Non-Regular Languages and The Pumping Lemma

- Not every language is a regular language.
- However, there are some rules that say "if these languages are regular, so is this one derived from them.
- There is also a powerful technique -- the pumping lemma -- that helps us prove a language not to be regular.
- Key tool: Since we know RE's, DFA's, NFA's, NFA-ε's all define exactly the regular languages, we can use whichever representation suits us when proving something about a regular language.

The Pumping Lemma

If $L$ is a regular language, then there exists a constant $n$ such that every string $w$ in $L$, of length $n$ or more, can be written as $w = xyz$, where:
- $0 < |y|$
- $|xy| \leq n$
- For all $i \geq 0$, $xy^iz$ is also in $L$

• Note $y^i = y$ repeated $i$ times; $y^0 = \epsilon$.

Intuitive Explanation

The automaton below has $n$ states and no loops. Expressed in terms of $n$, what is the longest string this automaton can accept?

Generally, in an automaton (graph) with $n$ states (vertices), any "walk" of length $n$ or greater must repeat some state (vertex)--that is, it must contain a cycle.
Proof of Pumping Lemma

• Since we claim \( L \) is regular, there must be a DFA \( A \) such that \( L = L(A) \).
• Let \( A \) have \( n \) states; choose this \( n \) for the pumping lemma
• Let \( w \) be a string of length \( \geq n \) in \( L \), say \( w = a_1a_2 \ldots a_m \), where \( m \geq n \).
• Let \( q_i \) be the state \( A \) is in after reading the first \( i \) symbols of \( w \).
  - \( q_0 \) = start state, \( q_1 = \delta(q_0, a_1) \), \( q_2 = \delta(q_0, a_1a_2) \), etc.

Since there are only \( n \) different states, two of \( q_i, q_j \) must be the same; say \( q_i = q_j \) where \( 0 \leq i < j \leq n \).
• Let \( x = a_1 \ldots a_i \); \( y = a_{i+1} \ldots a_j \); \( z = a_{j+1} \ldots a_m \).
• Then by repeating the loop from \( q_i \) to \( q_j \) with label \( a_i \ldots a_j \) zero times, once, or more, we can show that \( xy^iz \) is accepted by \( A \).

Example

DFA with 6 states that accepts an infinite language

Any string of length 6 or more contains a circuit

Some strings with length < 6 also contain a circuit (bbaa)

Path that this string takes through the DFA

\( b \ bba \ baba \)

\( x \ y \ z \)

1. \( x \) part: goes from start state to beginning of the first circuit
2. \( y \) part: circuit
3. \( z \) part: “the rest”
• PL gets its name because the repeated string is "pumped"
  – Note that because of the nature of FAs, we cannot control the number of times it is pumped
  – So, a regular language with strings of length $\geq n$ is always infinite!
• PL is only interesting for infinite languages
  – but works for finite languages, which are always regular--for finite languages $n$ is larger than the longest string, so nothing can be pumped

**PL Use**

• We use the PL to show a language $L$ is not regular.
  – Start by assuming $L$ is regular.
  – Then there must be some $n$ that serves as the PL constant.
    • We may not know what $n$ is, but we can work the rest of the "game" with $n$ as a parameter.
  – We choose some $w$ that is known to be in $L$.
    • Typically, $w$ depends on $n$.

**Example**

• Applying the PL, we know $w$ can be broken into $xyz$, satisfying the PL properties.
• Again, we may not know how to break $w$, so we use $x$, $y$, $z$ as parameters.
• We derive a contradiction by picking $i$ (which might depend on $n$, $x$, $y$, and/or $z$) such that $xy^iz$ is not in $L$.

• Consider the language $a^ib^j$
  • This language is not regular!
  • Intuitive explanation:
    – Imagine an FA to accept this language
    – Since the number of $a$'s must be equal to the number of $b$’s, must have some way to remember how many $a$’s were seen, and accept if the rest of the string contains the same number of $b$’s
How many states are needed?

Using the PL to prove $L = a^nb^n$ is not regular

- Suppose $L$ is regular. Then there is a constant $n$ satisfying the PL conditions.
- Consider the string $w = a^nb^n$. Note that the string is defined in terms of $n$.
- Then $w = xyz$, where $|xy| \leq n$ and $y \neq \epsilon$, and we can break this string into $xyz$ where for any $j \geq 0$ $xy^jz$ is in $L$.
- But because $|xy| \leq n$ and $|y| > 0$, the string $y$ has to consist of as only. So no matter what segment of the string $xy$ covers, pumping $y$ adds to the number of as and hence there are more as than bs.
- There is NO WAY to segment $w$ into $xyz$ such that pumping will not lead to a string that is not in the language!
- CONTRADICTION! $L$ is therefore not regular.

Important point

- It is necessary to show there is no segmentation of the chosen string that won’t lead to a contradiction.
  - This means considering every possible mapping of $xy$ onto the first $n$ symbols in the chosen string.
  - We chose our string to make this easy, since very possible segmentation consists of as only.
  - Pumping therefore disrupts the equivalence of the number of as and bs.

Example

- Consider the set of strings of a’s whose length is a square; formally, $L = \{a^i \mid i \text{ is a square}\}$.
  - We claim $L$ is not regular.
  - Suppose $L$ is regular. Then there is a constant $n$ satisfying the PL conditions.
    - Consider $w = a^n$, which is surely in $L$.
    - Then $w = xyz$, where $|xy| \leq n$ and $y \neq \epsilon$. 

– By PL, \(xyyz\) is in \(L\).
– The length of \(xyyz\) is greater than \(n^2\) and no greater than \(n^2 + n\). (Why?)
– However, the next perfect square after \(n^2\) is \((n+1)^2 = n^2 + 2n + 1\).
– Thus, \(xyyz\) is not of square length and is not in \(L\).
– Since we have derived a contradiction, the only unproven assumption -- that \(L\) is regular -- must be at fault, and we have a "proof by contradiction" that \(L\) is not regular.

The PL "game"

• Goal: win the PL game against our opponent by establishing a contradiction of the PL, while the opponent tries to foil us.

Four steps:
1. The number of states in the automaton is \(n\).
   Note that we don't have to know what \(n\) is, since we use the variable to define our string.
2. Given \(n\), we pick a string \(w\) in \(L\) of length equal to or greater than \(n\).
   • We are free to choose any \(w\), subject to \(w \in L\) and \(|w| \geq n\).
   • We usually define the string in terms of \(n\).
3. Our opponent chooses the decomposition \(xyz\), subject to \(|xy| \leq n\), \(|y| \geq 1\).
4. We try to pick \(i\) (the power factor in \(xy^iz\)) in such a way that the pumped string \(w_i\) is not in \(L\).
   • If we can do so, we win the game!

Example 1
\[\Sigma = \{a, b\}; L = \{ww^k \mid w \in \Sigma^*\}\]
– Whatever \(n\) is, we can always choose \(w\) as follows:

<table>
<thead>
<tr>
<th>a…ab…bb…ba…a</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n) (n) (n) (n)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

– Because of this choice and the requirement that \(|xy| \leq n\), the opponent is restricted in step 3 to choosing a \(y\) that consists entirely of \(a\).
– In step 4, we use \(i=2\). The string \(xy^2z\) has more \(a\)'s on the left than on the right, so it cannot be of form \(ww^k\). So \(L\) is not regular.
Example 2

$L = \{w | w \text{ has an equal number of } 1\text{'s and } 0\text{'s}\}$
- Given $n$, we choose the string $(01)^n$
- We need to show splitting this string into $xyz$ where $xy'z$ is in $L$ is impossible...

But it is possible!
• If $x = \varepsilon$, $y = 01$, and $z = (01)^{n-1}$, $xy'z$ is in $L$ for every value of $i$.

Are we out of luck?

First law of PL use:
If your string does not succeed, try another!
- Let's try $1^n0^n$.
- Again, we need to show splitting this string into $xyz$ where $xy'z$ is in $L$ is impossible...

But it is possible!
– If $x$ and $z$ are the empty string and $y$ is $1^n0^n$, then $xy'z$ always has an equal number of $0$'s and $1$'s.

Are we still in trouble?

Not this time…
• … the PL says that our string has to be divided so that $|xy| \leq n$ and $|y|$.
• If $|xy| \leq n$ then $y$ must consist only of $0$'s, so $xyyz \not\in L$.

Contradiction! We win!

Example 3

$L = \{ww | w \in \Sigma^*\}$
• We choose the string $a^nba^n$, where $n$ is the number of states in the FA. We now show that there is no decomposition of this string into $xyz$ where for any $j \geq 0$ $xy^jz$ is in $L$.
• Again, it is crucial that the PL insists that $|xy| \leq n$, because without it we could could pump the string if we let $x$ and $z$ be the empty string.
• With this condition, it's easy to show that the PL won't apply because $y$ must consist only of $a$'s, so $xyyz$ is not in $L$. 
• In the previous example as before, the choice of string is critical: had we chosen $a^n a^n$ (which is a member of $L$) instead of $a^n b a^n b$, it wouldn't work because it can be pumped and still satisfy the PL.

MORAL
Choose your strings wisely.

Example 4
$L = \{0^i 1^j \mid i > j\}$

• Given $n$, choose $s = 0^{n+1}1^n$.
• Split into $xyz$ … etc.
• Because by the PL $|xy| \leq n$, $y$ consists only of 0's.
• Is $xyyz$ in $L$?

• The PL states that $xy^iz$ is in $L$ even when $i = 0$.
• So, consider the string $xy^0z$
  – Removing string $y$ decreases the number of 0's in $s$
  – $s$ has only one more 0 than 1
  – Therefore, $xz$ cannot have more 0's than 1's, and is not a member of $L$.

Contradiction!
This strategy is called “pumping down”

Example 5
$L = \{a^i \mid i \text{ is prime}\}$

• Let $n$ be the pumping lemma value and let $k$ be a prime greater than $n$.
• If $L$ is regular, PL implies that $a^k$ can be decomposed into $xyz$, $|y| > 0$, such that $xy^iz$ is in $L$ for all $i \geq 0$.
• Assume such a decomposition exists.
• The length of $w = xy^kz$ must be a prime if $w$ is in $L$. But
  \[
  \text{length}(xy^kz) = \text{length}(xyz) + k \times \text{length}(y) = k + k \times \text{length}(y) = k \times (1 + \text{length}(y))
  \]
  The length of $xy^{k+1}z$ is therefore not prime, since it is the product of two numbers other than 1. So $xy^{k+1}z$ is not in $L$.
• Contradiction!
Example 6

\[ L = \{ w \in \Sigma^* \mid n_a(w) > n_b(w) \} \]

- Let \( n \) be the pumping lemma constant. Then if \( L \) is regular, PL implies that \( s = b^n a^{n+1} \) can be decomposed into \( xyz \), \( |y| > 0 \), \( |xy| \leq n \), such that \( xy^iz \) is in \( L \) for all \( i \geq 0 \).
- Since the length of \( xy \leq n \), there are three ways to partition \( s \):
  1. \( y \) consists of all \( a \)'s
     - Pumping \( y \) will lead to a string with more than 3 \( a \)'s — not in \( L \)
  2. \( y \) consists of all \( b \)'s
     - Pumping \( y \) will lead to a string with more than \( m \) \( b \)'s, and leave the number of \( c \)'s untouched, such that there are no longer 3 fewer \( c \)'s than \( b \)'s — not in \( L \)
  3. \( y \) consists of \( a \)'s and \( b \)'s
     - Pumping \( y \) will lead to a string with \( b \)'s before \( a \)'s, — not in \( L \)
- There is no way to partition \( a^3 b^m c^{m-3} \) so that pumped strings are still in \( L \).

Contradiction!

Example 7

\[ L = \{ a^3 b^m c^{m-3} \mid m > 3 \} \]

- Let \( n \) be the pumping lemma constant. Then if \( L \) is regular, PL implies that \( s = a^3 b^m c^{m-3} \) can be decomposed into \( xyz \), \( |y| > 0 \), \( |xy| \leq n \), such that \( xy^iz \) is in \( L \) for all \( i \geq 0 \).
- Since the length of \( xy \leq n \), there are three ways to partition \( s \):
  1. \( y \) consists of all \( a \)'s
     - Pumping \( y \) will lead to a string with more than 3 \( a \)'s — not in \( L \)
  2. \( y \) consists of all \( b \)'s
     - Pumping \( y \) will lead to a string with more than \( m \) \( b \)'s, and leave the number of \( c \)'s untouched, such that there are no longer 3 fewer \( c \)'s than \( b \)'s — not in \( L \)
  3. \( y \) consists of \( a \)'s and \( b \)'s
     - Pumping \( y \) will lead to a string with \( b \)'s before \( a \)'s, — not in \( L \)
- There is no way to partition \( a^3 b^m c^{m-3} \) so that pumped strings are still in \( L \).

Contradiction!

Remember

- You need to find only ONE string for which the PL does not hold to prove a language is not regular
- But you must show that for ANY decomposition of that string into \( xyz \) the PL holds
  - This sometimes means considering several different cases

The Pumping Lemma Poem

Any regular language \( L \) has a magic number \( p \)
And any long-enough word in \( L \) has the following property:
Among its first \( p \) symbols is a segment you can find
Whose repetition or omission leaves \( x \) among its kind.

So if you find a language \( L \) which fails this acid test,
And some long word you pump becomes distinct from all the rest,
By contradiction you have shown that language \( L \) is not
A regular guy, resilient to the damage you have wrought.

But if, upon the other hand, \( x \) stays within its \( L \),
Then either \( L \) is regular, or else you chose not well.
For \( w \) is \( xy^iz \), and \( y \) cannot be null,
And \( y \) must come before \( p \) symbols have been read in full.

As mathematical postscript, an addendum to the wise:
The basic proof we outlined here does certainly generalize.
So there is a pumping lemma for all languages context-free,
Although we do not have the same for those that are r.e.
Proving a language non-regular without the pumping lemma

- The pumping lemma isn’t the only way we can prove a language is non-regular

- Other techniques:
  - show that the desired DFA would require infinite states to model the intended language
  - use closure properties to relate to other non-RL languages

DFA Method

Consider the language \( \{a^i b^i \mid i \geq 0\} \) and a DFA to recognize it

- For any \( i \), let \( a_i \) be the state entered after processing \( a^i \), i.e., \( \delta(q_0, a^i) = a_i \).
- Consider any \( i \) and \( j \) such that \( i \neq j \).
  - \( \hat{\delta}(q_0, a_i b^i) \neq \hat{\delta}(q_0, a_j b^i) \), since the former is accepting, and the latter is rejecting.
  - \( \hat{\delta}(q_0, a_i b^i) = \hat{\delta}(\hat{\delta}(q_0, a_i), b^i) = \hat{\delta}(a_i, b^i) \), by definition of \( \hat{\delta} \) and definition of \( a_i \), respectively.
  - \( \hat{\delta}(q_0, a_j b^i) = \hat{\delta}(\hat{\delta}(q_0, a_j), b^i) = \hat{\delta}(a_j, b^i) \), by the same reasoning.

- Since \( a_i \) and \( a_j \) lead to different states on the same input, \( a_i \neq a_j \).
- Since \( i \) and \( j \) were arbitrary, and since there are an infinite number of ways to pick them, there must be an infinite number of states.
- Thus, there is no DFA to recognize this language, and the language is non-regular.
Closure Properties

• Certain operations on regular languages are guaranteed to produce regular languages
• Closure properties can also be used to prove a language non-regular (or regular)

Regular languages are closed under common set operations

• Union : \( L_1 \cup L_2 \)
• Intersection : \( L_1 \cap L_2 \)
• Concatenation : \( L_1L_2 \)
• Complementation : \( \overline{L} \)
• Star-closure : \( L_1^* \)

Other closures

• **Difference**: If \( L_1 \) and \( L_2 \) are regular, then \( L_1 - L_2 \) is also regular
  - **Proof**:
    Set difference is defined as 
    \( L_1 - L_2 = L_1 \cap \overline{L_2} \)
    We know that if \( L_2 \) is regular, so is \( \overline{L_2} \). We also know regular languages are closed under intersection. Therefore, we know that \( L_1 \cap \overline{L_2} \) is regular.

  Difference sometimes notated as \( L_1 \setminus L_2 \)

• **Reversal**: If \( L_1 \) is regular, then \( L_1^R \) is also regular.
  - **Proof**:
    Suppose \( L \) is a regular language. We can therefore construct an NFA with a single final state that accepts \( L \). We can then make the start state of this NFA the final state, make the final state the start state, and reverse the direction of all arcs in the NFA. The modified NFA accepts a string \( w^R \) if and only if the original NFA accepts \( w \). Therefore the modified NFA accepts \( L^R \).
Using regular language closure properties

Showing a language is regular

- show that by using two or more known regular languages and one or more of the operations over which regular languages are closed, you can produce that language.

Basic template

\[ L_{\text{REG}1} \text{ [OP] } L_{\text{REG}2} = L_{\text{REG}3} \]

where OP is one of the operations over which regular languages are closed

- \( L_{\text{REG}3} \) is the language in question (i.e., the one we need to prove is regular)
- \( L_{\text{REG}1} \) and \( L_{\text{REG}2} \) are known regular languages

If the two languages on the left side of the operator are regular then so too must be the one on the right side.

NB: Cannot assume that if the language on the right is regular, so too must be both languages on the left.

Example

If \( L \) is a regular language, is \( L_1 = \{uv \mid u \in L, |v| = 2\} \) also regular?

- We know \( L \) is regular
- Every string in \( L_1 \) consists of a string from \( L \) concatenated to a string of length 2
- The set of strings of length 2 (call it \( L_2 \)) over any alphabet is finite, and therefore this is a regular language since all finite languages are regular.
- Therefore we have \( L \) [concatenation] \( L_2 = L_1 \)

Since \( L_1 \) is the concatenation of two regular languages, \( L_1 \) must also be regular.

Example

Prove the language \( \{a^nb^m \mid n, m > 3\} \) is regular

- Show that this language can be produced using regular language closure properties on known regular languages \( L_1 = \{a^*b^m\}, L_2 = \{a, aa, aaa\}, L_3 = \{b, bb, bbb\} \) as follows:
  - concatenation: \( L_4 = L_2 \cdot L_3 \)
  - complementation: \( L_5 = L_4 \)
  - intersection: \( L_6 = L_5 \cap L_1 = \{a^nb^m \mid n, m > 3\} \)
Using regular language closure properties

Showing a language is not regular

- Use the same template:
  \[ L_{\text{REG}1} \text{ [OP]} L_{\text{REG}2} = L_{\text{REG}3} \]
- However:
  - the language in question is plugged into the template in the position of \( L_{\text{REG}1} \)
  - want to use a known regular language for \( L_{\text{REG}2} \)

If we can show that \( L_{\text{REG}3} \) is not regular, then it must be the case that \( L_{\text{REG}1} \) is not regular

Example

\[ L = \{ w \mid w \text{ in } \{a,b\}^* \mid w \text{ has equal number of } a \text{'s and } b \text{'s} \} \] is non-regular

- Use the template:
  \[ L \cap a^*b^* = \{ a^n b^n \mid n \geq 0 \} \]
- If both languages on the left side of the “=” are regular, the language on the right side is regular (closure of regular languages over intersection)
- \( \{ a^n b^n \mid n \geq 0 \} \) easily proved non-regular using the pumping lemma
- We know \( a^*b^* \) is regular
- Therefore \( L \) must be non-regular

Example

- Given
  - \( L_1 \) is regular
  - \( L_1 \cap L_2 \) is regular
  - \( L_2 \) is non-regular
- Is \( L_1 \cup L_2 \) regular?
- Use same strategy as previous example:
  - Make the “unknown” language (\( L_1 \cup L_2 \)) one of the languages on the left side in template
  - Make the other left side language a known regular language
  - Show that the language on the right side is not regular

The unknown language is a bit more complicated because it is the union of two other languages, but this doesn’t change anything

- Fill the template:
  - Use one of our known RLs on left side
  - Use known non-RL on right
- So we have
  \[ L_1 \cup L_2 \text{ [OP] (known regular language)} = L_2 \]
- Need to put in an operation and a known regular language that we know yields \( L_2 \)
- To do this, get \( L_2 \) isolated from \( L_1 \cap L_2 \)
Can’t “extract” $L_2$ from $L_1 \cup L_2$ using only $L_1$ or $L_1 \cap L_2$, since taking the difference of $L_1 \cup L_2$ and $L_1$ gives us only what is left of $L_2$ that is not in $L_1$.
- Have to “remove” anything that is in $L_1 \cap L_2$ from $L_1$, then subtract the result (everything in $L_1$ that is not also in $L_2$) from $L_1 \cup L_2$.
- Difference and union are closed for regular languages.
- After doing this we know we still have a regular language to subtract from $L_1 \cup L_2$.

Result:
$$ (L_1 \cup L_2) - (L_1 - (L_1 \cap L_2)) = L_2 $$

### Divide and Conquer

- $L = \{w \in \{a,b\}^* | w$ contains an even number of $a$’s and an odd number of $b$’s and all $a$’s come in runs of three$\}$
- **Regular:** $L = L_1 \cap L_2$, where
  - $L_1 = \{w \in \{a,b\}^* | w$ contains an even number of $a$’s and an odd number of $b$’s$\}$
  - $L_2 = \{w \in \{a,b\}^* |$ all $a$’s come in runs of three$\}$
- To prove it, build an FSA for each
  - Easier than FSA for the original language.

### Template
- $L_{REG1}$ is $(L_1 \cup L_2)$
- [OP] is “-” (difference)
- $L_{REG2}$ is $(L_1 - (L_1 \cap L_2))$
- $L_{REG3}$ is $L_2$
- We know
  - $L_{REG2}$ is regular
    - produced by applying the closure operations on two known regular languages ($L_1$ and $L_1 \cap L_2$)
  - if $L_{REG1}$ is regular, so is $L_{REG3}$
  - But we were given the fact that $L_2$ is non-regular

We can conclude that $L_{REG1} = L_1 \cup L_2$ is non-regular as well.
What the Closure Theorem for Union Does Not Say

• Closure theorem for union says: If \( L_1 \) and \( L_2 \) are regular, then \( L = L_1 \cup L_2 \) is regular.

• What happens if (for example) \( L \) is regular? Does that mean that \( L_1 \) and \( L_2 \) are also?

  Maybe.

What the Closure Theorem for Concatenation Does Not Say

• Closure Theorem for Concatenation says: If \( L_1 \) and \( L_2 \) are regular, then \( L = L_1 \cdot L_2 \) is regular.

• What happens (for example) if \( L_2 \) is not regular? Does that mean that \( L \) isn’t regular?

  Maybe.

Example

• We know \( a^+ \) is regular

• Consider two cases for \( L_1 \) and \( L_2 \)
  1. \( a^+ = (a^n \mid n > 0 \text{ and } n \text{ is prime}) \cup (a^n \mid n > 0 \text{ and } n \text{ is not prime}) \)
     \* \( a^+ = L_1 \cup L_2 \)
     \* Neither \( L_1 \) nor \( L_2 \) is regular!
  2. \( a^+ = (a^n \mid n > 0 \text{ and } n \text{ is even}) \cup (a^n \mid n > 0 \text{ and } n \text{ is odd}) \)
     \* \( a^+ = L_1 \cup L_2 \)
     \* Both \( L_1 \) and \( L_2 \) are regular!

• Consider two examples:
  1. \( \{aba^*b^n \mid n \geq 0\} = \{ab\} \cup \{a^n b^n \mid n \geq 0\} \)
     \* \( L = L_1 \cdot L_2 \)
     \* \( L_2 \) is not regular!
  2. \( \{aaa^* \} = \{a^* \} \cup \{a^n \mid n \text{ is prime}\} \)
     \* \( L = L_1 \cdot L_2 \)
     \* \( L_2 \) is not regular, but \( L \) is!
True or False?

• If $L_1 \subseteq L_2$ and $L_1$ is not regular, then $L_2$ is not regular.
  \[\text{False!} \ (a, b)^* \text{ is regular, and it has a non-regular subset } \{a^nb^n \mid n \geq 0\}\]

• If $L_1 \subseteq L_2$ and $L_2$ is not regular, then $L_1$ is not regular.
  \[\text{False!} \ \text{Non-regular languages have finite subsets, and finite languages are regular}\]

Hints

• When you need a known regular language, remember that $\Sigma^*$, $\varepsilon$, $a^*$, $a^*b^*$, etc. are regular

• When you need a known non-regular language, use $a^n b^n$ or any language with a similar dependency