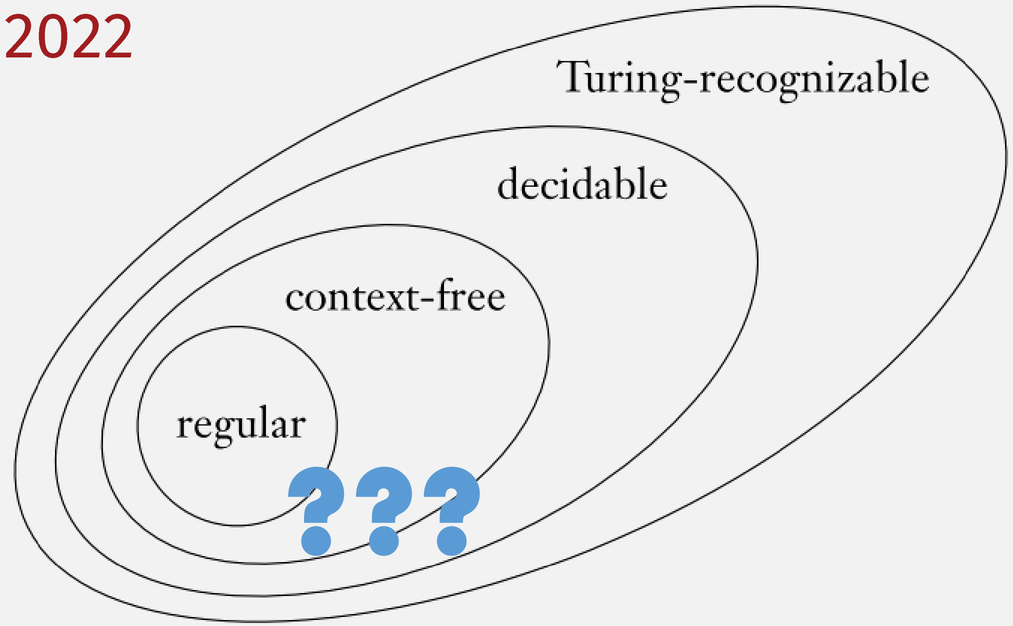
UMB CS 420

Non-Regular Languages

Monday, February 14, 2022



Announcements

- HW 2 due yesterday
- HW 3 released, due Sun 2/20 11:59pm EST
- <u>No class </u>next Monday Feb 2/21

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

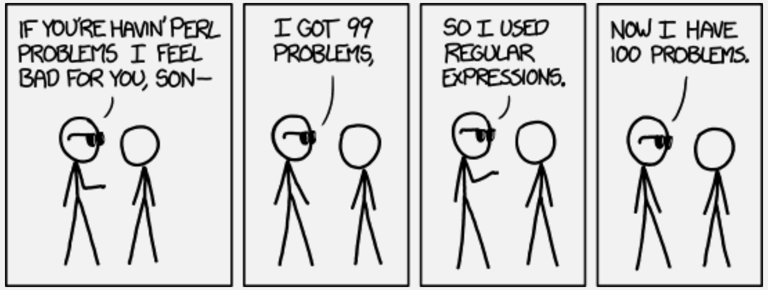
So Far: Regular or Not?

- Many ways to prove that a language is regular:
  - Construct a DFA or NFA (or GNFA) recognizing it
  - Come up with a regular expression describing the language

M ***recognizes language*** A

if $A = \{w|\ M \text{ accepts } w\}$

- But **not all languages are regular!**
  - E.g., HTML / XML (and most PL syntaxes) are not regular languages
  - That means it can't be represented 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

Trying to use regular expressions to recognize HTML language

1553

6572

I need to match all of these opening tags:

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

But not these:

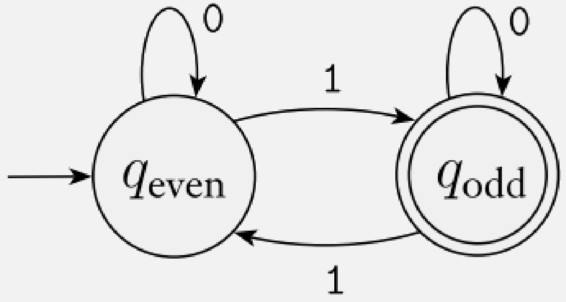
4414

You can't parse [X]HTML with regex. Because HTML can't be parsec
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 glimp*se of the world of rege**x parsers for HTML will ins**tantly transport a p*rogrammer's consciousness* into 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 comes, hi**s unholy radiancé destro*ying all enlightenment, HTML tags* ***leaking from your eyes like liq****uid* pain, the song of re*gular exp*re~~ssion parsing~~ will exti*nguish the voices of mor****tal man from the sp****here*
*I can see it can you see it it is beautiful t*he f`inal snuf`fing o*f the lie****s of Man ALL IS LOST ALL IS L****OST th*e pon*y he com*es he c~~omes he comes~~ **th**e ichor permeates all MY FACE *MY FACE* ᵒh god *no* **NO NOOOO N**Θ stop *the an*gles `are` not rea**l ZALGO IS TONY THE PONY HE COMES**

Have you tried using an XML parser instead?

Flashback: Designing DFAs or NFAs



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

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

A Non-Regular Language

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

- 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`" -> "`<tag>`" or "`(`"
    - "`1`" -> "`</tag>`" or "`)`"
- The problem is tracking the **nestedness**
  - Regular languages cannot count arbitrary nesting depths
  - So most programming language syntax is not regular!

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

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, all regular languages satisfy these 3 conditions!

This lemma describes a property that all regular languages have.

Note: this lemma **cannot be used to prove that a language is a regular language**!
(but we already know how to do that anyways)

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. $|xy| \leq p$.

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

Specifically, these conditions apply to strings in the language <u>longer than length p</u>

Lemma doesn't tell you an exact p!
(just that there must exist "some" p)

The Pumping Lemma: Finite Lang

So the pumping lemma is <u>only interesting for **infinite langs**</u>! (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 tell us what p is! Just that there <u>is</u> one.

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

What could p be?
Length of longest string + 1

\# strings in the language with at least length p? **None!**

Therefore, <u>all</u> strings with length at least p satisfy the pumping lemma conditions! ☺

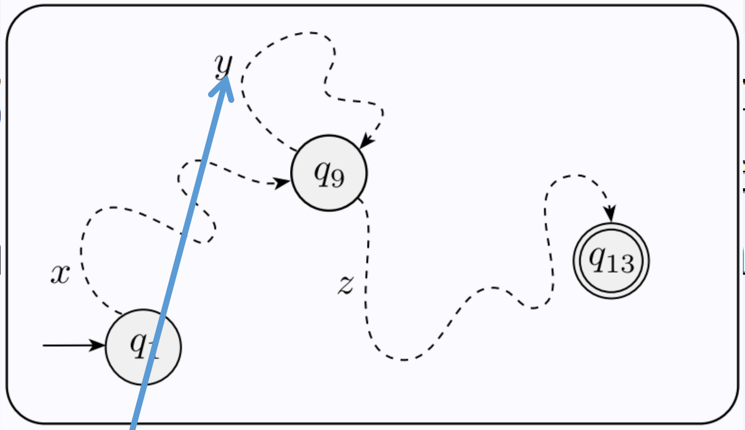
<u>Example</u>: a finite language {"ab", "cd"}

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

The Pumping Lemma, a Closer Look

Pumping lemma If A is a regular lang nber p (the pumping length) where if s is any string in en s may be divided into three pieces, $s = xyz$, satisfyin s:

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



strings of length p = "long enough":
should have a repeatable ("pumpable") part;
where "pumped" string is still in the language

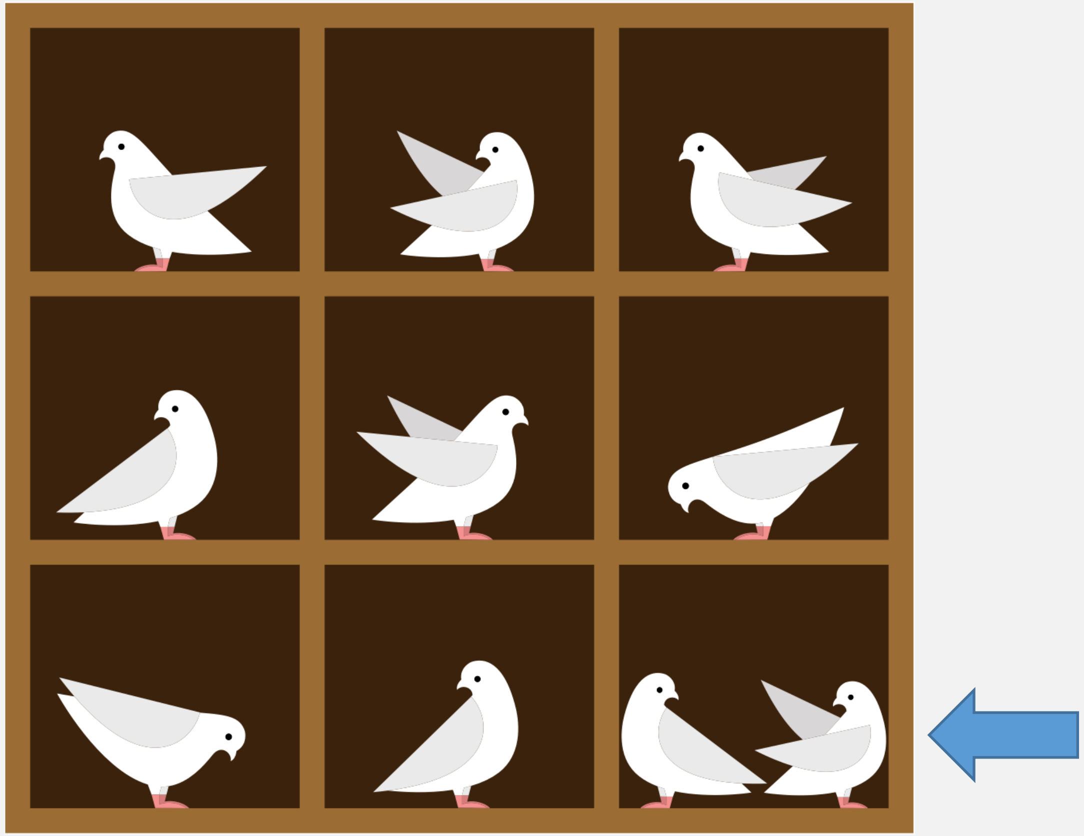
Strings that have a repeatable part can be split into:

- x = the part before any repeating
- y = the repeated part
- z = the part after any repeating

This makes sense because DFAs have a finite number of states, so for "long enough" (i.e., some length p) inputs, some state must repeat

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

The Pigeonhole Principle

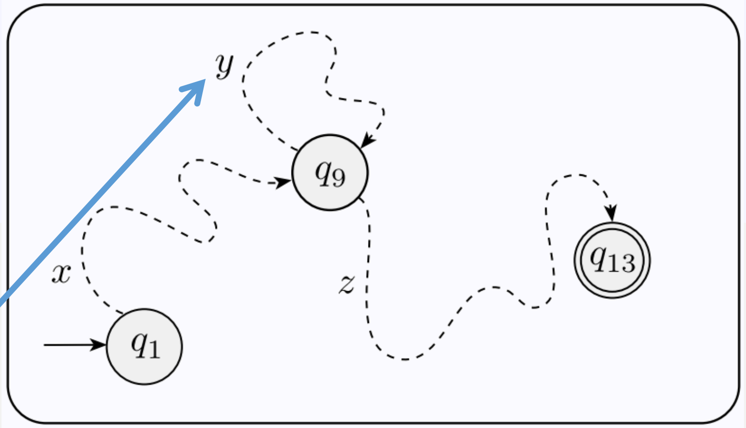


The Pumping Lemma, a Closer Look

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. $|xy| \leq p$.

But repeating once also means repeating <u>any number of times</u> is possible



This is the only way for regular languages to repeat (Kleene star)

In essence, **the pumping lemma is a <u>theorem about what repeated patterns regular languages can have</u>**

e.g., "**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

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

- Language is regular bc it's described by the regular expression 01*0
- Notice that the middle part is "pumpable"!
- E.g., "010" in the language can be split into three parts: $x = 0$, $y = 1$, $z = 0$
  - Any pumping (repeating) of the middle 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 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

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>

Pumping Lemma: Proving Non-Regularity

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 ...

This is the essence of <u>"proof by contradiction"</u>

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

Kinds of Mathematical Proof

- Proof by construction
  - Construct the object in question

- Proof by induction
  - Use to prove properties of recursive definitions or functions

- Proof by contradiction
  - Proving the contrapositive

How To Do Proof By Contradiction

- Assume the opposite of the statement to prove
- Show that the assumption leads to a contradiction
- Conclude 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 length p or longer 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):

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

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

$$x \quad y \quad z$$

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
  - Which is not in the language 0^n1^n
  - This means that 0^p1^p is not pumpable (according to pumping lemma)
  - Which means that that 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 or longer are pumpable
- Counterexample = 0^p1^p
- Choose xyz split so y contains:
  - all 1s

$$\overbrace{\underbrace{00\ldots0}_{x}}^{p\text{ 0s}}\overbrace{\underbrace{11}_{y}\underbrace{\ldots1}_{z}}^{p\text{ 1s}}$$

- 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 length p or longer are pumpable
- Counterexample = 0^p1^p
- Choose xyz split so y contains:
  - both 0s and 1s

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

$$\underbrace{00\ldots}_{x}\underbrace{011}_{y}\underbrace{\ldots1}_{z}$$

- 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**!

Did we examine every possible splitting?

Yes! QED

But maybe we didn'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. $|xy| \leq p$.

Repeating part y ...
must be in the first p characters!

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

$$0\underbrace{0\ldots0}_{}11\ldots1$$

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: $L = \{0^i1^j \mid i > j\}$ **is not** a regular language

Pumping Down

Proof (by contradiction):

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

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

$$\underbrace{0}_{x}\underbrace{0\ldots}_{y}\underbrace{011\ldots1}_{z}$$

- Choose xyz split so y contains:
  - all 0s
  - (Only possibility, by condition 3)

- Repeat y zero times (**pump down**): produces string with 0s ≤ 1s
  - Which is not in the language $\{0^i1^j \mid i > j\}$
  - This means that $\{0^i1^j \mid i > j\}$ does not satisfy the pumping lemma
  - Which means that that it is a not regular language
  - This is a **contradiction** of the assumption!

Check-in Quiz 2/14

On gradescope