Properties of Regular Languages

COMP 455 – 002, Spring 2019
What Languages Aren’t Regular?

The tool used for proving that a language is not regular is the *Pumping Lemma*. 
The Pumping Lemma

**Theorem 4.1** (the *Pumping Lemma for regular languages*): Let \( L \) be a regular language. Then there exists a constant \( n \) (which depends on \( L \)) such that for every \( w \in L \), where \( |w| \geq n \), there exists strings \( x \), \( y \), and \( z \) such that

i. \( w = xyz \),

ii. \( |xy| \leq n \),

iii. \( |y| \geq 1 \), and

iv. for all \( k \geq 0 \), \( xy^k z \in L \).
Proof of the Pumping Lemma

**Theorem 4.1**: Let $L$ be a regular language. Then there exists a constant $n$ (which depends on $L$) such that for every $w \in L$, where $|w| \geq n$, there exists strings $x$, $y$, and $z$ such that (i) $w = xyz$, (ii) $|xy| \leq n$, (iii) $|y| \geq 1$, and (iv) for all $k \geq 0$, $xy^kz \in L$.

- Since $L$ is regular, it is accepted by some DFA $M$.
- Let $n = $ the number of states in $M$.
- Pick any $w \in L$, where $|w| > n$.
- By the pigeonhole principle, $M$ must repeat a state when processing the first $n$ symbols in $w$. 
Proof of the Pumping Lemma

**Theorem 4.1**: Let $L$ be a regular language. Then there exists a constant $n$ (which depends on $L$) such that for every $w \in L$, where $|w| \geq n$, there exists strings $x$, $y$, and $z$ such that (i) $w = xyz$, (ii) $|xy| \leq n$, (iii) $|y| \geq 1$, and (iv) for all $k \geq 0$, $xy^kz \in L$.

$M$ must repeat a state, $q$, when processing the first $n$ symbols of input $w$.

Define strings $x$, $y$, and $z$ as in this figure:
Proof of the Pumping Lemma

Theorem 4.1: Let $L$ be a regular language. Then there exists a constant $n$ (which depends on $L$) such that for every $w \in L$, where $|w| \geq n$, there exists strings $x$, $y$, and $z$ such that (i) $w = xyz$, (ii) $|xy| \leq n$, (iii) $|y| \geq 1$, and (iv) for all $k \geq 0$, $xy^kz \in L$.

The second occurrence of state $q$ must occur within the first $n$ symbols of $w$, so $|xy| \leq n$. Also, $|y| \geq 1$.

It must be possible to repeat the “$y$-loop” 0 or more times, and the resulting string will still be accepted. Therefore, $xy^kz \in L$, for any $k \geq 0$. 

Jim Anderson (modified by Nathan Otterness)
Using the Pumping Lemma

To show that a language $L$ is not regular, show that the conditions of the Pumping Lemma do not hold.

Formally, the Pumping Lemma says:

$\exists n$

$\therefore \left( \forall w: w \in L \land |w| \geq n \right)$

$\therefore \left( \exists x, y, z: xyz = w \land |xy| \leq n \land |y| \geq 1 \right)$

$\therefore \left( \forall k: k \geq 0 \therefore x y^k z \in L \right)$
Using the Pumping Lemma

To show that a language $L$ is not regular, show that the conditions of the Pumping Lemma do not hold.

The negation of the Pumping Lemma is:

\[ \forall n \exists w : w \in L \land |w| \geq n \]

\[ \exists x, y, z : xyz = w \land |xy| \leq n \land |y| \geq 1 \]

\[ \exists k : k \geq 0 \exists x y^k z \notin L \]
Using the Pumping Lemma

\[ \forall n \quad \exists w : w \in L \land |w| \geq n \]
\[ \quad \forall x, y, z : xyz = w \land |xy| \leq n \land |y| \geq 1 \]
\[ \quad \exists k : k \geq 0 \quad \exists y^k z \notin L \]

Show that this statement is true for \( L \) to show that \( L \) is not a regular language!
Using the Pumping Lemma: Example

- Consider the language $0^{i^2}$, where $i \geq 1$.
- We want to prove that this language is not regular.
- We need to show that $\forall n :: \left( \exists w : w \in L \land |w| \geq n :: \left( \forall x, y, z : xyz = w \land |xy| \leq n \land |y| \geq 1 :: \exists k : k \geq 0 :: xy^kz \notin L \right) \right)$ is true for $L$.
- The proof is like a game with an adversary:
  - The adversary makes the “$\forall$” choices.
  - We make the “$\exists$” choices.
Using the Pumping Lemma: Example

- \( L = \{0^{i^2} \mid i \geq 1\} \)
- Assume \( L \) is regular (this will be a proof by contradiction).
- Select any \( n \).
- Let \( w = 0^{n^2} \).
- Select any \( xyz \) where \( w = xyz \), \(|xy| \leq n\), and \(|y| \geq 1\).
- This implies that \( 1 \leq |y| \leq n \).
- Let \( k = 2 \). This means that \( xy^kz \) has \( n^2 + |y| \) 0s.

\[
\forall n \left( \exists w: w \in L \land |w| \geq n \land \exists x, y, z: xyz = w \land |xy| \leq n \land |y| \geq 1 \land \forall k: k \geq 0 \implies xy^kz \notin L \right)
\]

Jim Anderson (modified by Nathan Otterness)
Using the Pumping Lemma: Example

- (Reminder) \( L = \{0^{i^2} \mid i \geq 1\} \)
- Let \( k = 2 \). This means that \( xy^kz \) has \( n^2 + |y| \) 0s.
- \( w \) has \( n^2 \) 0s, and any entry in \( L \) longer than \( w \) must have at least \((n + 1)^2\) 0s.

\[
\begin{align*}
\text{The length of } w &< n^2 + 1 \leq n^2 + |y| \leq n^2 + n < n^2 + 2n + 1. \\
\text{The length of } xy^2z & \text{ if } |y| = 1. \\
\text{The length of } xy^2z & \text{ if } |y| = n. \\
\text{The length of the next string in } L & = (n + 1)^2.
\end{align*}
\]

- So, \( xy^kz \not\in L \), contradicting the Pumping Lemma.
Using the Pumping Lemma: 2nd Example

Consider the language \( L = \{ x \mid x \text{ contains an equal number of 0s and 1s} \} \)

Is \( L \) regular?
Using the Pumping Lemma: 2\textsuperscript{nd} Example

- $L = \{ x \mid x \text{ contains an equal number of 0s and 1s}\}$
- Define $w = 0^n1^n$ for an arbitrary $n$. ($w \in L$)
- No matter how $w$ is divided into $x$, $y$, and $z$, $y$ must consist solely of 0s because $|xy| \leq n$ and $y \neq \varepsilon$.
- Therefore, $xy^0z$ has fewer 0s than 1s and is not in $L$.
  - In this case, $k = 0$.
  - This proof would also work with any $k \geq 2$, because $xy^kz$ would have more 0s than 1s.
A closure property of regular languages is a property that, when applied to a regular language, results in another regular language.

- Union and intersection are examples of closure properties.

We will demonstrate several useful closure properties of regular languages.

 Closure properties can also be useful for proving that languages aren’t regular.
Closure under Union

**Theorem 4.4:** If $M$ and $N$ are regular languages, then $M \cup N$ is a regular language.

**Proof:** Say that $R$ and $S$ are regular expressions where $L(R) = M$ and $L(S) = N$. Construct a regular expression $(R + S)$. This matches $M \cup N$. Since a regular expression exists for $M \cup N$, $M \cup N$ is a regular language.

**Note:** The proofs for concatenation and Kleene closure are similar.
Closure under Complementation

- If $L \subseteq \Sigma^*$, then the **complement** of $L$, denoted $\overline{L}$, is $\Sigma^* - L$.

- **Theorem 4.5:** If $L$ is a regular language over $\Sigma$, then $\overline{L}$ is also a regular language.

- **Proof sketch for Theorem 4.5:**
  1. Construct a DFA for $L$
  2. This can be transformed into a DFA for $\overline{L}$ by making all accepting states non-accepting and vice versa.
  3. This can be proven correct by induction.
Closure under Intersection

**Theorem 4.8**: If $M$ and $N$ are regular languages, then $M \cap N$ is a regular language.

**Proof**: $M \cap N = \overline{M} \cup \overline{N}$.

The book contains a more direct proof. The basic idea is to construct a DFA with states labeled $[p, q]$ where $p$ tracks the state of a DFA for $M$ and $q$ tracks the state of a DFA for $N$. 
Closure under Difference

$\mathit{M} - \mathit{N} \equiv \{x \mid x \in M \land x \notin N\}$

\textbf{Theorem 4.10}: If $M$ and $N$ are regular, then so is $M - N$.

\textbf{Proof}: $M - N = M \cap \bar{N}$. 
Closure under Reversal

- The **reversal** of $L$, written $L^R$ is $\{x \mid x^R \in L\}$ ($x^R$ is the string $x$ written backwards).

- **Theorem 4.11**: If $L$ is regular, then so is $L^R$. 
Proof Sketch for Theorem 4.11

- Start with a DFA for $L$.
- Construct an $\varepsilon$-NFA for $L^R$ as follows:
  1. Reverse all of the transitions in the DFA
  2. Make the DFA’s start state the only accepting state.
  3. Create a new start state with $\varepsilon$-transitions to all of the original accepting states.
- This can be proven correct by induction.
- (The book proves Theorem 4.11 by reasoning about regular expressions.)
Example: Reversal of a DFA

- A DFA for $L$:

- An NFA with $\varepsilon$-transitions for $L^R$:
Homomorphisms

- A homomorphism maps symbols in an alphabet $\Sigma$ to strings over a different alphabet, $\Delta$.

- Example:
  - $\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$
  - $h(0) = ab$, $h(1) = \varepsilon$

- Homomorphisms can be extended to strings:
  - $h(\varepsilon) = \varepsilon$
  - $h(xa) = h(x)h(a)$, for string $x$ and symbol $a$

- Homomorphisms can be extended to languages:
  - If $L$ is a language, $h(L) = \bigcup_{x \in L} h(x)$
Homomorphism Example

- $\Sigma = \{0, 1\}$, $\Delta = \{a, b\}$
- $h(0) = ab$, $h(1) = \varepsilon$
- Homomorphism of a string:
  - $h(0011) = abab$
- Homomorphism of a language:
  - $h(10^*1) = (ab)^*$
Closure under Homomorphism

**Theorem 4.14**: If $L$ is a regular language over $\Sigma$, and $h: \Sigma \rightarrow \Delta^*$ is a homomorphism, then $h(L)$ is also a regular language.

**Proof:**

- Let $L = L(R)$ be the language defined by some regular expression $R$.
- Replace each symbol $a$ in $R$ by $h(a)$. Call the resulting regular expression $h(R)$.
- We will prove $L(h(R)) = h(L)$. 
Proof that $L(h(R)) = h(L)$

- Let $E$ be a subexpression of $R$.
- **Claim:** $L(h(E)) = h(L(E))$.
- We will prove this by induction on the number of operators in $E$.

**Base case:** $E$ is $\varepsilon$, $\emptyset$, or $a$, where $a \in \Sigma$.

- The only interesting case is where $E$ is a single-character regular expression, so $h(L(E)) = \{h(a)\}$.
- $h(E)$ is a regular expression for the same string $h(a)$, so $h(L(E)) = L(h(E))$. 

Jim Anderson (modified by Nathan Otterness)
Proof that $L(h(R)) = h(L)$, continued

- Inductive step: $E$ has at least one operator, and therefore has the form $F + G$, $FG$, or $F^*$.

- Proof for $+$:

  $L(h(E))$
  
  $= L(h(F) + h(G))$, by the definition of $h$ for reg. exps
  
  $= L(h(F)) \cup L(h(G))$, by the definition of the $+$ operator.

  $h(L(E))$
  
  $= h(L(F) \cup L(G))$, since $L(E) = L(F) \cup L(G)$.

  $= h(L(F)) \cup h(L(G))$, since $h$ is applied to individual strings.
Proof that $L(h(R)) = h(L)$, continued

- We have shown the following properties are true for the $+$ operator:
  - $h(L(E)) = h(L(F)) \cup h(L(G))$
  - $L(h(E)) = L(h(F)) \cup L(h(G))$

- To conclude the proof for the $+$ operator, we note that $h(L(F)) = L(h(F))$ and $h(L(G)) = L(h(G))$ by the inductive hypothesis. So, $h(L(E)) = L(h(E))$.

- The proofs for concatenation and Kleene closure are similar.
If $h$ is a homomorphism from alphabet $\Sigma$ to alphabet $\Delta$, and $L$ is a language in $\Delta$, then $h^{-1}(L)$ is the set of strings $w$ in $\Sigma^*$ such that $h(w)$ is in $L$. 

A language $L$ over $\Delta^*$ may contain strings that $h$ can’t map to.
Inverse Homomorphism: Example

It can be difficult to determine $h^{-1}(L)$!

Example:

- $\Sigma = \{0,1\}$, $\Delta = \{a, b\}$.
- Let $h(0) = aa$, $h(1) = aba$.
- $L = (ab + ba)^* a$.

What is $h^{-1}(L)$?

What strings over $\{0,1\}$ can $h$ map to a string in $L$?
Inverse Homomorphism: Example

What is $h^{-1}(L)$?

- No string in $L$ begins with aa, so no string starting with 0 (over \{0, 1\}) can be mapped to a string in $L$.

- Strings starting with 1 will always be mapped to strings that start with a, because $h(1) = \text{aba}$.

- Strings in $L$ that can start with a:
  - aba
  - abab… 1 maps to this
  - abba… No string starting with 1 maps to these
  - a

- So, $h^{-1}(L) = \{1\}$.
Theorem 4.16: If $L$ is a regular language, then $h^{-1}(L)$ is also regular.

Proof:
- Let $M = (Q, \Delta, \delta, q_0, F)$ be a DFA accepting $L$.
- Let $h: \Sigma \rightarrow \Delta^*$.
- Define $M' = (Q, \Sigma, \delta', q_0, F)$ be a DFA accepting $h^{-1}(L)$. 

Jim Anderson (modified by Nathan Otterness)
Closure Under Inverse Homomorphism

\[ M' = (Q, \Sigma, \delta', q_0, F) \]

- On an input \( a \in \Sigma \), \( M' \) simulates the behavior of \( M \) on \( h(a) \).
- Formally, \( \delta'(q, [\hat{a}]) = \delta(q, [h(a)]) \).

Exercise: Prove by induction on \( |x| \) that \( M' \) accepts \( x \) if and only if \( M \) accepts \( h(x) \).
Closure Under Inverse Homomorphism

Example. $L = (ab + ba)^*a$, $h(0) = aa$, $h(1) = aba$.

This can be a good strategy for figuring out $h^{-1}(L)$. 
A **property** is a yes/no question about one or more languages. Some examples:

- Is $L$ empty?
- Is $L$ finite?
- Are $L_1$ and $L_2$ equivalent?

A property is a **decision property for regular languages** if an *algorithm* exists that can answer the question (for regular languages).
The book focuses largely on efficiency issues when discussing decision properties.

Efficiency is more of a COMP 550 topic (not a prerequisite), so these slides approach decision properties slightly differently.

We will consider the three properties from the previous slide: emptiness, finiteness, and equivalence.
We want an algorithm that takes two languages, $L_1$ and $L_2$, and determines if they are the same.

The algorithm:
1. Convert $L_1$ and $L_2$ to DFAs.
2. Convert $L_1$ and $L_2$ to minimal DFAs. (See next slide)
3. Determine if the minimal DFAs are the same.
Minimal DFAs

- Section 4.4 of the textbook gives an algorithm that takes a DFA and outputs a DFA that accepts the same language, but has a minimal number of states.

- This minimal DFA is unique.
  - This means that if two different DFAs define the same language, both will be converted to the same minimal DFA.

- We will be skipping this algorithm—it takes a long time to explain and won’t be used later in the class. You just need to know that it exists.
Emptiness and Finiteness

Theorem:

The set of strings accepted by a DFA $M$ with $n$ states is:

- nonempty if and only if $M$ accepts a string of length less than $n$;
- infinite if and only if $M$ accepts a string of length $k$, where $n \leq k < 2n$.

(This approach is different from the book.)
Proof of “Nonempty” Claim

“...nonempty if and only if $M$ accepts a string of length less than $n$”

▸ “If” : Obvious

▸ “Only if” : Let $w$ be the length of the shortest string accepted by $M$.

❖ If $|w| < n$ we’re done.
❖ If $|w| \geq n$, then by the Pumping Lemma $w = xyz$, and $xz \in L(M)$. Contradiction.

Jim Anderson (modified by Nathan Otterness)
Proof of “Infinite” Claim

 “…infinite if and only if $M$ accepts a string of length $k$, where $n \leq k < 2n$”

▶ “If”: If $w \in L(M)$ and $n \leq |n| < 2n$, then $L(M)$ is infinite by the Pumping Lemma.

▶ “Only if”: Assume $L(M)$ is infinite.

❖ Then, there exists some $w$ where $|w| \geq n$. If such a string has a length under $2n$, we’re done.

❖ Otherwise…
Proof of “Infinite” Claim, continued

“…infinite if and only if $M$ accepts a string of length $k$, where $n \leq k < 2n$”

…otherwise, all strings with a length greater than $n$ are longer than $2n$. Assume this is the case.

Let $w$ be the shortest such string. By the Pumping Lemma, $w = xyz$, $|y| \geq 1$, and $xz \in L(M)$.

- If $|xz| \geq 2n$, we’ve contradicted the assumption that $w$ is the shortest string longer than $2n$.
- If $|xz| < 2n$, then $n \leq xz < 2n$. This contradicts the assumption that the shortest string longer than $n$ is longer than $2n$. 
Decision Algorithms

Algorithm for “nonemptiness”:
❖ See if any string with length at most $|n|$ is in $L(M)$.
❖ Can be done using breadth- or depth-first search to find paths from the start state to a final state.

Algorithm for “infiniteness”:
❖ See if any string with length $|k|$, where $n \leq k < 2n$ is in $L(M)$.
❖ More efficient to check for “reachable cycles”.
❖ Can use depth-first search, but it’s less efficient.
Another Equivalence Algorithm

We can now build another algorithm for testing equivalence:

- Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$
  - $M_1$ and $M_2$ are DFAs
- Create $M_3$, where $L(M_3) = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$
  - Note that $L(M_3)$ is nonempty if and only if $L_1 \neq L_2$.
- Test whether $L_3$ is empty using the “nonemptiness” algorithm.