UMB CS 420

Non-Regular Languages

Wednesday March 6, 2024
Announcements

• HW 4 out
  • Due Mon 3/18 12pm EST (noon)
  • (After spring break)

• Problem 4, Part 2c Update:
  • Prove the statement for
    • 1 base case
    • 1 recursive case
So Far: Regular or Not?

- 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

- Bc we proved: Regular Expression $\iff$ NFA $\iff$ DFA $\iff$ 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)!

A language is a set of strings.

$M$ recognizes language $A$ if $A = \{w \mid M$ accepts $w\}$
Someone Who Did Not Parse

RegEx match open tags except XHTML self-closing

I need to match all of these opening tags:

- \(<p>\)
- \(<a href=“foo”>\)

But not these:

- 1553
- 6572

You can't parse [X]HTML with regex. Because HTML can't be parsed with regex. Regex is not a tool that can be used to correctly parse HTML. As I have mentioned before, using regex to parse HTML-and-regexp questions here so many times before, the use of regex will allow you to consume HTML. Regular expressions are a tool that is highly sophisticated to understand the constructs employed by HTML, HTML regular language and hence cannot be parsed by regular expression.

Someone who paid attention in 420 ...

Have you tried using an XML parser instead?

ummm ... this is getting a little weird

very weird ...

???

lost all is lost

zalgo is the pony he comes

hmm ... what's this?
Flashback: Designing DFAs or NFAs

- Each state “remembers” information about input
  - E.g., $q_{\text{even}} = \text{“seen even # of 1s”}$
  - $q_{\text{odd}} = \text{“seen odd # of 1s”}$
  - 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

\[ L = \{ \theta^n \, 1^n \mid n \geq 0 \} \]

- A DFA recognizing \( L \) would require infinite states! (impossible)
  - States representing zero \( \theta \)s seen, one \( \theta \) seen, two \( \theta \)s, ...

- This language is the same as many PLs, e.g., HTML!
  - To better see this replace:
    - “\( \theta \)” with “<tag>“ or “(“
    - “1” with “</tag>” or “)“

- The Problem: remembering nestedness
  - Need to count arbitrary nesting depths
    - E.g., if { if { if { ... } } } }
  - Thus: most programming language syntax is not regular!

So, how can we prove non-regularness?
Prove: Spider-Man does not exist

In general, proving something **not true** is different (and harder) than proving it **true**.

In some cases, it’s possible, but typically requires **new proof techniques**.

We know how to: prove a language is **regular**
Can we: prove a language is **not regular**?

**YES!** but requires a new proof technique!

Step 1: find a fact that is true for all regular languages...
A Fact (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. $|xy| \leq p$.

**Remember:** To *use* an “If $X$ then $Y$” statement,
1. prove $X$ is true,
2. *conclude* that $Y$ is true

This is an “If $X$ then $Y$” statement
Flashback: The Modus Ponens Inference Rule

If we know these statements are true ... 
- If $P$ then $Q$
- $P$

Then we also know this statement is true ... 
- $Q$
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. $|xy| \leq p$.

... then we can conclude ...

Uh ... whatever this says ...

To use The **Pumping lemma** for a language $A$ ...

... first prove that $A$ is a regular language ...

Q: Can we use The **Pumping lemma** to prove that a language is regular?

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)
    - No!

  - “If not $X$ then not $Y$” (inverse)
    - No!

  - “If not $Y$ then not $X$” (contrapositive)
    - Yes!
**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 $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

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

... then the language is not regular!

**Contrapositive:**
“If $X$ then $Y$” is equivalent to “If not $Y$ then not $X$”
Logical Inference Rules

**Modus Ponens** (known facts)

- **Premises**
  - 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

---

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

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

---

How to: prove a **language is not regular**?
Fact 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. $|xy| \leq p$.

**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 Pumping Lemma: Finite 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. $|xy| \leq p$.

So finite languages (specifically, all strings in the language “of length at least $p$”) must satisfy these conditions (whatever they are).

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

**Example**: a finite language \{“ab”, “cd”\}

- All finite languages are regular!
- (can easily construct DFA/NFA/Regular Expression recognizing them)
Langs With 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 $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

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

- $x =$ part before any repeating
- $y =$ repeated (or “pumpable”) part
- $z =$ part after any repeating

**DFAs have finite states, so for “long enough” (i.e., length $\geq p$) 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

**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 $i \geq 0$, $xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

So a substring that *can* repeat once, can also be repeated *any number of times*.

This is the only way for regular languages to have repeating patterns (Kleene star).

In essence, the Pumping lemma is a theorem about repeating patterns in regular languages.

“long enough length” = $p = \# \text{states} + 1$ (some state must repeat)
In-class exercise: Infinite Languages

Example: infinite language $A = \{“00”, “010”, “0110”, “01110”, …\}$
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. $|xy| \leq p$.

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 $A = \{ \text{"00"}, \text{"010"}, \text{"0110"}, \text{"01110"}, \ldots \}$

- It’s regular bc it has regular expression $01^*0$

**Pumping lemma** summary:

“All infinite regular languages must have a star in its regular expression”!
**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:**
- The Pumping lemma **cannot prove** that a language is regular!
- But ... we **can use it to prove** that a language is **not regular**

**Pumping lemma 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 \( X \) then \( Y \)

If \( \neg Y \) then \( \neg X \)

**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 \( i \geq 0 \), \( xy^i z \in A \),
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 \( X \)”. 
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. **Assume:** the opposite of the statement to prove
2. **Show:** the assumption leads to a **contradiction**
3. **Conclude:** the original statement must be true
Pumping Lemma: Non-Regularity Example

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

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

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

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 pumppable
- **Counterexample** = $0^p1^p$

Now we must find a contradiction...

We must show that there is no possible way to split this string to satisfy the conditions of the pumping lemma!

Reminder: Pumping lemma says:
- all strings $0^n1^n \geq$ length $p$ are splittable into $xyz$ where $y$ is pumppable
- So find string $\geq$ length $p$ that is not splittable into $xyz$ where $y$ is pumppable
Want to prove: $0^n1^n$ is not a regular language

Possible Split: $y = \text{all } 0$s

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** = $0^p1^p$

- **Choose** $xyz$ split so $y$ contains:
  - all $0$s

- **Pumping $y$:** produces a string with more $0$s than $1$s
  - ... 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? (contraposition)
  - This is a contradiction of the assumption?

---

**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

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

So the proof is not done!

Is there another way to split into $xyz$?
Possible Split: $y = \text{all 1s}$

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** = $0^p1^p$

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

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

- **Counterexample** = $0^p1^p$

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

  \[ x \rightarrow y \rightarrow z \]

  With $y$ containing:

  \[
  \begin{align*}
  00 & \ldots 011 & \ldots 1 \\
  p \hspace{0.5cm} 0s & \hspace{0.5cm} \hspace{0.5cm} p \hspace{0.5cm} 1s
  \end{align*}
  \]

- **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**: 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. $|xy| \leq p$.

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

$p$ 0s

00 ... 011 ... 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 $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Repeating party 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

**Proof (by contradiction):**

• **Assume: \( L \) is a regular language**
  - So it must satisfy the pumping lemma
  - I.e., all strings \( \geq p \) 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 \( \leq \# 1s \)
  - ... not in the language \( \{0^i1^j \mid i > j \} \)
  - So \( \{0^i1^j \mid i > j \} \) 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!
Submit in-class work 3/6

On gradescope