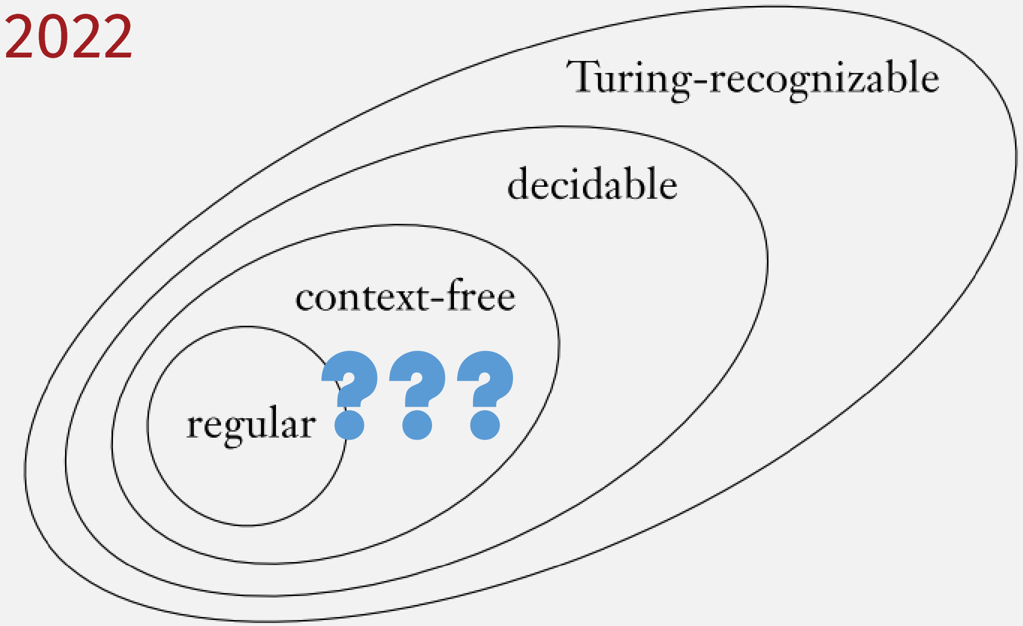
UMB CS 420

# Non-Regular Languages

Tuesday, October 11, 2022



# Announcements

- HW 3 in
  - ~~Due Sun 10/9 11:59pm EST~~
- HW 4 out
  - Due Sun 10/16 11:59pm EST
- Submitted hw must correctly assign pages to each problem
  - Incorrectly assigned problems are marked zero
  - We will re-grade one time, if re-grade request is submitted

A ***language*** is a set of strings.

# *So Far:* Regular or Not?

$M$ ***recognizes language*** $A$ if $A = \{w \mid M \text{ accepts } w\}$

- Many ways to prove a language is regular:
  - Construct a **DFA** recognizing it
  - Construct an **NFA** recognizing it
  - Create a **regular expression** describing the language

- Regular Expression ⇔ NFA ⇔ DFA ⇔ Regular Language

- But not all languages are regular!
  - Most programming language syntaxes are not regular
    - e.g., language of all python programs, or all HTML/XML pages, are not regular
  - That means:
    - There's no DFA or NFA recognizing those languages
    - And they can't be described with a regular expression (a common mistake)!

# Someone Who Did Not T

## RegEx match open tags except XHTML self-con

Asked 10 years, 10 months ago Active 1 month ago Viewed 2.9m times

1553

I need to match all of these opening tags:

```
<p>
<a href="foo">
```

Trying to use regular expressions to describe the non-regular HTML language

6572

But not these:

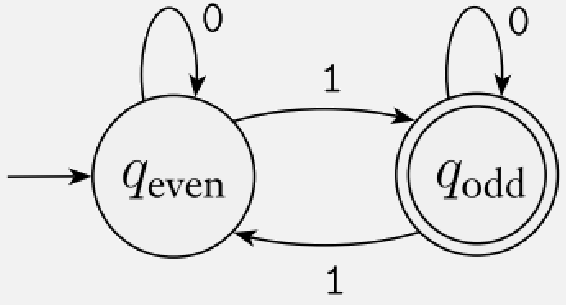
4414

You can't parse [X]HTML with regex. Because HTML can't be parsed
Regex is not a tool that can be used to correctly parse HTML. As I h
HTML-and-regex questions here so many times before, the use of re
allow you to consume HTML. Regular expressions are a tool that is
sophisticated to understand the constructs employed by HTML. HTM
regular language and hence cannot be parsed by regular expressior
queries are not equipped to break down HTML into its meaningful pa
times but it is not getting to me. Even enhanced irregular regular exp
used by Perl are not up to the task of parsing HTML. You will never

HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The <center> cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the trangession of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML h ummm ... ty to an eternity of dread torture and security holes *using regex* as a tool to process HTML establishes a breach *between this world* and the dread realm of corrupt entities (like SGML entities, but *more corrupt*) *a mere glimps*e of the world of rege**x parsers for HTML will ins**tantly transport a p*rogrammer's consciousness i*nto a world of ceaseless screaming, he comes~~, the pestilent s~~lithy regex-infection wil**l devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fi*ght h**e com̡e̶s, ̕h̵is** un̨ho͞ly radiańcé destro҉ying all enli̍̈́̂̈́ghtenment, HTML tags **lea͠ki̧n͘g fr̶ǫm ̡yo͟ur eye͢s̸ ̛l̕ik͏e liq**uid pain, the song of re̸gular exp~~ression parsing~~ will exti*nguish the voices of mor*tal man from the sp*here I can see it can you see ̲͚̖͔̙î̩́t̲͎̩̱͔́̋̀ it is beautiful t*he f`inal snuf`fing o*f the lie*s of Man ALL IS LOŚ͖̩͇̗̪̏̈́T ALL I*S LOST the pon̷y he comes he c̶̮omes ~~he comes t~~he ich*or permeates all MY FAC*E MY FACE ᵒh god n**o NO NOO̼OO NΘ** stop t*he an**͑̾̾̅ͫ͏̙̤g͇̫͛͆̾ͫ̑͆l͖͉̗̩̳̟̍ͫͥͨe̠̅s* `a̧͈͖r̽̾̈́͒͑e` n**ot rè̑ͧ̌aͨl̘̝̙̃ͤ͂̾̆ ZA̡͊͠͝LGΌ ISͮ̂҉̯͈͕̹̘̱ TO͇̹̺ͅƝ̴ȳ̳ TH̘Ë͖́̉ ͠P̯͍̭O̚N̐Y̡ H̸̡̪̯ͨ͊̽̅̾̎Ȩ̬̩̾͛ͪ̈́̀́͘ ̶̧̨̱̹̭̯ͧ̾ͬC̷̙̲̝͖ͭ̏ͥͮ͟Oͮ͏̮̪̝͍M̲̖͊̒ͪͩͬ̚̚͜Ȇ̴̟̟͙̞ͩ͌͝S̨̥̫͎̭ͯ̿̔̀ͅ**

Have you tried using an XML parser instead?

# *Flashback:* Designing DFAs or NFAs




- Each state "stores" some information
    - E.g., $q_{\text{even}}$ = "seen even # of 1s", $q_{\text{odd}}$ = "seen odd # of 1s".
  - Finite states = finite amount of info (must decide in advance)

- So DFAs can't keep track of an arbitrary count!
  - would require infinite states

# A Non-Regular Language

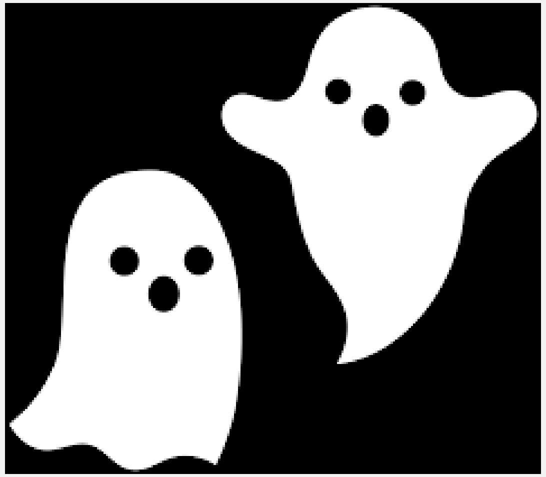
An arbitrary count

$L = \{ \mathtt{0}^n\mathtt{1}^n \mid n \geq 0 \}$

- A DFA recognizing $L$ would require infinite states! (impossible)
  - States representing zero `0`s, one `0`, two `0`s, …
- This language represents the essence of many PLs, e.g., HTML!
  - To better see this replace:
    - “`0`” with “`<tag>`“ or “`(`“
    - “`1`” with “`</tag>`” or “`)`”
- The problem is tracking the **nestedness**
  - Regular languages cannot count arbitrary nesting depths
    - E.g., `if { if { if { … } } }`
  - So most programming language syntax is not regular!

Still, how do we **prove non-regularness?**

# Prove: Ghosts Do Not Exist



It's hard to prove that something is <u>not true</u>!

In some cases, it's possible, but typically requires complicated proof techniques!

So: proving a language <u>is not regular</u> ...
is harder than proving a language <u>is regular</u>

# A Lemma About Regular Languages

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Specifically, <u>all regular languages</u> satisfy these 3 conditions!

This lemma describes a property that <u>all regular</u> languages have.

<u>Hint:</u> This is an "If $X$ then $Y$" statement

<u>Note</u>: this lemma only applies to <u>known regular languages</u>!

(but maybe **it can be** used to prove that a language **is not regular!**)

Can we use this to prove that language **is regular?**

NO (but we already know how to do that anyways)

# Equivalence of Conditional Statements

- Yes or No? "If $X$ then $Y$" is equivalent to:
  - "If $Y$ then $X$" (**converse**)
    - No!
  - "If not $X$ then not $Y$" (**inverse**)
    - No!
  - "If not $Y$ then not $X$" (**contrapositive**)
    - <u>Yes!</u>

If-then statement

... then **the language** is **not regular**

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Equivalent (**contrapositive**):
If **any of these are not true** ...

**Contrapositive:**
"If *X* then *Y*" is <u>equivalent</u> to "If **not** *Y* then **not** *X*"

# Logical Inference Rules

## Modus Ponens

Premises (known facts)

- If $P$ then $Q$
- $P$ is true

Conclusion (new fact)

- $Q$ is true

## Modus Tollens (contrapositive)

Premises (known facts)

- If $P$ then $Q$
- $Q$ is *not* true

Conclusion (new fact)

- $P$ is *not* true

# Lemma About Regular Languages: Details

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

<u>All regular languages</u> satisfy these three conditions!

Specifically, these conditions apply to strings in the language of <u>length</u> $\geq$ ***p***

<u>NOTE</u>:
- Lemma doesn't give an exact $p$!
- Just that there is some string length $p$ …
- … those strings must obey the 3 conditions

# The Pumping Lemma: Finite Lang

Conclusion: **pumping lemma is only interesting for infinite langs!** (containing strings with repeatable parts)

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

**Lemma doesn't say what $p$ is! Just that there is a $p$**

**So finite langs** (specifically, all strings in the language "of length at least $p$") **must satisfy these conditions**

**Possible $p$ for finite langs?**

How about:

$p$ = LENGTH(longest string) + 1

**# strings in the language with length ≥ $p$?** **None!**

Therefore, **all strings with length ≥ $p$ satisfy the pumping lemma conditions!** ☺

Example: **a finite language {"ab", "cd"}**

- **All finite langs are regular**
- (can easily construct DFA/NFA recognizing them)

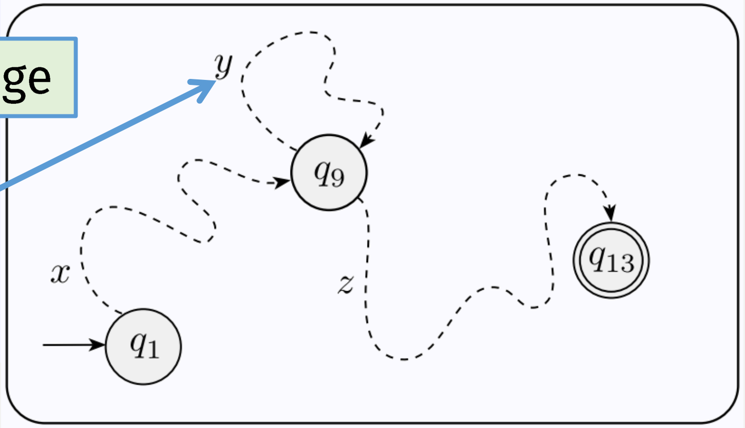
# The Pumping Lemma, a Closer Look

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $x\boxed{y^i}z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

"pumped" string still in language

("long enough") strings of length ≥ $p$ have a <u>repeatable</u> ("pumpable") part



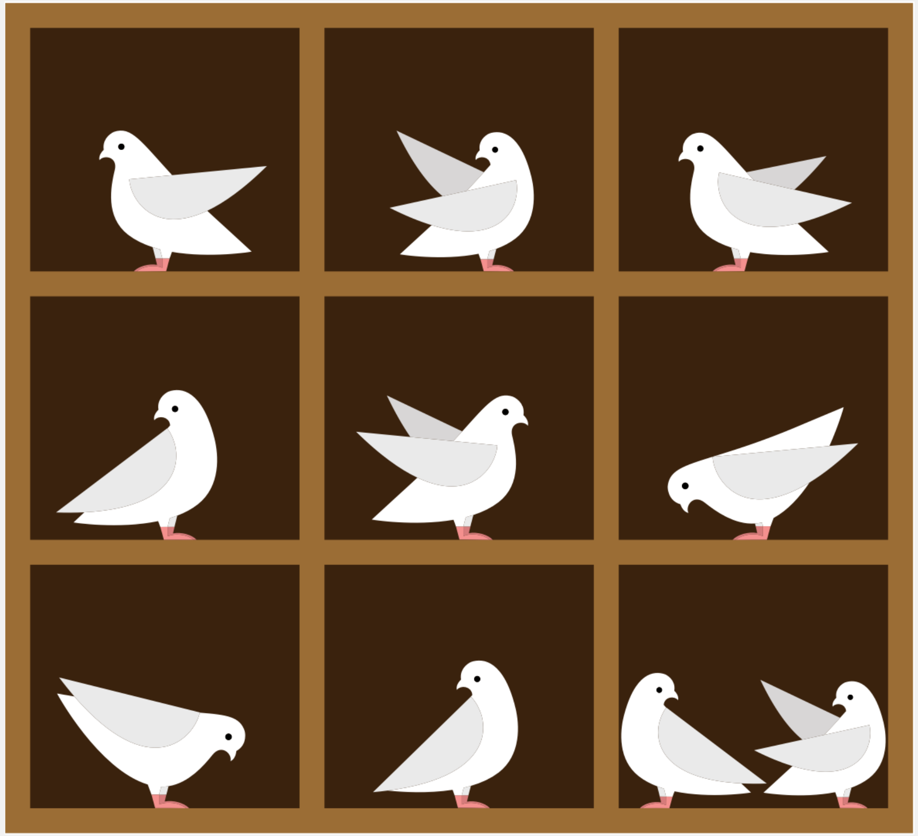

Strings that have a <u>repeatable</u> part can be split into:

- $x$ = part <u>before</u> any repeating
- $y$ = <u>repeated</u> (or "pumpable") part
- $z$ = part <u>after</u> any repeating

This makes sense because DFAs have finite states, so for "long enough" (i.e., length ≥ $p$) inputs, <u>some state must repeat</u>

e.g., "long enough length" = $p$ = **# states** +1
(The Pigeonhole Principle)

# The Pigeonhole Principle



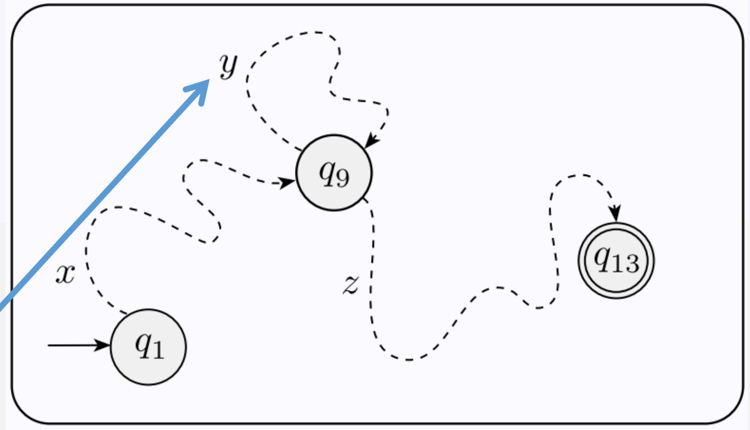
If # birds > # holes, then there must be > 1 bird in some hole

# The Pumping Lemma, a Closer Look

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $x\boxed{y^i}z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

So **a substring that can repeat once, can also be repeated <u>any number of times</u>**




Also, **this is the <u>only</u> way for regular languages to repeat** (Kleene star)

In essence, **the pumping lemma is a theorem about the <u>structure of repeatable patterns in regular languages</u>**

**"long enough length" =** $p$ **= # states** +1
(some state must repeat)

# The Pumping Lemma: Infinite Languages

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

"pumpable" part of string

Note: "pumpable" part cannot be empty

E.g., "010" ∈ *A*, so pumping lemma says splittable into three parts *xyz*
- $x = 0, y = 1, z = 0$

<u>Example</u>: *infinite* language *A* = {"00", $\boxed{\text{“010”}}$, "0110" , "01110", ...}
- It's regular bc it has regular expression **01*0**

... and "pumping" (repeating) middle *y* part creates a string that is <u>still in the language</u>
- repeat <u>once</u> ($i = 1$): "0**1**0",
- repeat <u>twice</u> ($i = 2$): "0**11**0",
- repeat <u>three</u> times ($i = 3$): "0**111**0"

# Summary: The Pumping Lemma …

- … states properties that are true for all regular languages
- … specifically, properties about repetition in regular languages

**IMPORTANT**:

- The Pumping Lemma cannot prove that a language is regular!

- But … we can use it to prove that a language is not regular

If-then statement

... then **the language is not regular**

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Equivalent (**contrapositive**):
If any of these are **not** true ...

**Contrapositive:**
"If $X$ then $Y$" is <u>equivalent</u> to "If **not** $Y$ then **not** $X$"

# Kinds of Mathematical Proof

- Deductive Proof
  - Logically infer conclusion from known definitions and assumptions
- Proof by induction
  - Use to prove properties of recursive definitions or functions
- Proof by contradiction
  - Proving the contrapositive

# How To Do Proof By Contradiction

3 easy steps:

1. <u>Assume the opposite</u> of the statement to prove

2. Show that the assumption <u>leads to a contradiction</u>

3. <u>Conclude</u> that the original statement must be true

# Pumping Lemma: Non-Regularity Example

Let $B$ be the language $\{0^n 1^n | n \geq 0\}$. We use the pumping lemma to prove that $B$ is not regular. The proof is by contradiction.

Want to prove: $0^n1^n$ **is not** a regular language

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Reminder: Pumping lemma says: all strings $0^n1^n \geq$ length $p$ **are splittable** into $xyz$ where $y$ is pumpable

So find string $\geq$ length $p$ that is **not splittable** into $xyz$ where $y$ is pumpable

Proof (by contradiction):

Now we must find a contradiction …

- Assume: $0^n1^n$ **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length $p$ are pumpable
- Counterexample = $0^p1^p$

Want to prove: $0^n1^n$ **is not** a regular language

... then **not** true

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Contrapositive: If **not** true ...

# Possible Split: $y$ = all 0s

Reminder: Pumping lemma says: all strings $0^n1^n \geq$ length $p$ **are splittable** into $xyz$ where $y$ is pumpable

So find string $\geq$ length $p$ that is **not splittable** into $xyz$ where $y$ is pumpable

Proof (by contradiction):

contradiction

- Assume: $0^n1^n$ **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length $p$ are pumpable
- Counterexample = $0^p1^p$
- Choose $xyz$ split so $y$ contains:
  - all 0s

$$\overbrace{\underbrace{00}_{x}\underbrace{\ldots 0}_{y}}^{p\ 0\text{s}}\overbrace{\underbrace{11\ldots 1}_{z}}^{p\ 1\text{s}}$$

BUT ... pumping lemma requires **only one** pumpable splitting

So the proof is not done!

Is there another way to split into $xyz$ ?

- Pumping $y$: produces a string with more 0s than 1s
  - ... not in the language $0^n1^n$
  - So $0^p1^p$ **is not pumpable** (according to pumping lemma)
  - So $0^n1^n$ is a not regular language (contrapositive)
  - This is a **contradiction** of the assumption!

Want to prove: $0^n1^n$ is **not** a regular language

# Possible Split: $y$ = all 1s

Proof (by contradiction):

- Assume: $0^n1^n$ **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings ≥ length $p$ are pumpable
- Counterexample = $0^p1^p$

$$\underbrace{\overbrace{00\ \ldots\ 0}^{p\text{ 0s}}}_{x}\overbrace{\underbrace{11}_{y}\underbrace{\ \ldots\ 1}_{z}}^{p\text{ 1s}}$$

- Choose $xyz$ split so $y$ contains:
  - all 1s

- Is this string pumpable?
  - No!
  - By the same reasoning as in the previous slide

Is there another way to split into $xyz$ ?

Want to prove: $0^n1^n$ is **not** a regular language

# Possible Split: $y$ = 0s and 1s

Proof (by contradiction):

- Assume: $0^n1^n$ **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings $\geq$ length $p$ are pumpable
- Counterexample = $0^p1^p$

$$\overbrace{00\ldots0}^{p\text{ 0s}}\overbrace{11\ldots1}^{p\text{ 1s}}$$

$$\underbrace{00\ldots}_{x}\underbrace{\ldots011\ldots}_{y}\underbrace{\ldots1}_{z}$$

- Choose $xyz$ split so $y$ contains:
  - both 0s and 1s

Did we examine every possible splitting?

**Yes! QED**

- Is this string pumpable?
  - No!
  - Pumped string will have equal 0s and 1s
  - But they will be in the wrong order: so there is still a **contradiction**!

But maybe we did't have to …

# The Pumping Lemma: Condition 3

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

The repeating part $y$ ...
must be in the first $p$ characters!

$$\overbrace{00 \ldots 0}^{p\text{ 0s}}11 \ldots 1$$

$y$ must be in here!

# The Pumping Lemma: Pumping Down

**Pumping lemma** If $A$ is a regular language, then there is a number $p$ (the pumping length) where if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $\boxed{i \geq 0,}$ $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Repeating part $y$ must be non-empty …
but can be repeated zero times!

Example: $L = \{0^i1^j \mid i > j\}$

Want to prove: $L = \{0^i 1^j \mid i > j\}$ **is not** a regular language

# Pumping Down

Proof (by contradiction):

contradiction

- Assume: $L$ **is** a regular language
  - So it must satisfy the pumping lemma
  - I.e., all strings ≥ length $p$ are pumpable
- Counterexample = $0^{p+1}1^p$

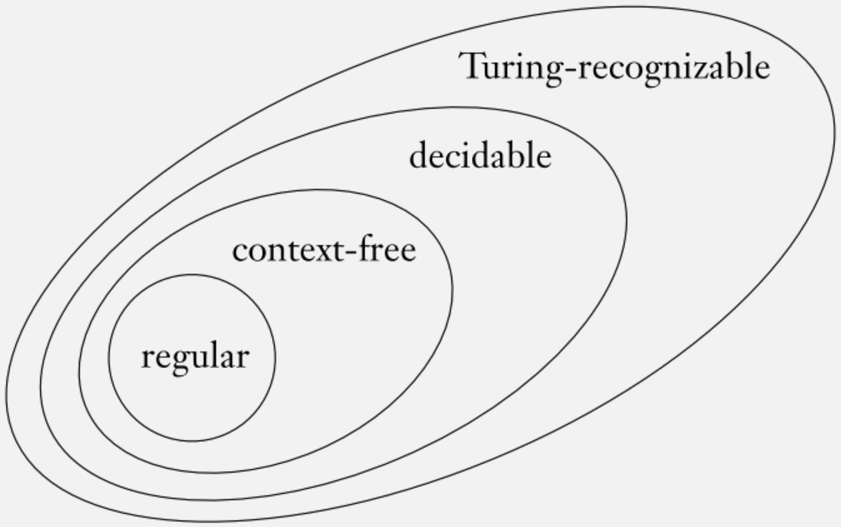
$$\underbrace{00}_{x}\underbrace{\ldots}_{y}\underbrace{0 11 \ldots 1}_{z} \qquad \overbrace{00 \ldots 0}^{p+1 \text{ 0s}}\overbrace{11 \ldots 1}^{p \text{ 1s}}$$

- Choose $xyz$ split so $y$ contains:
  - all 0s
  - (Only possibility, by condition 3)
- Repeat $y$ zero times (**pump down**): produces string with # 0s ≤ # 1s
  - … not in the language $\{0^i 1^j \mid i > j\}$
  - So $\{0^i 1^j \mid i > j\}$ does not satisfy the pumping lemma
  - So it is a not regular language
  - This is a **contradiction** of the assumption!

# Next Time (and rest of the Semester)

- If a language is not regular, then what is it?

- There are many more classes of languages!



# Check-in Quiz 10/11

On gradescope