or not by just feeding it into this black box.

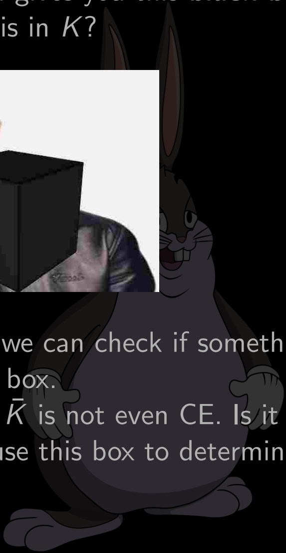
Turing reductions!

But what if some person comes along and gives you this **black box** (or **oracle**) that tells you whether something is in K?

Is K now computable? Yes, because now we can check if something is in K or not by just feeding it into this black box.
What about $\bar{K}$? Without Eminem's help, $\bar{K}$ is not even CE. Is it now computable?

Turing reductions!

But what if some person comes along and gives you this **black box** (or **oracle**) that tells you whether something is in K?



Is K now computable? Yes, because now we can check if something is in K or not by just feeding it into this black box.
What about $\bar{K}$? Without Eminem's help, $\bar{K}$ is not even CE. Is it now computable? Yes, because we can again use this box to determine if something is in $\bar{K}$ (so not in K) or not.

Turing reductions!

If we can compute (the indicator of) K, then we can compute $\bar{K}$. So in some sense, K is *at least as hard to compute* as K: once we are able to compute K, we will also be able to compute $\bar{K}$. We can *reduce* the problem of computing $\bar{K}$ to the problem of computing K.

Turing reductions!

If we can compute (the indicator of) K, then we can compute $\bar{K}$. So in some sense, K is *at least as hard to compute* as K: once we are able to compute K, we will also be able to compute $\bar{K}$. We can *reduce* the problem of computing $\bar{K}$ to the problem of computing K.

Definition: Let $A, B \subseteq \mathbb{N}$ be sets. We say that A **Turing reduces to** B, written $A \leq_T B$, if we can compute A given a black box for B.

Turing reductions!

If we can compute (the indicator of) K, then we can compute $\bar{K}$. So in some sense, K is *at least as hard to compute* as K: once we are able to compute K, we will also be able to compute $\bar{K}$. We can *reduce* the problem of computing $\bar{K}$ to the problem of computing K.

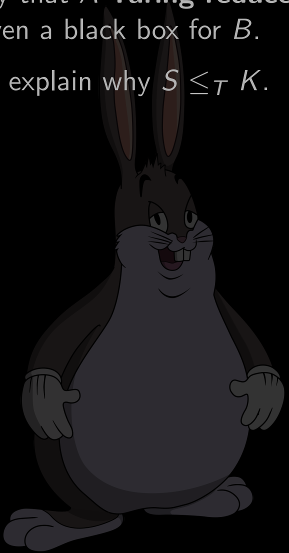
Definition: Let $A, B \subseteq \mathbb{N}$ be sets. We say that A **Turing reduces to** B, written $A \leq_T B$, if we can compute A given a black box for B.

You may think of $A \leq_T B$ as saying "A is less difficult than B", in that we can reduce the problem of computing A into the problem of computing B.

Turing reductions!

Definition: Let $A, B \subseteq \mathbb{N}$ be sets. We say that A **Turing reduces to** B, written $A \leq_T B$, if we can compute A given a black box for B.

Task: Let S be a computable set. Briefly explain why $S \leq_T K$.



Turing reductions!

Definition: Let $A, B \subseteq \mathbb{N}$ be sets. We say that A **Turing reduces to** B, written $A \leq_T B$, if we can compute A given a black box for B.

Task: Let S be a computable set. Briefly explain why $S \leq_T K$.
Ans: Since S is computable, given a black box for K, we can just throw away the black box and compute S directly!

Turing reductions!

Definition: Let $A, B \subseteq \mathbb{N}$ be sets. We say that A **Turing reduces to** B, written $A \leq_T B$, if we can compute A given a black box for B.

Task: Let $K = \{x : \phi_x(x) \text{ halts}\}$, and

$$H = \{(x, e) : \phi_e(x) \text{ halts}\}.$$

Show that $K \leq_T H$.

Turing reductions!

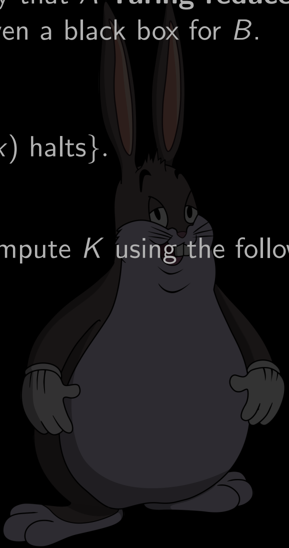
Definition: Let $A, B \subseteq \mathbb{N}$ be sets. We say that A **Turing reduces to** B, written $A \leq_T B$, if we can compute A given a black box for B.

Task: Let $K = \{x : \phi_x(x) \text{ halts}\}$, and

$$H = \{(x, e) : \phi_e(x) \text{ halts}\}.$$

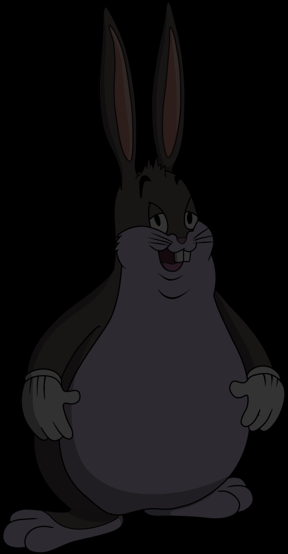
Show that $K \leq_T H$.
Ans: Given a black box for H, we can compute K using the following procedure:

```
def is_in_K(x):
  if (x, x) in H:
    return True
  else: return False
```



Turing reductions!

Task: Let $K = \{x : \phi_x(x) \text{ halts}\}$, and $H = \{(x, e) : \phi_e(x) \text{ halts}\}$. Show that $H \leq_T K$. This is a bit trickier!



Turing reductions!

Task: Let $K = \{x : \phi_x(x) \text{ halts}\}$, and $H = \{(x, e) : \phi_e(x) \text{ halts}\}$. Show that $H \leq_T K$. This is a bit trickier!

Ans: Given a black box for K, we can compute H using the following procedure:

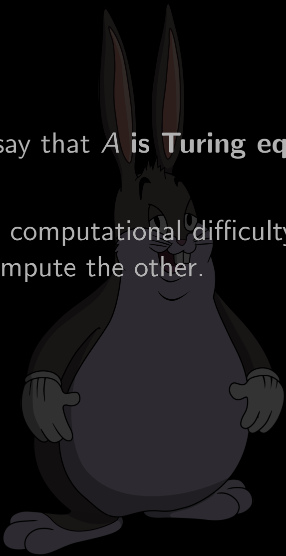
```
def is_in_H(x, e):
  Construct the TM M that does the following:
    M(y):
      (ignore y)
      Run the eth Turing machine on x
      Return if it halts
  Let z be the Turing Machine # of M
  if z in K:
    return True
  else: return False
```

Notice: We construct M, but we don't actually run it! Running M might result in a loop.

Turing reductions!

Definition: If $A \leq_T B$ and $B \leq_T A$, we say that A **is Turing equivalent to** B, and write $A =_T B$.

In some sense, this says A is equivalent in computational difficulty to B: if we can compute one, then we can also compute the other.

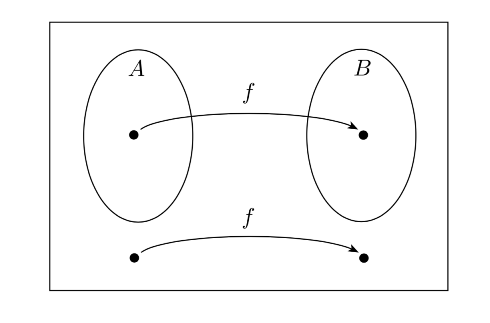


m-reductions...

We'll introduce another reduction mechanism, called an *m*-reduction.

Definition: Let A, B be sets. We say that $A \leq_m B$ if there exists a *computable* function f such that

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



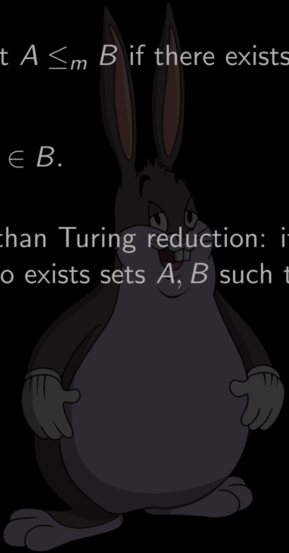
stolen from `https://liyanxu.blog/2019/05/06/mapping-reducibility-turing-reducibility-kolmogorov-complexity/`

m-reductions...

Definition: Let A, B be sets. We say that $A \leq_m B$ if there exists a *computable* function f such that

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

It turns out that *m*-reduction is stronger than Turing reduction: if $A \leq_m B$, then $A \leq_T B$. However, there do exists sets A, B such that $A \leq_T B$ but not $A \leq_m B$.



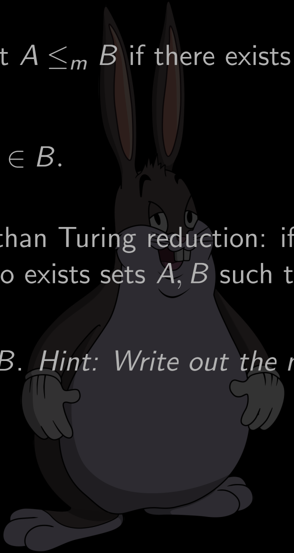
m-reductions...

Definition: Let A, B be sets. We say that $A \leq_m B$ if there exists a *computable* function f such that

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

It turns out that *m*-reduction is stronger than Turing reduction: if $A \leq_m B$, then $A \leq_T B$. However, there do exists sets A, B such that $A \leq_T B$ but not $A \leq_m B$.

Task: Show that if $A \leq_m B$, then $A \leq_T B$. *Hint: Write out the meaning of each of those definitions.*



m-reductions...

Definition: Let A, B be sets. We say that $A \leq_m B$ if there exists a *computable* function f such that

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

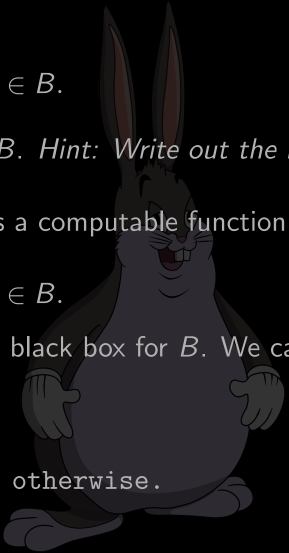
Task: Show that if $A \leq_m B$, then $A \leq_T B$. *Hint: Write out the meaning of each of those definitions.*

Ans: Suppose $A \leq_m B$. Then there exists a computable function f such that

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

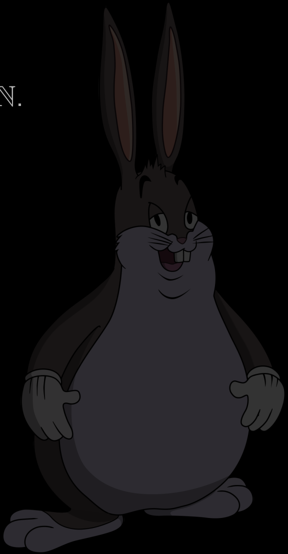
To show $A \leq_T B$, suppose we are given a black box for B. We can compute A as follows:

```
def is_in_A(x):
  return True if f(x) in B, False otherwise.
```



m-reductions...

Task: Show that $\emptyset \leq_T \mathbb{N}$, but not $\emptyset \leq_m \mathbb{N}$.

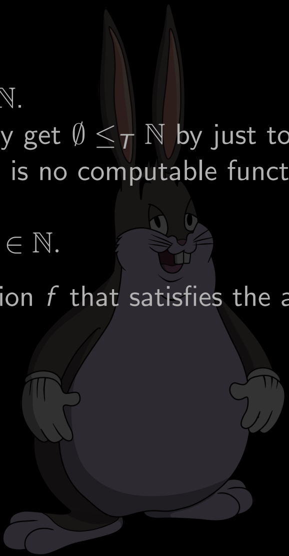


m-reductions...

Task: Show that $\emptyset \leq_T \mathbb{N}$, but not $\emptyset \leq_m \mathbb{N}$.
Ans: $\emptyset$ is computable, so we automatically get $\emptyset \leq_T \mathbb{N}$ by just tossing away the black box for $\mathbb{N}$. However, there is no computable function f such that

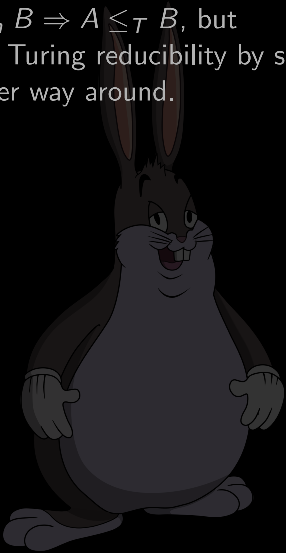
$$x \in \emptyset \Leftrightarrow f(x) \in \mathbb{N}.$$

This is because there isn't even any function f that satisfies the above, regardless of computability of f! (Why?)



m-reductions...

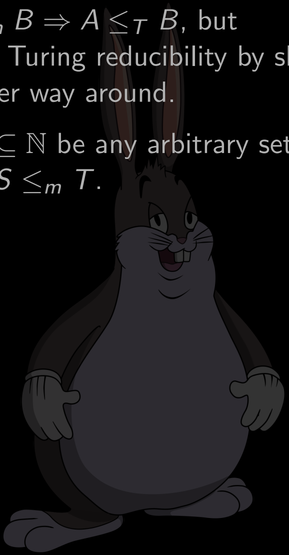
So what we have just shown is that $A \leq_m B \Rightarrow A \leq_T B$, but $A \leq_T B \not\Rightarrow A \leq_m B$. Thus, we may show Turing reducibility by showing m-reducibility, but not necessarily the other way around.



m-reductions...

So what we have just shown is that $A \leq_m B \Rightarrow A \leq_T B$, but $A \leq_T B \not\Rightarrow A \leq_m B$. Thus, we may show Turing reducibility by showing m-reducibility, but not necessarily the other way around.

Task: Let $S \subseteq \mathbb{N}$ be computable, and $T \subseteq \mathbb{N}$ be any arbitrary set satisfying $T \neq \emptyset$ and $T \neq \mathbb{N}$. Show that $S \leq_m T$.



m-reductions...

So what we have just shown is that $A \leq_m B \Rightarrow A \leq_T B$, but $A \leq_T B \not\Rightarrow A \leq_m B$. Thus, we may show Turing reducibility by showing m-reducibility, but not necessarily the other way around.

Task: Let $S \subseteq \mathbb{N}$ be computable, and $T \subseteq \mathbb{N}$ be any arbitrary set satisfying $T \neq \emptyset$ and $T \neq \mathbb{N}$. Show that $S \leq_m T$.

Ans: Since $T \neq \emptyset$, there is some $p \in T$. Since $T \neq \mathbb{N}$, there is some $q \in \mathbb{N}, q \notin T$. Define f by

$$f(x) = \begin{cases} p & x \in S \\ q & x \notin S. \end{cases}$$

Since S is computable, so is f. Furthermore, $x \in S \Leftrightarrow f(x) \in T$.