
Hem-Ogi 2.1: One Way Functions GEM

Group 2:

Benjamin Van Durme
Pin Lu
Ross Messing
Shivashankar Balu
Tanushree Mittal

Definitions

■ One Way Functions :

- A function that is easy to compute and hard to invert
- There are no known functions that have been proven to be one way
 - Much like we don't know if $P=NP$...
- In general, we want to say that f is one way if :

$$f(x) = y$$

can be computed in polynomial time, but its inverse:

$$g(y) = x$$

cannot be computed in polynomial time

Definition 2.1 : Honest

■ Honest :

- We say a function f , is honest if

$$(\exists \text{polynomial } q)(\forall y \in \text{range}(f))(\exists x) \\ [|x| \leq q(|y|) \wedge f(x) = y]$$

Honesty says that for each element x where $f(x)$ is defined, the length of the result, y , is at most polynomially longer than the length of x

Why do we need this?

We are trying to prevent “cheating” by allowing someone to claim that the inverse is not “easy” because it takes more than polynomial time to write the output

Example of Honesty

- Consider the function $f(x) = 1^{\lceil \log \log \log (\max\{|x|, 4\}) \rceil}$
 - The output is so short relative to the input that it will take triple exponential time to write the inverse
 - Thus, f is polynomial time computable, but not polynomial time invertible
 - naively, this would **seem** to be a one way function
 - However, the “non-easy” invertibility of f is only due to a “cheap trick” where we’ve forced the inversion function to spend all of its time simply writing the result
 - That’s not fair!
 - We preclude these types of functions by requiring all those that are “truly” one way to be HONEST

Definition 2.2 : Poly time invertible

- A function f is polynomial-time invertible if there is a polynomial-time computable function g such that :

$$(\forall y \in \text{range}(f))$$

$$[(y \in \text{domain}(g)) \wedge (y \in \text{domain}(f)) \wedge (g(y) = y)]$$

- Which is just to say that f can be “reversed engineered” in a somewhat similar amount of time

Definition 2.3 : One way

- A function f is one way if :
 - f is polynomial-time computable, and
 - f is not polynomial time invertible, and
 - f is honest

Definition 2.4 : One to one

- A function $f : \Sigma^* \rightarrow \Sigma^*$ is one to one if:
$$(\forall y \in \Sigma^*) [|\{ x \mid f(x) = y \}| \leq 1]$$

Theorem 2.5

1. One-way functions exist iff $P \neq NP$
 2. One-to-one one-way functions exist iff $P \neq UP$
- We will be spending the rest of class proving these two points. The proof for the second point is a modification of the first, so pay close attention to the details, as we'll be glossing over some things the second time around.

Proof : One way functions exist iff $P \neq NP$

- Breaking this up, we get:

- *if* :

- $P \neq NP \Rightarrow$ one way functions exist

- *only if* :

- One way functions exist $\Rightarrow P \neq NP$

We will now tackle this in two stages, proving each direction as a separate sub-proof

Proof : $P \neq NP \Rightarrow$ one way functions exist

- We are going to assume “ P is not equal to NP ”
- Now imagine a non-deterministic, polynomial-time computable Turing machine (NPTM) \mathcal{N} , where $\mathcal{L}(\mathcal{N}) = \mathcal{A}$
- Let \mathcal{A} be in $NP-P$
 - P does not equal NP , so this set exists

Proof... the function f

- Let $\langle \cdot, \cdot \rangle$ be our standard pairing function
 - For reference, this is polynomial time computable and invertible
- Now, consider an arbitrary function f that takes as input the paired values $\langle x, w \rangle$

$$f(\langle x, w \rangle) = \begin{cases} 0x & w \text{ is an accepting path in } N(x) \\ 1x & \text{otherwise} \end{cases}$$

- f is polynomial time computable
 - It just has to verify that w is an accepting path for x
- f is also honest
 - Why?

Proof... f is honest

- When w represents an accepting path of an NPTM when run on x , then we know that no path in such a machine can be longer than some polynomial $p(|x|)$
- When w does not represent such a path, then we have no *a priori* knowledge as to the length of w ; indeed, $|w|$ could be super-exponential in the length of x
 - This could spell trouble for f 's honesty
- However, **all** values of w such that $|w| > p(|x|)$ will lead f to output $1x$
- Note that since we can only define f if we already have some machine \mathcal{N} , then we “get to” set the polynomial bound used to keep f honest with full knowledge as to the polynomial bound constraining \mathcal{N}
 - While both polynomials must be with respect to essentially the same string (x vs $1x$), we have the right to make the honesty bound polynomially larger than the bound on \mathcal{N}
 - This means that there is at least one value of w that will be “too long” to be an accepting path, but is still “short enough” to allow f to fulfill the honesty condition
 - As we only need *at least* one honest preimage for every output, then this solves our concern about w
 - This is a form of *out-flanking*
- So, whether or not w is an accepting path, $\langle x, w \rangle$ is still just a polynomial expansion away from $x \cdot w$, which is itself polynomial in length with respect to x (specifically, this is true for *at least* one w for each output of f)
- The **range** of f is : $\{ 0x, 1x \}$
 - $|x| + |0| = |x| + 1$
- So, given these facts, is it true that $|\langle x, w \rangle| \leq q(|0x|)$?
 - Of course it is
- Therefore, f is honest

Proof... assume f can be easily inverted

- Now we assume f is polynomial time invertible via some function g
- Given this function g , we can use it to construct a Deterministic PTM \mathcal{M} , such that $\mathcal{L}(\mathcal{M}) = \mathcal{A}$
 - Earlier we said that $\mathcal{L}(\mathcal{N}) = \mathcal{A}$

Proof... the machine \mathcal{M}

- The machine \mathcal{M} on arbitrary input x :
 - Check if $0x$ is in the domain of g
 - If not, then reject
 - Otherwise
 - Call $g(0x)$, which will return some value $\langle xw \rangle$
 - Test whether w is an accepting path of $\mathcal{N}(x)$
 - If yes, then accept
 - Otherwise reject

Proof... what does \mathcal{M} buy us?

- With \mathcal{M} in hand, we can conclude that \mathcal{A} must belong to P
 - because we just gave a DPTM that accepts \mathcal{A}
- But wait:
 - Earlier we assumed that \mathcal{A} was not in P
 - We did this by stating that \mathcal{A} was in $NP-P$
 - \mathcal{A} cannot be in both P and $NP-P$
 - **Contradiction**

Proof... what went wrong?

- The existence of \mathcal{M} was entirely based on our assumption that g exists
- Therefore f must actually **not** be polynomial time invertible
- This makes f a one way function by our definition
- Therefore:

$P \neq NP \Rightarrow$ one way functions exist

Proof: One way functions exist $\Rightarrow P \neq NP$

- We now prove the other direction.
- Consider the following language:

$\mathcal{L} =$

$$\{ \langle z, \text{pre} \rangle \mid (\exists y) [|y| + |\text{pre}| \leq p(|z|) \wedge f(\text{pre} \cdot y) = z] \}$$

- We claim that \mathcal{L} is clearly in NP.
 - Why ?

Proof... $\mathcal{L} \in \text{NP}$

- Imagine a NPTM \mathcal{N} , such that on arbitrary input $\langle z, \text{pre} \rangle$:

- For each string $y \in \Sigma^*$, where $|y| + |\text{pre}| \leq p(|z|)$



- Check if $f(\text{pre} \cdot y) = z$ ← **Polynomial time**

$2^{p(|z|)}$ number of y 's,
but can be “guessed”
in parallel

Non-deterministic poly time

Proof... assume $\mathcal{L} \in P$

- Now that we've shown \mathcal{L} to be in NP, we are going to *assume* that $\mathcal{L} \in P$
 - Obviously we are setting ourselves up for a contradiction
- We are going to use this assumption to construct a machine that will allow us to “easily” invert f , via a prefix search
- First, let \mathcal{M} be a DPTM that accepts \mathcal{L}
 - Note that we don't care how it actually works, we just need to know that it exists
 - Using \mathcal{M} , we can construct a new machine \mathcal{M}' that, on arbitrary input z , does the following...

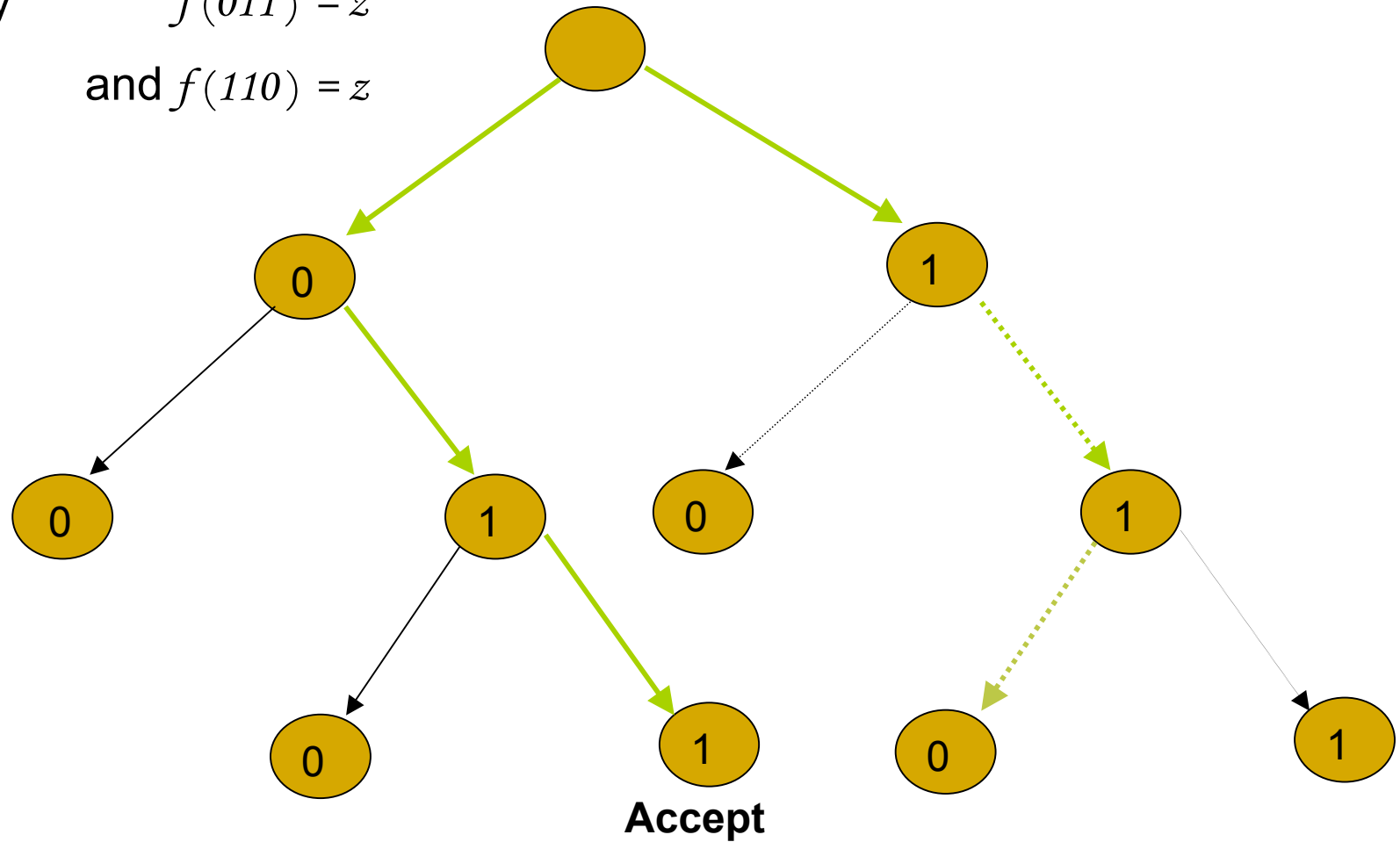
Proof... the machine \mathcal{M}'

- Simulate \mathcal{M} on $\langle z, \varepsilon \rangle$:
 - if \mathcal{M} rejects, then \mathcal{M}' rejects
 - if $f(\varepsilon) = z$, then \mathcal{M}' accepts
- Otherwise, let $\chi = \varepsilon$
- Simulate \mathcal{M} on $\langle z, \chi 0 \rangle$:
 - if it accepts
 - let $\chi = \chi 0$
 - if $f(\chi) = z$ then \mathcal{M}' accepts
 - else repeat 3
 - else goto 4
- Simulate \mathcal{M} on $\langle z, \chi 1 \rangle$:
 - if it accepts
 - let $\chi = \chi 1$
 - if $f(\chi) = z$ then \mathcal{M}' accepts
 - else goto 3
 - else goto 3

Note that we do not actually need to simulate \mathcal{M} at this step, nor will we ever encounter the final goto (Can you tell why?)

Example

Lets say $f(011) = z$
and $f(110) = z$



Proof... we find a contradiction

- With the machine \mathcal{M}' in hand, we can “easily” invert f
 - \mathcal{M}' will find one bit of information with each step
 - Because f is honest, the inverse of $f(z)$ has to be polynomial with respect to z
 - Therefore, \mathcal{M}' will find the inverse of $f(z)$ in polynomial time, bit by bit
- However, if we can easily invert f , then f can't possibly be one-way
- f being a one-way function was one of our basic assumptions
 - **CONTRADICTION**

Proof... the fallout

- As f has to remain one-way, \mathcal{M}' must not really exist
- \mathcal{M}' existed by virtue of \mathcal{M}
- \mathcal{M} existed because we assumed $\mathcal{L} \in \mathsf{P}$
- Therefore, as \mathcal{L} **is** in NP , but now cannot be in P , then it must be in $\mathsf{NP-P}$
- We have achieved our goal:

One way functions exist $\Rightarrow P \neq NP$

Proof : One way functions exist iff $P \neq NP$

✓ $P \neq NP \Rightarrow$ one way functions exist

✓ One way functions exist $\Rightarrow P \neq NP$

■ Thus, we have just proven part 1 of Thm 2.5

Proof of Second Point: One-to-one one way functions exist iff $P \neq UP$

- Before we tackle this proof, what is UP ?

UP

- It is the class of problems that have a unique witness.
- A language \mathcal{L} is in UP if
 - If an NP machine \mathcal{N} accepts an input x in language \mathcal{L}
 - And, for all such input x , the computation $\mathcal{N}(x)$ has at most one accepting path
- Formally:
$$\text{UP} = \{ \mathcal{L} \mid \text{there is a NPTM } \mathcal{N} \text{ such that } \mathcal{L} = \mathcal{L}(\mathcal{N}) \text{ and, for all } x, \mathcal{N}(x) \text{ has at most one accepting path} \}$$

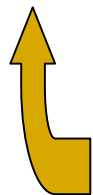
Proof... break up the bi-conditional

- As before, we will tackle each direction separately
 - *if* :
 $P \neq UP \Rightarrow$ one-to-one one way functions exist
 - *only if* :
One-to-one one way functions exist $\Rightarrow P \neq UP$

Proof : $P \neq UP \Rightarrow$ one-to-one one way functions exist

- Let \mathcal{A} be a language in UP-P
- Imagine a NPTM \mathcal{N} , where $\mathcal{L}(\mathcal{N}) = \mathcal{A}$
- Consider the revised function f :

$$f(\langle x, w \rangle) = \begin{cases} 0x & w \text{ is an accepting path in } N(x) \\ 1\langle x, w \rangle & \text{otherwise} \end{cases}$$



Note how we've changed f

Proof...

- Our revised f is now clearly one-to-one
 - Since the non-accepting witnesses give unique results
 - There is only one accepting path, thus we do not need to “rig” 0_χ to make it unique
- Just as in the last proof, we can again try to assume there is a polynomial time inverse function g
- Using g , we can construct a similar DPTM \mathcal{M}
 - The one-to-one-ness of f does not change the character of the machine

Proof... the machine \mathcal{M}

- The machine \mathcal{M} on arbitrary input x :
 - Check if $0x$ is in the domain of g
 - If not, then reject
 - Otherwise
 - Call $g(0x)$, which will return some value $\langle xw \rangle$
 - Test whether w is an accepting path of $\mathcal{N}(x)$
 - If yes, then accept
 - Otherwise reject

Proof... what does \mathcal{M} buy us?

- With \mathcal{M} in hand, we can conclude that \mathcal{A} must belong to P
 - because we just gave a DPTM that accepts \mathcal{A}
- But wait:
 - Earlier we assumed that \mathcal{A} was not in P
 - We did this by stating that \mathcal{A} was in $UP-P$
 - \mathcal{A} cannot be in both P and $UP-P$
 - **Contradiction**

Proof : One-to-one one way functions exist $\Rightarrow P \neq UP$

- Recall what we did for $P \neq NP$
- Consider the language:

$\mathcal{L} =$

$$\{ \langle z, \text{pre} \rangle \mid (\exists y) [|y| + |\text{pre}| \leq p(|z|) \wedge f(\text{pre} \cdot y) = z] \}$$

- \mathcal{L} is obviously in UP if f is one-to-one
- We can try to claim that it is in P
- But this will fail to the same prefix search technique that we explained earlier for $P \neq NP$
 - One distinction: there will never be a case where both x_0 and x_1 could be accepted at the same level, as the prefix at every intermediate length must be unique since f is one-to-one

Proof... contradiction

- As \mathcal{L} is in UP, but cannot be in P, then it must be the case that $P \neq UP$
- This gives us our result:
One-to-one one way functions exist $\Rightarrow P \neq UP$
- We have (quickly) shown both directions of the bi-conditional
- Thus we've proven point 2 of Thm. 2.5

Conclusion

- We have provided an introduction to the notion of (one-to-one), one way functions
- Key points to take away:
 - There are no known one-way functions
 - Their existence is tied to whether $P=NP$
 - In the case of 1-to-one one way functions, their existence is tied to a more strongly regulated version of NP, the class UP
- In the next lecture we will expand this last statement to cover a constant bounded version of UP

Hem-Ogi 2.2 :

Unambiguous One Way Functions exist
 \Leftrightarrow bounded ambiguity one way
functions exist

Group 2:

Benjamin Van Durme

Pin Lu

Ross Messing

Shivashankar Balu

Tanushree Mittal

Last lecture

- One Way Functions
 - One way functions exist , $P \neq NP$
 - One-to-one one-way functions exist , $P \neq UP$

Today's lecture

- We will be expanding our last claim made previously dealing with one-to-one, one way functions and the class UP
 - Extend this statement to handle a slightly broader class
- First need cover new definitions:
 - k -to-one / bounded ambiguity
 - $UP_{\leq k}$
- Then onto an inductive proof
- Any time left will be spent going over definitions required for the final section of Chapter 2
- If we *still* have time left, I will speak on the issues raised by Lane from Monday's lecture

Definition 2.6: k -to-one functions

- A function f is k -to-one :

$$(\forall y \in \text{range}(f)) [|\{x \mid f(x) = y\}| \leq k]$$

- If there is a $k \in \{1, 2, 3, \dots\}$ such that f is k -to-one, then we say that f is of *bounded ambiguity*
 - Special case: when $k=1$ then f is said to be *unambiguous*

Thm 2.7: Unambiguous one way functions exist \Leftrightarrow bounded ambiguity one way functions exist

■ Breaking this up, we get:

□ *if:*

Bounded ambiguity one way functions exist \Rightarrow
Unambiguous one way functions exist

□ *only if:*

Unambiguous one way functions exist \Rightarrow Bounded
ambiguity one way functions exist

Proof: Unambiguous one way functions exist
 \Rightarrow Bounded ambiguity one way functions exist

- This turns out to be trivial
- Unambiguous one way functions are simply a special case of bounded ambiguity one way functions :

$$(\forall y) \in \text{range}(f) \parallel [\{x \mid f(x) = y\} \leq k]$$

□ When $k=1$, then f is a one-to-one (unambiguous) function

- Thus we've (quickly) shown the “only if” direction

Proof: Bounded ambiguity function exist \Rightarrow
Unambiguous one way functions exist

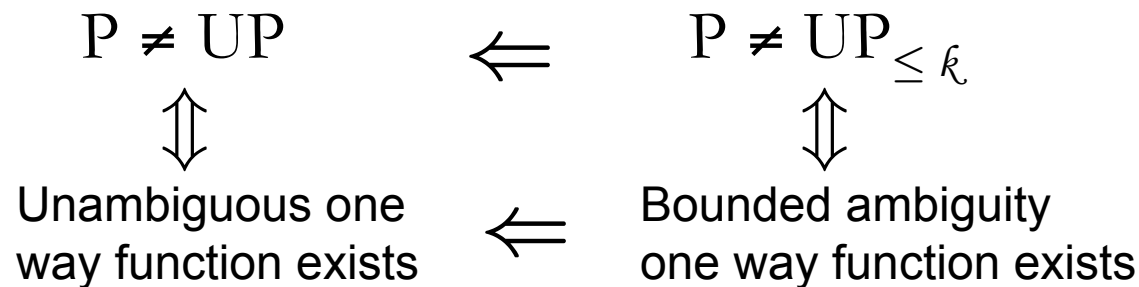
- Before beginning with the other half of the bi-conditional, we should make sure we understand the class of languages $UP_{\leq k}$

$UP_{\leq k}$

- A language \mathcal{L} is in $UP_{\leq k}$ if there is a NPTM \mathcal{N} such that:
 - $(\forall x \in \mathcal{L}) [\mathcal{N}(x) \text{ has at least one and at most } k \text{ accepting paths }]$
 - $(\forall x \in \mathcal{L}^c) [\mathcal{N}(x) \text{ has no accepting paths }]$
- Similar to UP, only rather than the associated machine being restricted to having a unique accepting path, in this case there may be up to some constant number of such paths

Proof... strategy for indirect proof

- Proving the “if” will be done using an indirect path
- Observe the following diagram:



- We implicitly use the second point of Thm 2.5
- The bounded version of this point is analogous, and we thus will rely on it as a “Fact”
- From there we will use an inductive proof to show that $P=UP \Rightarrow P=UP_{\leq k}$
- At this point we rely on the contrapositive of this statement to complete the indirect attack

Proof...

■ Fact 2.9

For each $k \geq 2$, k -to-one one-way functions exist $\Leftrightarrow P \neq UP_{\leq k}$

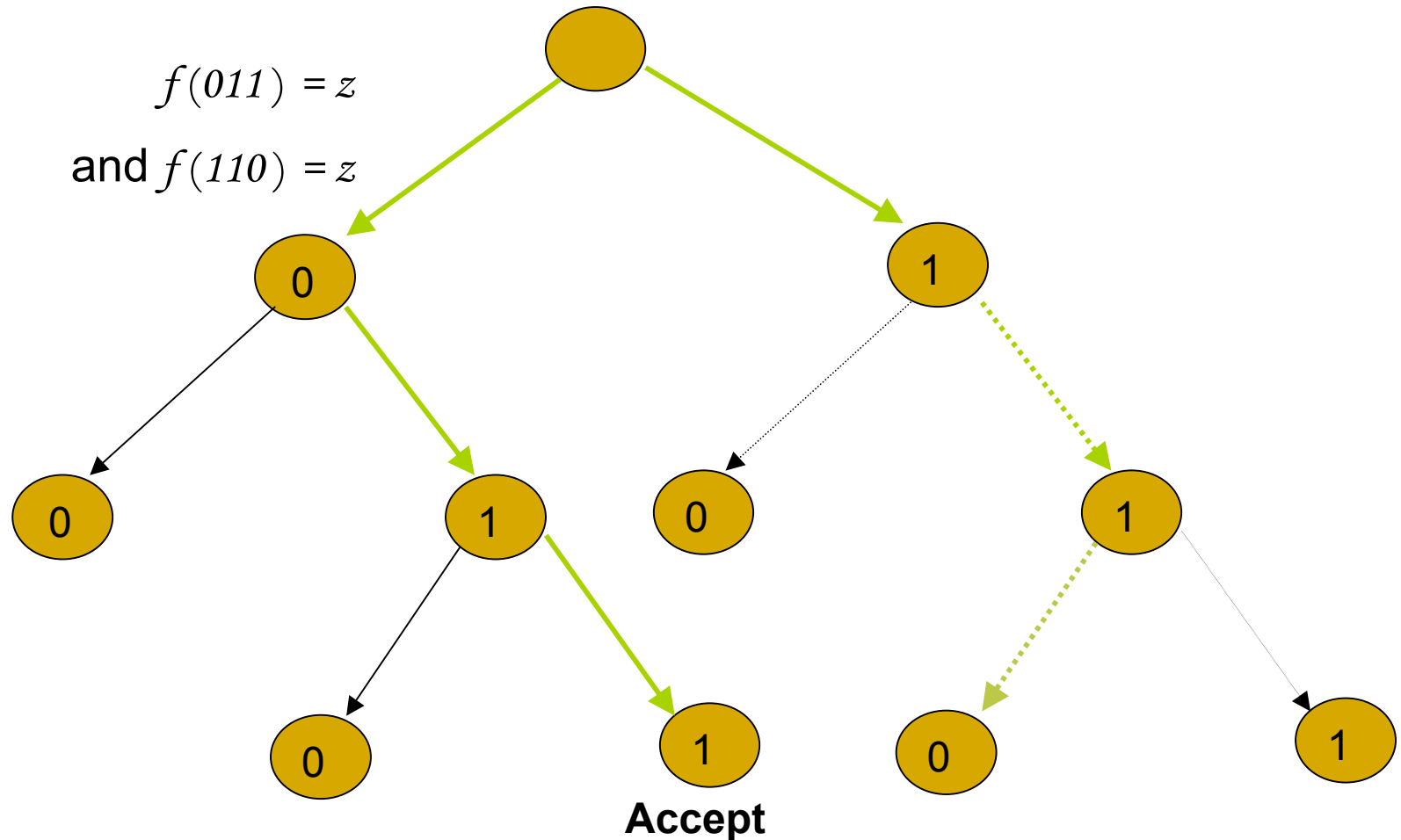
- This proof runs as that used for the second point of Theorem 2.5 (last class)

Recall from Monday's lecture that one-to-one (unambiguous) one-way functions exist $\Leftrightarrow P \neq UP$

Lets say

$$f(011) = z$$

and $f(110) = z$



Proof...

- We will now prove by induction that, $\forall k \in \{1, 2, 3 \dots\}$:

$$P = UP \quad \Rightarrow \quad P = UP_{\leq k}$$

Proof... base case

- Our base case is when $k = 1$
- When $k = 1$, then $UP_{\leq k} = UP_{\leq 1}$
 - Because $UP_{\leq 1} = UP$
- Therefore:
 - $P = UP \Rightarrow P = UP_{\leq 1}$
- Now to handle larger values of k ...

Proof... frame the inductive step

- First assume that we have:

- $P = UP \Rightarrow P = UP_{\leq k'}$

- Now use this to show that:

- $P = UP \Rightarrow P = UP_{\leq k'+1}$

Proof: $P = UP \Rightarrow P = UP_{\leq k'+1}$

- Assume $P = UP$
- Let \mathcal{L} be a arbitrary member of $UP_{\leq k'+1}$
- This means there is a NPTM \mathcal{N} where:
 - $\mathcal{L} = \mathcal{L}(\mathcal{N})$
 - \mathcal{N} has at most $k' + 1$ accepting paths

Proof...

- Consider the following language:

$$\mathcal{B} = \{ \chi \mid \mathcal{N}(\chi) \text{ has exactly } k' + 1 \text{ accepting paths} \}$$

- Perhaps not so clearly, $\mathcal{B} \in \text{UP}$
 - Why?

$\mathcal{B} \in \text{UP}$

- Let $\mathcal{N}_{\mathcal{B}}$ be a NPTM such that $\mathcal{L}(\mathcal{N}_{\mathcal{B}}) = \mathcal{B}$
- $\mathcal{N}_{\mathcal{B}}(\chi)$ is going to guess various paths $\mathcal{N}(\chi)$ might take
 - Each guess will each contain exactly $k'+1$ paths of $\mathcal{N}(\chi)$
 - Just because that is how we are defining the machine: a guess contains $k'+1$ elements
 - The paths contained in each guess will be arranged lexicographically (“uniquely sorted”)
 - This means that no two guesses will contain exactly the same set of paths
 - For each guess, $\mathcal{N}_{\mathcal{B}}(\chi)$ verifies whether each of the $k'+1$ paths are accepting paths
 - Only if all $k'+1$ paths in a given guess “check out” will $\mathcal{N}_{\mathcal{B}}(\chi)$ accept
- As we said, no two guesses by $\mathcal{N}_{\mathcal{B}}(\chi)$ will consider exactly the same set of paths
- As the guesses contain exactly $k'+1$ paths, and there are only $k'+1$ accepting paths in $\mathcal{N}(\chi)$, then there will be at most one guess that leads $\mathcal{N}_{\mathcal{B}}(\chi)$ to accept
- Note that in the cases where there are **not** $k'+1$ accepting paths in $\mathcal{N}(\chi)$, then it can only be the case that there are strictly **less than** this many accepting paths
 - In these cases $\mathcal{N}_{\mathcal{B}}(\chi)$ will reject, as the guess is hard-coded at $k'+1$ and every path in the guess must be an accepting one for $\mathcal{N}_{\mathcal{B}}(\chi)$ to accept
- This means that $\mathcal{B} \in \text{UP}$

Proof...


- We assumed that $P = UP$
- Therefore, as $B \in UP$ then $B \in P$
- This means that there must be a deterministic algorithm for deciding membership in B

Proof...

- Consider the language:

$$\mathcal{D} = \{x \mid x \notin \mathcal{B} \wedge x \in \mathcal{L}(\mathcal{N}) \}$$

Note that this exists as $\mathcal{B} \in \mathcal{P}$

- $\mathcal{N}_{\mathcal{D}}(x)$:
 - Simulate $\mathcal{M}_{\mathcal{B}}(x)$ 
 - If $\mathcal{M}_{\mathcal{B}}(x)$ accepts, then $\mathcal{N}_{\mathcal{D}}(x)$ rejects (*ie there are exactly $\ell'+1$ accepting paths*)
 - Otherwise
 - Simulate $\mathcal{N}(x)$
 - Accept if a given path of $\mathcal{N}(x)$ accepts
 - Otherwise reject

Proof...

- $\mathcal{N}_{\mathcal{D}}(x)$ has k' or less accepting paths
- Therefore $\mathcal{D} \in \text{UP}_{\leq k'}$
- As we assumed:
 - $P = \text{UP} \Rightarrow P = \text{UP}_{\leq k}$
- And since $\mathcal{D} \in \text{UP}_{\leq k'}$
- Then it must be the case that $\mathcal{D} \in P$

Proof... P is closed under union

- At this point we have:
 - $\mathcal{B} \in P$
 - $\mathcal{D} \in P$
- Now recall that P is closed under union
- This means that $\mathcal{B} \cup \mathcal{D} \in P$

Proof... $\mathcal{B} \cup \mathcal{D} = \mathcal{L}$

- $\mathcal{B} \cup \mathcal{D}$ contains all those χ 's such that, for a given χ :
 - $\mathcal{N}(\chi)$ has exactly $k' + 1$ accepting paths, or
 - $\mathcal{N}(\chi)$ has at least one and at most k' accepting paths
- But this means that $\mathcal{B} \cup \mathcal{D} = \mathcal{L}$
 - \mathcal{L} was our arbitrarily chosen language from $\text{UP}_{\leq k'+1}$
- As both \mathcal{B} and \mathcal{D} are in P , then the following must hold:
 - $\mathcal{B} \cup \mathcal{D} = \mathcal{L} \in \text{P}$

Proof... inductive proof completed

- If $\mathcal{L} \in \mathcal{P}$ under our assumptions then :

$$\mathcal{P} = \text{UP} \Rightarrow \mathcal{P} = \text{UP}_{\leq k' + 1}$$

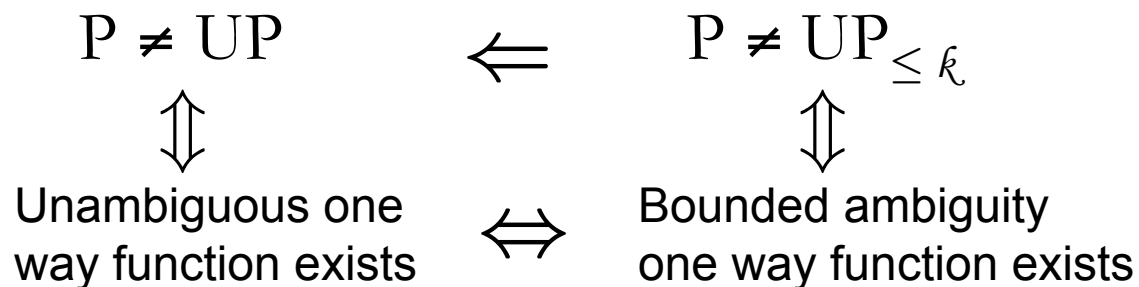
- This was our inductive step

- Which means we can conclude:

$$\mathcal{P} = \text{UP} \Rightarrow \mathcal{P} = \text{UP}_{\leq k}$$

Proof... recalling our mission

- We are trying to show that the existence of unambiguous one way functions is explicitly tied to the existence of *bounded ambiguity* one-to-one functions
- We broke up the if-and-only-if to see that one direction was trivial, while the other direction involved a round-about path:



Proof... we are done

- This means that we have finished the proof:
- Theorem 2.7
 - Unambiguous one way functions exist \Leftrightarrow bounded ambiguity one way functions exist

Summary

- Key take aways:
 - On Monday we showed that:
 - The existence of one-to-one one way functions are tied to whether the language class P equals UP
 - Today we showed a stronger version:
 - k -to-one one way functions exist iff $P \neq UP_{\leq k}$
 - In addition, we showed that 1-to-one one way functions exist iff k -to-one one way functions exist
 - Certainly an interesting fact!
- At this point we will move on to section 2.3 of the textbook, in order to provide a first glimpse of the required definitions

Definition 2.10: Honesty

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is honest if

$$\begin{aligned} & (\exists \text{ polynomial } q) (\forall y \in \text{range}(f)) \\ & (\exists x, x') [|x| + |x'| \leq q(|y|) \wedge f(x, x') = y] \end{aligned}$$

- Informally:
 - A 2-ary function f is honest if there's a polynomial p such that $p(|f \text{'s output}|)$ is greater than the sum of the length of both inputs

Defn 2.11: polynomial time invertible

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is polynomial time invertible if there is a polynomial time computable function g such that, for every $y \in \text{range}(f)$:

$$y \in \text{domain}(g) \wedge$$

$$(\text{first}(g(y)), \text{second}(g(y))) \in \text{domain}(f) \wedge$$

$$f(\text{first}(g(y)), \text{second}(g(y))) = y,$$

where the project functions $\text{first}(z)$ and $\text{second}(z)$ denote, respectively, the first and second components of the unique ordered pair of strings that, when paired, give z

Defn 2.12: One way function

- A 2-ary function $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is one-way if
 - f is polynomial time computable
 - f is not polynomial time invertible and
 - f is honest

Defn 2.13: s-honest

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is s-honest if
 - $(\exists \text{ polynomial } q) (\forall y, a: (\exists b)[f(a, b) = y])$
 $(\exists b') [|b'| \leq q(|y| + |a|) \wedge f(a, b') = y].$
 - $(\exists \text{ polynomial } q) (\forall y, b: (\exists a)[f(a, b) = y])$
 $(\exists a') [|a'| \leq q(|y| + |b|) \wedge f(a', b) = y].$

Defn: 2.14 strongly non invertible

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is strongly-noninvertible if it is s-honest and yet neither of the following conditions holds:
 - There is a polynomial-time computable function $g: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ such that $(\forall y \in \text{range}(f)) (\forall x_1, x_2: (x_1, x_2) \in \text{domain}(f) \wedge f(x_1, x_2) = y) [(y, x_1) \in \text{domain}(g) \wedge f(x_1, g(y, x_1)) = y]$
 - There is a polynomial-time computable function $g: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ such that $(\forall y \in \text{range}(f)) (\forall x_1, x_2: (x_1, x_2) \in \text{domain}(f) \wedge f(x_1, x_2) = y) [(y, x_2) \in \text{domain}(g) \wedge f(g(y, x_2), x_1) = y]$

Defn: 2.14 strongly non invertible contd...

- A 2-ary function is strongly non-invertible if, even given one of its inputs and its output, the other input cannot be computed in polynomial time.

Defn: 2.15: Associativity & commutativity

- A total, 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is associative if:

$$(\forall x, y, z) [f(f(x, y), z) = f(x, f(y, z))]$$

- A total, 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is commutative if:

$$(\forall x, y) [f(x, y) = f(y, x)]$$

Theorem 2.16:

- One-way functions exist if and only if strongly noninvertible, total, commutative, associative, 2-ary one way functions exist

Hem-Ogi 2.3 :

One-way functions exist \Leftrightarrow strongly
noninvertible, total, commutative,
associative, 2-ary one-way functions exist

Group 2:

Ben Van Durme

Pin Lu

Ross Messing

Shiva Shankar Balu

Tanushree Mittal

Definition 2.10: Honesty

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is honest if

$$\begin{aligned} & (\exists \text{ polynomial } q) (\forall y \in \text{range}(f)) \\ & (\exists x, x') [|x| + |x'| \leq q(|y|) \wedge f(x, x') = y] \end{aligned}$$

- Informally:
 - A 2-ary function f is honest if there's a polynomial p such that $p(|f \text{'s output}|)$ is greater than the sum of the length of two arguments which give that output

Defn 2.11: polynomial time invertible

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is polynomial time invertible if there is a polynomial time computable function g such that, for every $y \in \text{range}(f)$:

$$\begin{aligned} & y \in \text{domain}(f) \wedge \\ & (\text{first}(g(y)), \text{second}(g(y))) \in \text{domain}(f) \wedge \\ & f(\text{first}(g(y)), \text{second}(g(y))) = y, \end{aligned}$$

where the functions $\text{first}(z)$ and $\text{second}(z)$ denote, respectively, the first and second components of the ordered pair of strings that can be paired to form z

Defn 2.12: One way function

- A 2-ary function $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is one-way if
 - f is polynomial time computable
 - f is not polynomial time invertible and
 - f is honest

Defn 2.13: s-honest

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is s-honest if
 - $(\exists \text{ polynomial } q) (\forall y, a: (\exists b)[f(a, b) = y])$
 $(\exists b') [|b'| \leq q(|y| + |a|) \wedge f(a, b') = y].$
 - $(\exists \text{ polynomial } q) (\forall y, b: (\exists a)[f(a, b) = y])$
 $(\exists a') [|a'| \leq q(|y| + |b|) \wedge f(a', b) = y].$
- For any $y \in f$'s range, there exists an a and b such that $f(a, b) = y$. We say that f is s-honest if there exists a bounding polynomial q , and an argument b' such that $q(|y| + |a|) \geq |b'|$, and $f(a, b') = f(a, b) = y$.

Defn: 2.14 strongly noninvertible

- A 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is strongly noninvertible if it is s-honest but neither of the following conditions hold:
 - There is a polynomial-time computable function $g: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ such that $(\forall y \in \text{range}(f)) (\forall x_1, x_2 : (x_1, x_2) \in \text{domain}(f) \wedge f(x_1, x_2) = y) [(y, x_1) \in \text{domain}(g) \wedge f(x_1, g(y, x_1)) = y]$
 - There is a polynomial-time computable function $g: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ such that $(\forall y \in \text{range}(f)) (\forall x_1, x_2 : (x_1, x_2) \in \text{domain}(f) \wedge f(x_1, x_2) = y) [(y, x_2) \in \text{domain}(g) \wedge f(g(y, x_2), x_1) = y]$
- A 2-ary function is strongly noninvertible if, even given one of its inputs and its output, the other input cannot be computed in polynomial time.

Defn: 2.15: Associativity & commutativity

- A total, 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is associative if:

$$(\forall x, y, z) [f(f(x, y), z) = f(x, f(y, z))]$$

- A total, 2-ary function $f: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is commutative if:

$$(\forall x, y) [f(x, y) = f(y, x)]$$

Proposition 2.17

- The following are equivalent
 - One-way functions exist
 - 2-ary one-way functions exist
 - $P \neq NP$

Proof of Proposition 2.17

- One-way functions exist $\Leftrightarrow P \neq NP$
 - See Theorem 2.5 in section 2.1
- One-way functions exist \Leftrightarrow 2-ary one-way functions exist
 - One-way functions exist \Leftarrow 2-ary one-way functions exist
 - One-way functions exist \Rightarrow 2-ary one-way functions exist

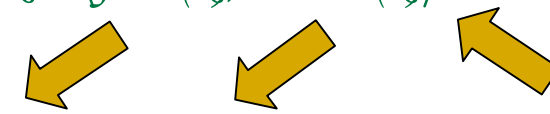
One-way functions exist \Leftarrow 2-ary one-way functions exist

- One-way functions exist if 2-ary one-way functions exist

- Let f be any 2-ary one-way function, and define g as

$$g(x) = f(\text{first}(x), \text{second}(x))$$

$x = \langle \text{first}(x), \text{second}(x) \rangle$



One to One

where $\text{first}(x)$ and $\text{second}(x)$ respectively denote the first and second component of the unique pair mapping to x by the pairing function

- Clearly, g is one-way function.

One-way functions exist \Rightarrow 2-ary one-way functions exist

- One-way functions exist only if 2-ary one-way functions exist
 - Let h be any one-way function. Define h' :
 - $h'(x, y) = \langle h(x), y \rangle$. Then h' is an obvious 2-ary one-way function
 - Or $h''(x, y) = \langle h(x), h(y) \rangle$. Then h'' is also a 2-ary one-way function, but with strong noninvertibility (see Definition 2.14)

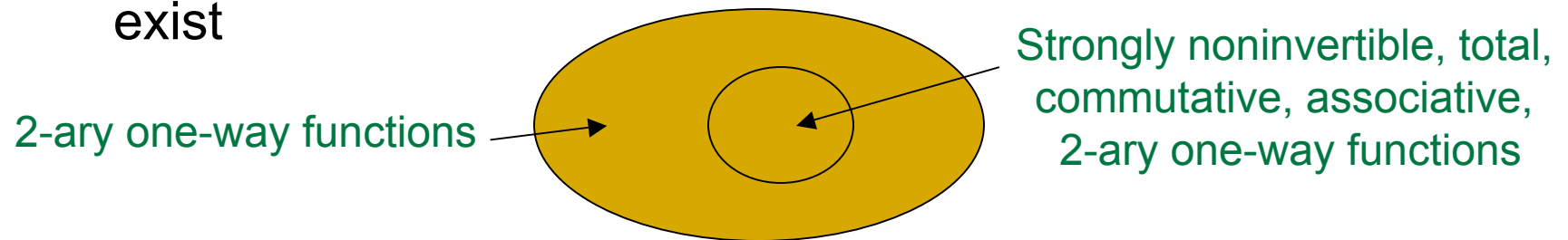
Theorem 2.16

- One-way functions exist \Leftrightarrow strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist.

Proof : *if* direction of Theorem 2.16

■ *If*

- By Proposition 2.17, one-way functions exist \Leftrightarrow 2-ary one-way functions exist
- Strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist \Rightarrow 2-ary one-way functions exist



- Therefore, strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist \Rightarrow One-way functions exist

Proof : *only if* direction of Theorem 2.16

■ *only if*


□ By proposition 2.17, we have

■ $P \neq NP \Leftrightarrow$ One-way functions exist \Leftrightarrow 2-ary one-way functions exist

□ To prove the goal that One-way functions exist \Rightarrow strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist, we can equivalently show

■ $P \neq NP \Rightarrow$ strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist

Proof : *only if* direction of Theorem 2.16

- $P \neq NP \Rightarrow$ strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist
 - By the premise that $P \neq NP$, then there exists a NPTM \mathcal{N}' such that $\mathcal{L}(\mathcal{N}') \in NP - P$
 -  How do we do this Standard Machine Manipulation?
By a Standard Machine Manipulation, there exists a polynomial p and a NPTM \mathcal{N} such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N}')$ and $\forall x$ the computation paths of $\mathcal{N}(x)$ have exactly $p(|x|)$ bits

Standard Machine Manipulation

- Standard Machine Manipulation
 - We construct \mathcal{N} as follows:
 - First, we construct a polynomial q , such that $q(x) = \text{Max}(p'(x), x+1)$, where p' where p' refers to the polynomial time bound for \mathcal{N}' .
 - As $\mathcal{N}'(x)$ runs, we count the number of nondeterministic guesses it makes, and call that m . At the end of each computation path of $\mathcal{N}'(x)$, we make $q(|x|) - m$ additional nondeterministic dummy guesses.
 - Therefore, for each input x , the length of any computation path of $\mathcal{N}(x)$ is exactly $q(|x|)$.
 - Obviously, it is guaranteed that the length of each computation path is greater than the length of the input
 - So we have built a new NPTM \mathcal{N} from \mathcal{N}' . \mathcal{N} accepts the same language as \mathcal{N}' and for each input x , the length of all computation paths of $\mathcal{N}(x)$ are exactly of length $q(|x|)$, which is greater than $|x|$

Definition of Witness


■ Definition

- All computation paths are viewed as potential witnesses for $x \in \mathcal{L}(\mathcal{N})$.
- We call a path a witness for $x \in \mathcal{L}(\mathcal{N})$ if it is an accepting path of $\mathcal{N}(x)$.
- We define $\mathcal{W}(x)$ as the set of all witnesses for $x \in \mathcal{L}(\mathcal{N})$.
- Note that no string can be the witness of itself for the previously defined NPTM \mathcal{N} , because our machine manipulation requires that the length of any computation path is greater than the length of the input.

Definition of the function f

- Now we define a function f , which we will prove to be a strongly noninvertible, total, commutative, associative, 2-ary one-way function.

$$f(u, v) = \begin{cases} \langle x, \text{lexmin}(w_1, w_2) \rangle & \text{If } u = \langle x, w_1 \rangle \wedge v = \langle x, w_2 \rangle \\ & \wedge \{w_1, w_2\} \subseteq W(x) \\ \langle x, x \rangle & \text{If } (\exists w \in W(x))[\{u, v\} = \\ & \{\langle x, x \rangle, \langle x, w \rangle\}] \\ \langle t, t1 \rangle & \text{Otherwise,} \end{cases}$$

 **t is any fixed string that is not in $\mathcal{L}(\mathcal{N})$**

Proof : f is total and polynomial-time computable

- f is defined over $\forall (x_1, x_2) \in \Sigma^* \times \Sigma^*$, thus f is total
- f is polynomial-time computable
 - Pairing function is polynomial-time computable
 - We get two pairs for two arguments of f , respectively
 - The string comparison is poly-time computable
 - Test if the first elements of both arguments match
 - Test the second element of each pair to check if it is the witness on NPTM \mathcal{N} of the first element of the pair.
 - $\mathcal{N}(x)$ is checkable in deterministic polynomial time

Proof : f is commutative

- If the input (u, v) falls into the first case,

$$u = \langle x, w_1 \rangle \wedge v = \langle x, w_2 \rangle \wedge \{w_1, w_2\} \subseteq W(x)$$

- The commutativity of f holds, because function $lexmin$ itself is commutative. No matter which order it's in, the output is always $\langle x, q \rangle$, where q is the lexicographically less of u 's and v 's second components

$$f(u, v) = \langle x, lexmin(w_1, w_2) \rangle$$

Proof : f is commutative

- If the input (u, v) falls into the last two cases of f , then $f(u, v) = f(v, u)$ holds

- Case 2: If one of the arguments is the pair $x \in \mathcal{L}(\mathcal{N})$, and its witness w , and the other is the pair $\langle x, x \rangle$

$$(\exists w \in W(x))[\{u, v\} = \{\langle x, x \rangle, \langle x, w \rangle\}]$$



Note that this is a set, so the order of the two arguments does not matter

- Case 3:
 - Since the first two cases are commutative, if an input pair (x, y) does not fall into the first two cases, (y, x) also cannot, which means $f(x, y) = f(y, x) = \langle t, t1 \rangle$

f is s-honest

- f is s-honest

- Witnesses for NPTM \mathcal{N} are of length bounded polynomially in the length of their input string
- Therefore, for the first two cases of f , when we fix one argument, the length trick cannot succeed on the other argument, since two arguments with the same first element must be no more than polynomially longer or shorter than each other.

f is s-honest

■ f is s-honest

- For the third case of f , given the output $\langle t, t1 \rangle$ and one fixed argument, we can always find another argument $\langle a, b \rangle$ whose length falls within a polynomial bound, and we can ensure that it produces the correct output by ensuring that a isn't the same as the first element of the other argument

Proof : f is strongly noninvertible

- Assume f is not strongly noninvertible
 - Since we have proven that f is s -honest, strong noninvertibility must fail because at least one of the two conditions in the definition of strong noninvertibility holds. This means that given the output and one argument, the other argument can be computed in polynomial-time

Proof : f is strongly noninvertible

- Then, there exists a polynomial-time function g such that, when we consider Case 2,
 - If $x \in \mathcal{L}(\mathcal{N})$, $g(\langle x, x \rangle, \langle x, x \rangle)$ should output $\langle x, w \rangle$, where $w \in \mathcal{W}(x)$

One argument and the output

The other argument

- This gives us a deterministic polynomial-time algorithm to test input x 's membership in $\mathcal{L}(\mathcal{N})$
 - On input x , first compute $g(\langle x, x \rangle, \langle x, x \rangle)$, reject if the output is not of the form $\langle x, w \rangle$
 - Then simulate $\mathcal{N}(x)$ on computation path w , accept x if $\mathcal{N}(x)$ accepts

Proof : f is strongly noninvertible

- But we've revealed a contradiction!
 - Remember, we've assumed that $\mathcal{L}(\mathcal{N}) \in \text{NP-P}$
 - But now we have a deterministic polynomial-time algorithm to test membership in $\mathcal{L}(\mathcal{N})$
 - Therefore, the assumption that f is not strongly noninvertible must be wrong
- So, f satisfies the definition of strong noninvertibility

Proof : f is honest

- It is easy to verify f is honest in Case 1 and 2
 - The pairing function is polynomial-time computable and invertible

$|\langle a, b \rangle|$ and $|a| + |b|$ are bounded by one another

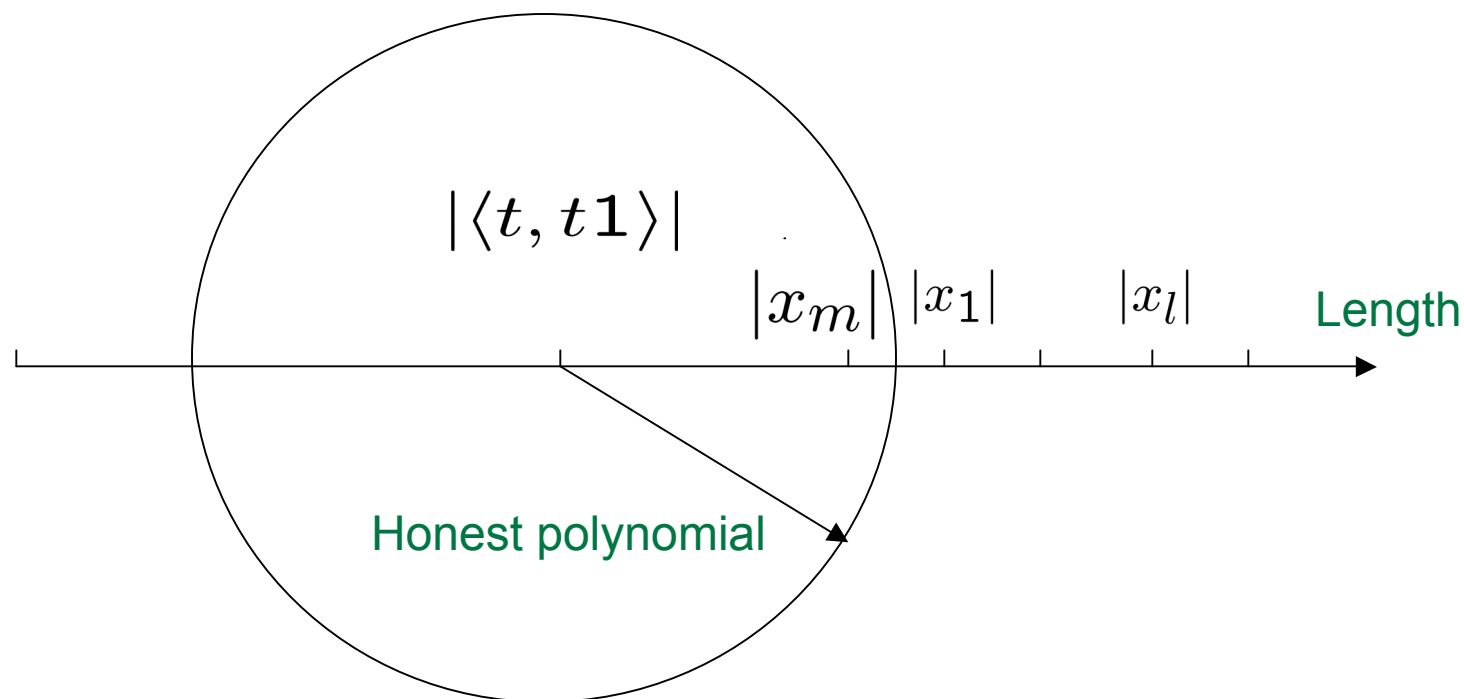
- The witnesses of all strings in $\mathcal{L}(\mathcal{N})$ are length-bounded by \mathcal{N} 's polynomial time bounding polynomial. Furthermore, as required by our machine manipulation, $\forall x \in \mathcal{L}(\mathcal{N}), |w| = q(|x|)$, which is still polynomial
- Thus, f cannot dramatically distort the length of input

Proof : f is honest

- For Case 3, we expand the honesty polynomial to cover the shortest input mapping to $\langle t, t1 \rangle$. By the definition of honesty, we only need to guarantee there exists one input for each output whose length is polynomially bounded by each output
- How does it work?

Proof : f is honest

- Suppose $x_m = \langle x_m', x_m'' \rangle$ is the shortest input on which f outputs $\langle t, t1 \rangle$



Proof : f is associative

- f is associative \Leftrightarrow For each $z, z', z'' \in \Sigma^*$,

$$f(f(z, z'), z'') = f(z, f(z', z''))$$

Some definitions

- As previously defined, $first(z)$ and $second(z)$ are the first and second elements of the pair z created by our pairing function
- A string a is *Legal* if

$$(\exists x)(\exists w)[w \in W(x) \wedge a = \langle x, w \rangle]$$

Discuss over all cases

- Case 1: At least two of z, z', z'' are not *legal*

- Then, $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle t, t1 \rangle$

- Case 2: If it is not the case that

$$first(z) = first(z') = first(z'')$$

- Again, $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle t, t1 \rangle$

- Case 3: if $first(z) = first(z') = first(z'')$ and exactly one of z, z', z'' is not *legal* and the one that is not *legal* is not of the form $\langle first(z), first(z) \rangle$

- Still, $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle t, t1 \rangle$

Discuss over all cases

- Case 4: if $first(z) = first(z') = first(z'')$ and exactly one of z, z', z'' is not *legal* and the one that is not *legal* is of the form $\langle first(z), first(z) \rangle$

- $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle first(z), first(z) \rangle$

- Case 5: if $first(z) = first(z') = first(z'') = \chi$ and all of z, z', z'' are *legal*

$$z = \langle x, w_1 \rangle \wedge z' = \langle x, w_2 \rangle \wedge z'' = \langle x, w_3 \rangle \\ \wedge \{w_1, w_2, w_3\} \subseteq W(x)$$

- $f(f(z, z'), z'') = f(z, f(z', z'')) = \langle first(z), q \rangle$, where q is the lexicographically least of $second(z), second(z'), second(z'')$. This works because lexicographic minimum is associative.

Conclusion

- We have shown that $P \neq NP \Rightarrow f$ is a strongly noninvertible, total, commutative, associative, 2-ary one-way function
- Therefore, $P \neq NP \Rightarrow$ strongly noninvertible, total, commutative, associative, 2-ary one-way functions exist
- Theorem 2.16 is proved