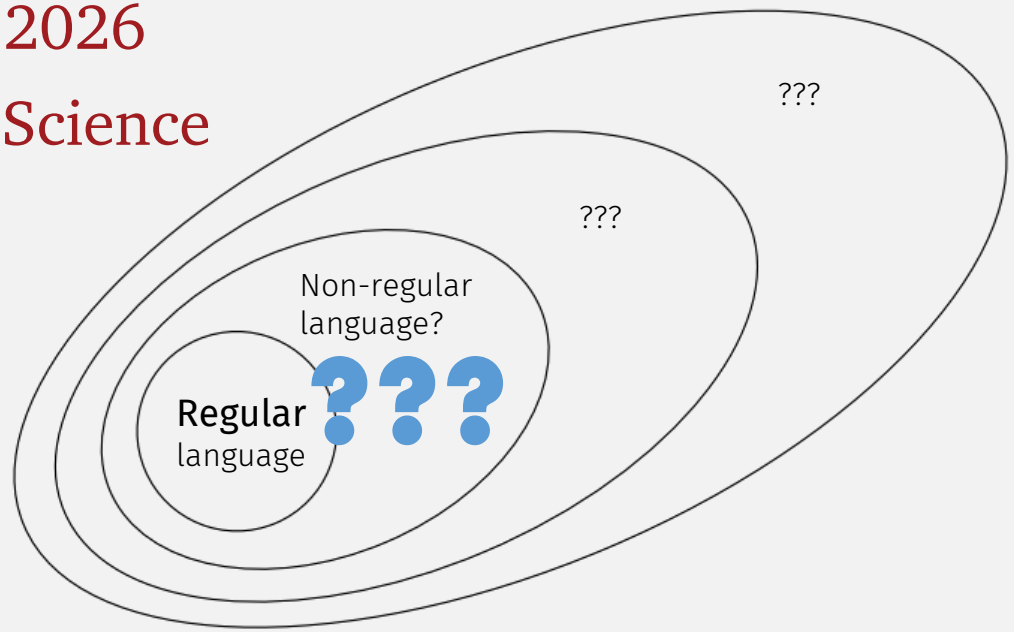
CS 420

Non-Regular Languages

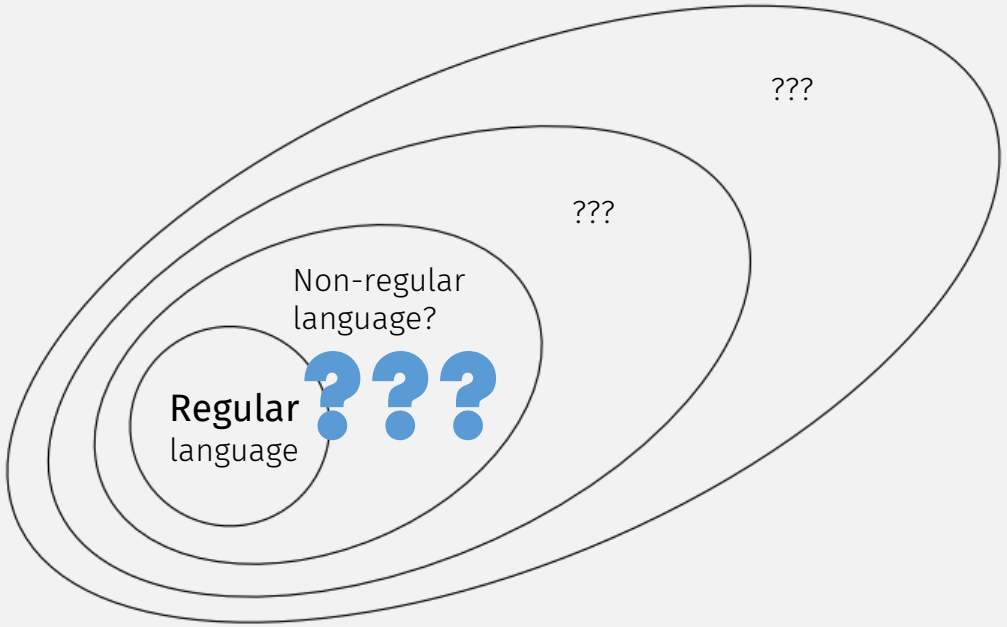
Thursday, October 20, 2026

UMass Boston Computer Science



Announcements

- HW 5
  - Due: **Thu 10/22 12:30pm**



In-class questions preview – Regular or not?

Q1.1 Regular Languages
1 Point

True or False. The Pumping Lemma can be used to prove that a language is a regular language.

Q1.2 non-Regular Languages
1 Point

True or False. The Pumping Lemma can be used to prove that a language is **not** a regular language.

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

So Far: Regular or Not?

IF M IS-A **DFA** Machine AND M **recognizes** L
THEN L IS-A **Regular** Language

L IS-A **Regular** Language

IF AND ONLY IF

N IS-A **NFA** Machine AND N **recognizes** L

L IS-A **Regular** Language

IF AND ONLY IF

R IS-A **Regular** Expression AND R **describes** L

- Many ways to **prove a language is Regular**:
  - **Construct** a **DFA** recognizing it (DefRegLang)
  - **Construct** an **NFA** recognizing it (RegLangIFFNFA ⇐)
  - **Create** a **regular expression** describing it (RegLangIFFRegEx ⇐)
- Bc we proved: **Regular Expression ⇔ NFA ⇔ DFA ⇔ Regular Language**
- But ***not*** **all languages** are **Regular!**
  - E.g., **programming language syntaxes** are **not Regular**
    - **language of all `Python` programs,** or **all HTML/XML pages,** are **not Regular**
  - That means:
    - There is ***no*** **DFA** or **NFA** that:
      - **accepts valid `Python` programs** (and **rejects invalid ones**)
    - And, there is ***no*** **regular expression** that:
      - **describes all valid `Python`** or **HTML programs** (a common mistake)**!**

Someone Who Didn't Pay Atte

RegEx match open tags except XHTML self-cor

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 expressio
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

Someone who paid attention in 420 👍 …

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 cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like mayonnaise in a microwave. 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 has doomed humanity 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 glimpse* of the world of rege**x parsers for HTML will ins**tantly transport a *programmer's consciousness* into a world of ceaseless screaming, he comes, the pestilent slithy regex-infection will **devour your** HTML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fight* he **comes, his** unholy radiance *destroying all enlightenment, HTML tags* ***leaking from your eyes like liquid*** pain, the song of regular expression parsing will ext*inguish the voices of mortal man from the sphere I can see it can you see it it is beautiful* the f`inal snuf`fing *of the lies of Man* ***ALL IS LOST ALL IS*** LOST the *pony he comes* he comes he comes ***the Eldritch*** horror permeates all MY FACE *MY FACE oh god* ***no NOOOOOO*** NO stop *the an*gles `are` **not real ZALGO IS** TONY **THE PONY HE COMES**

ummm … this is getting a little weird

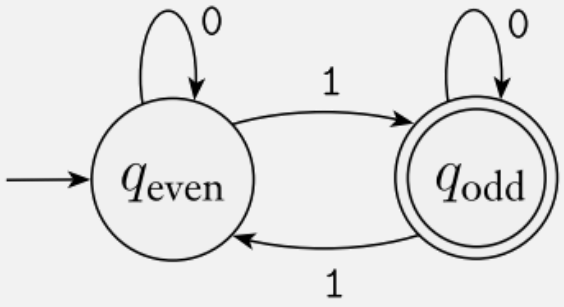
very weird …

?????

Have you tried using an XML parser instead?

hmm … what's this?

Flashback: Designing DFAs or NFAs



- Each state "remembers" information about input
  - E.g., q_{even} = "seen even # of 1s"
    q_{odd} = "seen odd # of 1s"
  - But finite states = finite amount of info storage (and must decide in advance)

- So DFAs can't remember 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 **0s seen,**
  - … one **0 seen,**
  - … two **0s seen** …
- This **language** represents the essence of **many PLs,** e.g., **HTML!**
  - Replace:
    - "**0**" with "**<tag>**" or "**(**"
    - "**1**" with "**</tag>**" or "**)**"

But, **how can we *prove* non-regularness?**

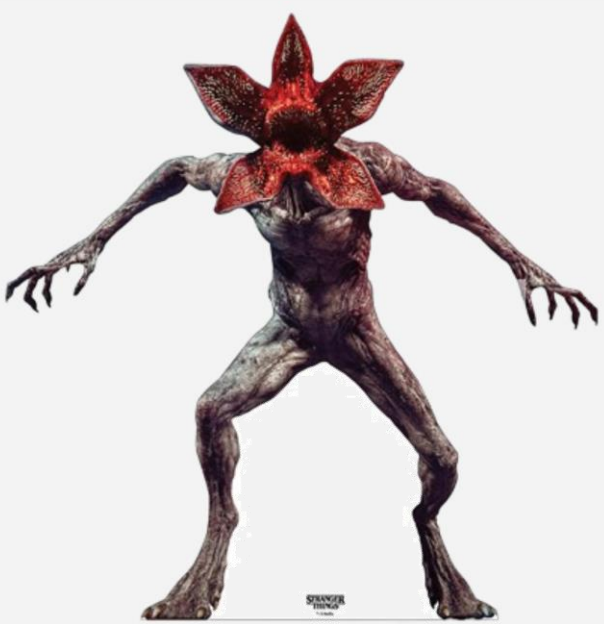
- The Problem: **remembering nestedness**
  - **Need to count arbitrary nesting depths**
    - E.g., **(((…)))**
  - Thus: **most programming language syntax** is **not regular!**

Thought Exercise

(WARNING: **Scary monster about to appear!**)

Prove: Monsters do not exist

???



Proving something not TRUE is **different** (and usually harder) **than proving it** TRUE

It's **sometimes possible,** but often **needs new proof techniques!**

We know how to: **prove** a language is **Regular**
Now, can we: **prove** a language is **not Regular?**

Quantified Logical Statements

- **"Exists"** (Existential)
  - **"Easier"** to **prove TRUE**
    - Just need **one example!**

$\exists x P(x)$ is true when $P(x)$ is true for at least one value of x.

"There **exists** a **natural number** n such that, $n \cdot n = 25$"

n=5

- **"For all"** (Universal)
  - **"Harder"** to **prove TRUE**
    - Need to **prove true for all examples**

$\forall x P(x)$ is true when $P(x)$ is true for all values of x.

"For all natural numbers n, $2 \cdot n = n + n$"

Proof by induction, on natural number n …!

Quantified Logical Statements (this course)

$L = \{ \ldots \}$ IS-A **Regular** Language

Language L is not Regular?

- **"Exists"** (Existential)
  - "Easier" to prove TRUE → There exists one DFA that recognizes L
    - Just need one example!
  - **"Harder" to prove FALSE** → There are no possible DFAs that recognize L
    - Need to prove false for all examples

- **"For all"** (Universal)
  - "Harder" to prove TRUE → For all Regular languages L, L^* IS-A **Regular** Language
    - Need to prove true for all examples
  - **"Easier" to prove FALSE** → For all strings in a regular language …
    - Just need one (counter)example!

Key is finding such a statement about Regular languages!

A Fact (Lemma) About Regular Languages

∀A

THEPUMPINGLEMMA 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. $|xy| \leq p$.

Fact is TRUE for all Regular languages!

This is an "IF X THEN Y" statement

Remember: To ***use*** (i.e., **APPLY**) an "IF X THEN Y" **statement,**
1. **need** to have: **proof of X,**
2. to **get** a: **proof of Y**

THEPUMPINGLEMMA 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. $|xy| \leq p$.

… then we can *conclude* …

Uh … whatever this says …

To *use* THEPUMPINGLEMMA for a language A …

… first need to *prove:* A IS-A **Regular** Language …

Q: Can we *use* THEPUMPINGLEMMA to prove: A IS-A **Regular** Language?

NO (but we already know many other ways to do that!)

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

Equivalence of Conditional Statements

- Yes or No? **"If X then Y" is equivalent to:**
  - "If Y then X" (**converse**) *Seen Previously*
    - No!
  - "If not X then not Y" (**inverse**)
    - No!
  - **"If not Y then not X" (contrapositive)**
    - Yes!

... THEN the language is **not** Regular!

i.e., NOT (*A* IS-A **Regular** Language)

IF-THEN statement

Abbrev: *A* IS-NOT-A **Regular** Language

THEPUMPINGLEMMA

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 equivalent 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

Step 1: **find** a **fact that is true for all regular languages** ...

Step 2: where **the fact can be proven not true!**

Conclusion (new fact)

- P is **not** TRUE

How to: **prove** a **language** is **not regular?**

Fact About Regular Languages: Details

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$, 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. $|xy| \leq p$.

Conditions are on: strings in the language with length $\geq p$

Any regular language satisfies these three conditions!

NOTE:
- Lemma doesn't give an exact p!
- Only that there is *some* string length p …

The exact value of p differs for every regular language

Types of Regular Languages

Finite Languages

Conclusion: THEPUMPINGLEMMA only interesting for **infinite langs**! (which contain strings with repeating parts)

Lemma doesn't say what p is! Just that "there is a p ..."

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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. $|xy| \leq p$.

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

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"}** → **`ab` ∪ `cd`**

- All finite languages are **Regular** Languages!
- (can easily construct **DFA/NFA/Regular Expression** recognizing them)

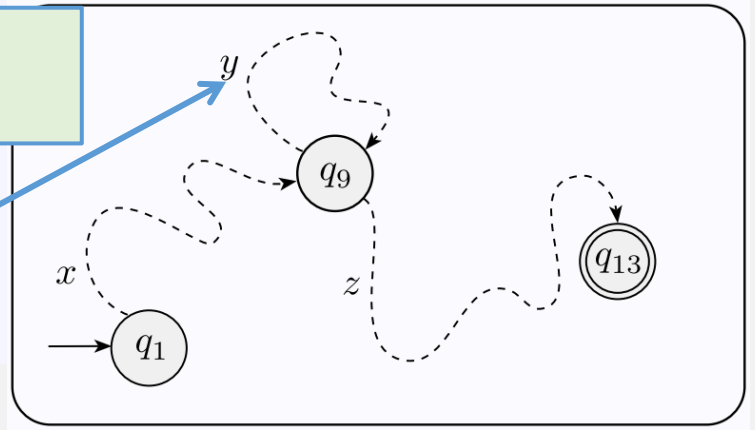
Langs With Strings With Repeatable Parts

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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 requires: "**pumped**" string still in language!

repeatable ("pumpable") part (= repeatable state in DFA!)



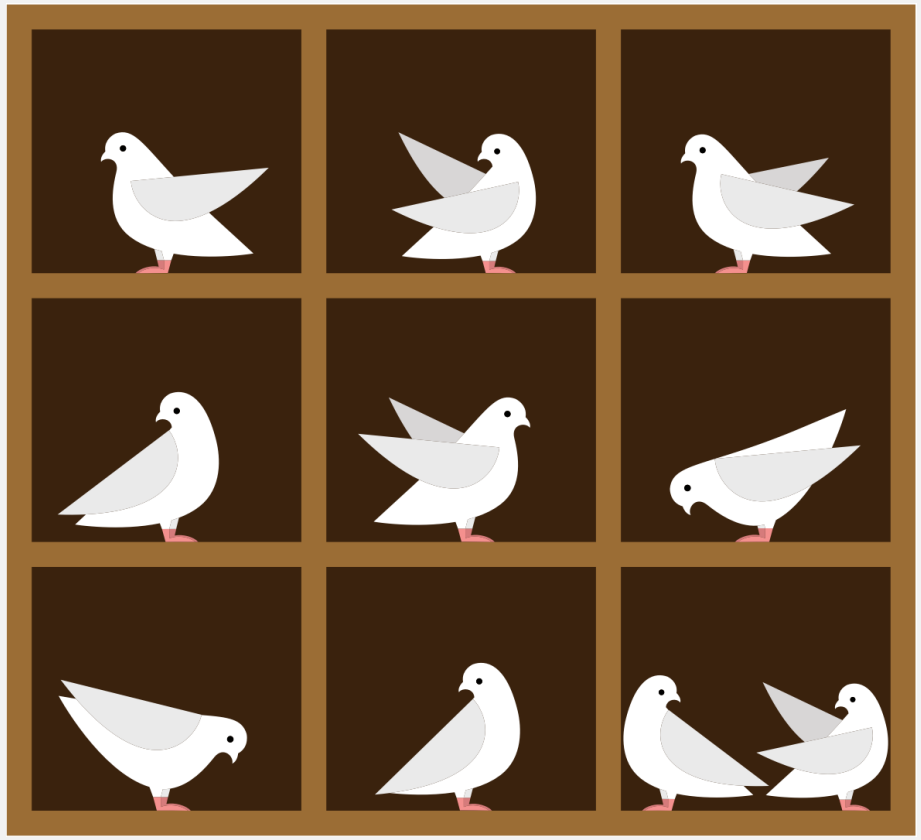
Strings with a **repeatable part** can be **split into 3 parts:**

- x = part before any repeating
- y = repeatable (or "pumpable") part
- z = part after any repeating

DFAs have finite states, so for "long enough" (i.e., length ≥ p) inputs, some state must repeat!

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

The Pigeonhole Principle



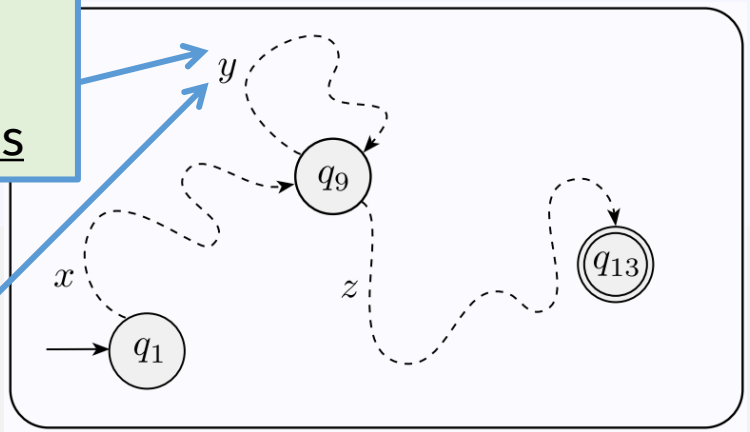
IF # birds > # holes,
THEN there must be > 1
bird in some hole

The Pumping Lemma, a Closer Look

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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. $|xy| \leq p$.

So a **substring** that:
- *can* repeat once,
- *can also* be repeated multiple times



This is the only way for Regular languages to have repeating patterns (KLEENE star)

In essence, ThePumpingLemma is a **theorem** about repeating patterns in regular languages

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

In-class exercise: Infinite Languages

Split the string "010" into **three parts** *xyz*, e.g.
x = "0" y = "1" z = "0"
so that **repeating** (non-empty) y **part any number of times creates** a **new string still in** A

Now do "0110":
x = "0" y = "1" z = "10"

Example: ***infinite*** language A = {"00", "010", "0110", "01110", ...}

Or ...?

(there could be more than one possible splitting)

The Pumping Lemma: Infinite Languages

ThePumpingLemma IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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" $\in A$, so pumping lemma says it's splittable into three parts xyz, e.g. $x = 0$, $y = 1$, $z = 0$

Example: *infinite* language A = {"00", "010", "0110" , "01110", ...}

- It's **Regular** bc it has **Regular** Expression 01^*0

ThePumpingLemma summary:
"All infinite regular languages must have a star in its regular expression"!

... and "pumping" (repeating) middle y part creates a string that is still in the language
- repeat once ($i = 1$): "**010**",
- repeat twice ($i = 2$): "**0110**",
- repeat three times ($i = 3$): "**01110**"

Summary: The Pumping Lemma …

- … states properties that are true for all regular languages
- … specifically, properties about "long enough" strings in reg. langs
- In general, it describes repeating patterns in reg. langs

IMPORTANT:

- ThePumpingLemma cannot prove that a language is **Regular!**
- But … we can use it to prove that a language is **not Regular**

ThePumpingLemma summary:
"All infinite regular languages must have a star in its regular expression"!

… by showing that the repeating pattern is not expressible with a star **Regular** Expression!

IF-THEN statement

… THEN the language is not regular

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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. $|xy| \leq p$.

In other words …

s IS-PUMPABLE

Equivalent (**contrapositive**):
IF any of these are not true …

Contrapositive:
"IF X THEN Y" is equivalent to "IF **not** Y THEN **not** X"

Kinds of Proofs

- Implication (IF-THEN) `mk-IFTHEN-proof`
  - "assume" IF part, "prove" THEN part
- Using IF-THEN Proof `APPLY <some ifthenproof> TO …`
  - Logically infer (i.e., with modus ponens) conclusion from known definitions and assumptions
- Proof by induction `mk-Inductive-proof`
  - Used to prove properties of recursive definitions or functions
- Proof by contradiction  `mk-Contradiction-proof`
  - Proving the contrapositive

How To Do Proof By Contradiction

3 easy steps:

1. Assume: the opposite of the statement to prove

2. Show: the assumption leads to a contradiction

3. Conclude: the original statement must be true

Want to prove: **not** ($\{ 0^n1^n \mid n \geq 0 \}$ IS-A **Regular** Language)

Pumping Lemma: Non-Regularity Example

This repetition pattern cannot be expressed with a star regular expression?

Let B be the language $\{0^n1^n \mid n \geq 0\}$. We use THEPUMPINGLEMMA to prove that B is not regular. The proof is by contradiction.

... by showing that the repeating pattern is not expressible with a star regular expression!

THEPUMPINGLEMMA summary:
"All infinite regular languages must have a star in its regular expression"!

Want to prove: **not** ({ 0^n1^n | $n \geq 0$ } IS-A **Regular** Language)

THEPUMPINGLEMMA IF A IS-A **Regular** Language
THEN p IS-A NatNum
AND
IF $s \in A$ AND $|s| \geq p$
THEN s IS-Pumpable

Assume opposite ...

mk-Contradiction-proof
(with ASSUMPTION = proof of { 0^n1^n | $n \geq 0$ } IS-A **Regular** Language):

Statements

1. p IS-A NatNum
2. IF $s \in$ { 0^n1^n | $n \geq 0$ } AND $|s| \geq p$ THEN s IS-Pumpable

Justifications

FIRST APPLY THEPUMPINGLEMMA **TO** ASSUMPTION

SECOND APPLY THEPUMPINGLEMMA **TO** ASSUMPTION

8. 0^p1^p IS-Pumpable

Counterexample

9. NOT 0^p1^p IS-Pumpable

... Prove a contradiction

10. **Contradiction!** → **mk-AND-proof** Proof8, Proof9

(Proof requires proofs of stmts X AND NOT X!)

Want to prove: **not** ({ 0^n1^n | $n \geq 0$ } IS-A **Regular** Language)

THEPUMPINGLEMMA IF A IS-A **Regular** Language
THEN p IS-A NatNum
AND
IF $s \in A$ AND $|s| \geq p$
THEN s IS-Pumpable

`mk-Contradiction-proof`
`(with` ASSUMPTION = proof of { 0^n1^n | $n \geq 0$ } IS-A **Regular** Language):

	Statements	Justifications
1.	p IS-A NatNum	**`FIRST APPLY`** THEPUMPINGLEMMA **`TO`** ASSUMPTION
2.	IF $s \in \{ 0^n1^n \mid n \geq 0 \}$ AND $\|s\| \geq p$ THEN s IS-Pumpable	**`SECOND APPLY`** THEPUMPINGLEMMA **`TO`** ASSUMPTION
3.	$p \geq 0$	Proof1
4.	$0^p1^p \in \{ 0^n1^n \mid n \geq 0 \}$	Proof3?
5.	$\| 0^p1^p \| = 2p$	**BY** DefLength
6.	$2p \geq p$	**BY** Definition of NatNums
7.	$\| 0^p1^p \| \geq p$	**SUBST** Proof5 **`INTO`** Proof6
8.	0^p1^p IS-Pumpable	**`APPLY mk-AND-proof`** Proof4 Proof7 **`TO`** Proof2
9.	NOT 0^p1^p IS-Pumpable	???
10.	**Contradiction!**	**`mk-AND-proof`** Proof8, Proof9

To make proofs more understandable, will begin to allow combining less imporant steps

(Proof requires proofs of stmts X AND NOT X!)

Want to prove: **not** ({ 0^n1^n | $n \geq 0$ } IS-A **Regular** Language)

THEPUMPINGLEMMA IF A IS-A **Regular** Language
THEN p IS-A NatNum
AND
IF $s \in A$ AND $|s| \geq p$
THEN s IS-Pumpable

`mk-Contradiction-proof`
`(with` ASSUMPTION = proof of { 0^n1^n | $n \geq 0$ } IS-A **Regular** Language):

	Statements	Justifications
1.	p IS-A NatNum	**`FIRST APPLY`** THEPUMPINGLEMMA **`TO`** ASSUMPTION
2.	IF $s \in \{ 0^n1^n \mid n \geq 0 \}$ AND $\|s\| \geq p$ THEN s IS-Pumpable	**`SECOND APPLY`** THEPUMPINGLEMMA **`TO`** ASSUMPTION
3.	$0^p1^p \in \{ 0^n1^n \mid n \geq 0 \}$	**BY** Proof1, Definition of Sets
4.	$\| 0^p1^p \| \geq p$	**BY** DefLength, Definition of **NatNums**
8.	0^p1^p IS-Pumpable	**`APPLY`** Proof2 **`TO mk-AND-proof`** Proof3 Proof4
9.	NOT 0^p1^p IS-Pumpable	???
10.	**Contradiction!**	**`mk-AND-proof`** Proof8, Proof9

(Proof requires proofs of stmts X AND NOT X!)

Want to prove: **not** ({ 0^n1^n | $n \geq 0$ } IS-A **Regular** Language)

THEPUMPINGLEMMA IF A IS-A **Regular** Language
THEN p IS-A NatNum
AND
IF $s \in A$ AND $|s| \geq p$
THEN s IS-Pumpable

mk-Contradiction-proof
(with ASSUMPTION = proof of { 0^n1^n | $n \geq 0$ } IS-A **Regular** Language):

	Statements	Justifications
1.	p IS-A NatNum	**FIRST APPLY** THEPUMPINGLEMMA **TO** ASSUMPTION
2.	IF $s \in$ { 0^n1^n | $n \geq 0$ } AND $\|s\| \geq p$ THEN s IS-Pumpable	**SECOND APPLY** THEPUMPINGLEMMA **TO** ASSUMPTION
3.	$0^p1^p \in$ { 0^n1^n | $n \geq 0$ }	**BY** Proof1, Definition of Sets
4.	$\| 0^p1^p \| \geq p$	**BY** DefLength, Definition of **NatNums**
5.	0^p1^p IS-Pumpable	**APPLY** Proof2 **TO** **mk-AND-proof** Proof3 Proof4
6.	NOT 0^p1^p IS-Pumpable	??? **BY** *NotPumpableAnalysis* **ON** - Pumping Length: p - Language: { 0^n1^n | $n \geq 0$ } - Counterexample String: 0^p1^p
7.	**Contradiction!**	**mk-AND-proof** Proof5, Proof6

(Proof requires proofs of stmts X AND **NOT** X!)

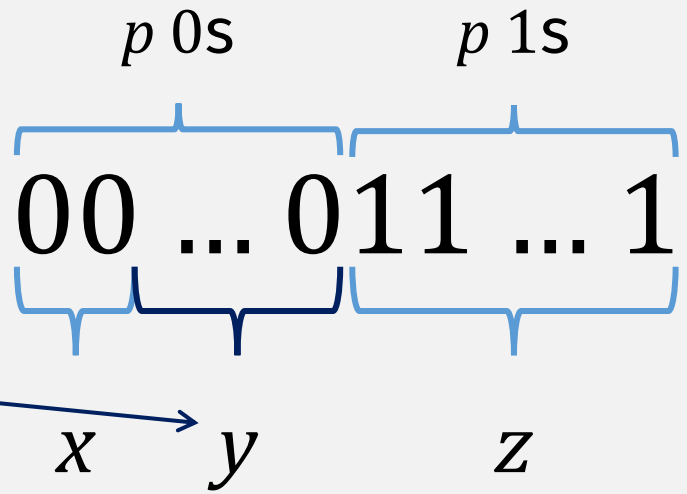
Requires 3 arguments: **counterexample string, Lang, and pumping length p**

Counterexample strings not in lang or shorter than p not allowed!

let *NotPumpableAnalysis* = **`mk-NotPumpable-proof`** (with Pumping Length: p
Language: L
Counterexample String: s
where
$s \in L$ AND $|s| \geq p$
):

NEED TO PROVE:
there is no split into xyz where
- y can be repeated
- and pumped string still in language!

Counterexample = 0^p1^p



Split 1
- Choose xyz split so y contains:
  - all 0s

BUT ...
THEPUMPINGLEMMA 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 ...
  - ... which is not in the language $\{ 0^n1^n \mid n \geq 0 \}$!
  - So 0^p1^p is not pumpable? (not yet!)

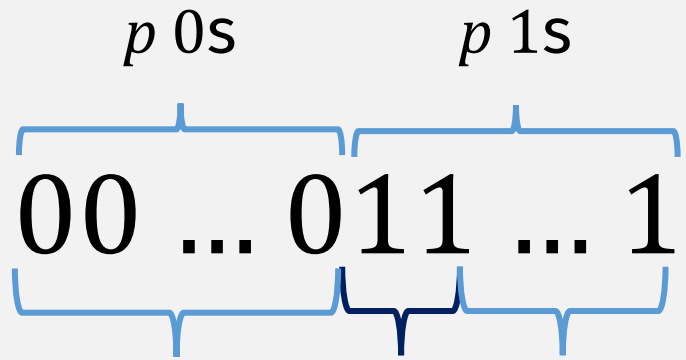
NEED TO PROVE:
s IS-NOT Pumpable

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. $|xy| \leq p$.

s IS-PUMPABLE

Counterexample = 0^p1^p



Split 2

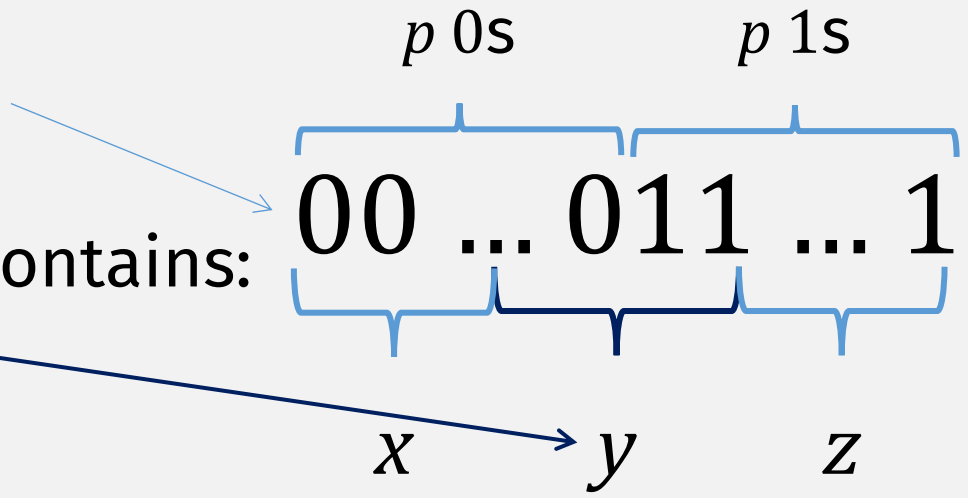
- Choose xyz split so y contains:
  - all 1s

x y z

Is there another way to split into xyz ?

- Is this string pumpable (repeating y produces string still in 0^n1^n)?
  - No!
  - By the same reasoning as in the previous slide

Counterexample = 0^p1^p



Split 3

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

- Is this string pumpable (repeating y produces string still in 0^n1^n)?
  - No!
  - Pumped string will have equal 0s and 1s …
  - But they will be in the wrong order

Did we examine every possible splitting?

Yes! So we proved 0^p1^p IS-NOT Pumpable!

But maybe we didn't have to look at so many cases …

The Pumping Lemma: Condition 3

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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. $|xy| \leq p$.

In other words ...

s IS-PUMPABLE

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

Split 1

p 0s

00 ... 011 ... 1

(this is the only case we needed to consider!)

y must be in here!

The Pumping Lemma: Pumping Down

THEPUMPINGLEMMA IF A IS-A **Regular** Language, THEN p IS-A NatNum (the pumping length) AND IF $s \in A$ AND $|s| \geq p$,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. $|xy| \leq p$.

In other words ...

s IS-PUMPABLE

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

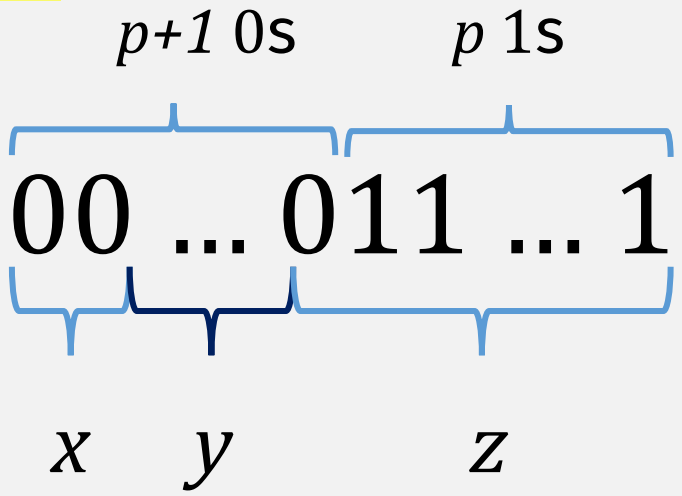
Example: $L = \{0^i1^j \mid i > j\}$

Want to prove: $L = \{0^i 1^j \mid i > j\}$ IS-NOT-A **Regular** Language

Pumping Down

Proof sketch (by contradiction):

- Assume: L IS-A **Regular** Language
  - So **it must satisfy** THEPUMPINGLEMMA
  - I.e., **all strings ≥ length p are pumpable**
- Counterexample = $0^{p+1}1^p$



- 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 THEPUMPINGLEMMA
  - But **Regular** Languages must be pumpable!
  - This is a **contradiction**!

Road Map for Remainder of Semester

- If a language is not **Regular**, then what is it?

- There are many more classes of languages!

