Regular Expressions

and

The Limits of Regular Languages
Announcements

• Midterm *tonight* in Cubberly Auditorium, 7PM – 10PM.
  • Open-book, open-note, open-computer, closed-network.
  • Covers material up to and including last Monday's lecture.
Regular Expressions
Atomic Regular Expressions

- The regular expressions begin with three simple building blocks.
- The symbol $\emptyset$ is a regular expression that represents the empty language $\emptyset$.
- The symbol $\varepsilon$ is a regular expression that represents the language $\{ \varepsilon \}$
  - This is not the same as $\emptyset$!
- For any $a \in \Sigma$, the symbol $a$ is a regular expression for the language $\{ a \}$
Compound Regular Expressions

- We can combine together existing regular expressions in four ways.
- If $R_1$ and $R_2$ are regular expressions, $R_1R_2$ is a regular expression representing the **concatenation** of the languages of $R_1$ and $R_2$.
- If $R_1$ and $R_2$ are regular expressions, $R_1 | R_2$ is a regular expression representing the **union** of $R_1$ and $R_2$.
- If $R$ is a regular expression, $R^*$ is a regular expression for the **Kleene closure** of $R$.
- If $R$ is a regular expression, $(R)$ is a regular expression with the same meaning as $R$. 

Operator Precedence

- Regular expression operator precedence is

\[(R)\]

\[R^*\]

\[R_1R_2\]

\[R_1 | R_2\]

- So \(ab^*c|d\) is parsed as \(((a(b^*))c)|d\)
Regular Expressions, Formally

- The **language of a regular expression** is the language described by that regular expression.
- Formally:
  - $\mathcal{L}(\epsilon) = \{\epsilon\}$
  - $\mathcal{L}(\emptyset) = \emptyset$
  - $\mathcal{L}(a) = \{a\}$
  - $\mathcal{L}(R_1 R_2) = \mathcal{L}(R_1) \mathcal{L}(R_2)$
  - $\mathcal{L}(R_1 | R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$
  - $\mathcal{L}(R^*) = \mathcal{L}(R)^*$
  - $\mathcal{L}((R)) = \mathcal{L}(R)$
Regular Expressions are Awesome

• Let $\Sigma = \{0, 1\}$

• Let $L = \{ w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring} \}$

\[(0 \mid 1)^*00(0 \mid 1)^*\]

\[
11011100101 \\
0000 \\
1111101111001111
\]
Regular Expressions are Awesome

• Let $\Sigma = \{0, 1\}$
• Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$(0|1)(0|1)(0|1)(0|1)$

0000
1010
1111
1000
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid |w| = 4 \}$

$$(0|1)^4$$

0000
1010
1111
1000
Regular Expressions are Awesome

• Let $\Sigma = \{0, 1\}$
• Let $L = \{ w \in \Sigma^* | w \text{ contains at most one } 0 \}$
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w \text{ contains at most one } 0 \}$

$1^* (0 \mid \varepsilon) 1^*$

- 11110111
- 111111
- 0111
- 0
Regular Expressions are Awesome

- Let $\Sigma = \{0, 1\}$
- Let $L = \{ w \in \Sigma^* \mid w$ contains at most one $0 \}$
Regular Expressions are Awesome

• Let $\Sigma = \{ \text{a, ., @} \}$, where a represents “some letter.”

• Regular expression for email addresses:

  \[
  aa^* (.aa*)^* @ aa^*.aa^* (.aa*)^*
  \]

  cs103@cs.stanford.edu
  first.middle.last@mail.site.org
  barack.obama@whitehouse.gov
Regular Expressions are Awesome

- Let $\Sigma = \{ a, ., @ \}$, where $a$ represents “some letter.”
- Regular expression for email addresses:

  $$a^+(a^+)^*@a^+(a^+)^+$$

  - $cs103@cs.stanford.edu$
  - $first.middle.last@mail.site.org$
  - $barack.obama@whitehouse.gov$
Regular Expressions are Awesome

\[ a^+ (a^+)^* @ a^+ (a^+)^+ \]
The Power of Regular Expressions

**Theorem:** If $R$ is a regular expression, then $\mathcal{L}(R)$ is regular.

**Proof idea:** Induction over the structure of regular expressions. Atomic regular expressions are the base cases, and the inductive step handles each way of combining regular expressions.

Sketch of proof at the appendix of these slides.
The Power of Regular Expressions

Theorem: If $L$ is a regular language, then there is a regular expression for $L$.

*This is not obvious!*

Proof idea: Show how to convert an arbitrary NFA into a regular expression.
From NFAs to Regular Expressions

Regular expression: \((s_1 | s_2 | \ldots | s_n)^*\)

Key idea: Label transitions with arbitrary regular expressions.
From NFAs to Regular Expressions

Key idea: If we can convert any NFA into something that looks like this, we can easily read off the regular expression.
From NFAs to Regular Expressions

Regular expression: $R$

Regular expression: $R_1$
From NFAs to Regular Expressions

Regular expression: $R$

$R_{11}^* R_{12} (R_{22} \mid R_{21} R_{11}^* R_{12})^*$
From NFAs to Regular Expressions
From NFAs to Regular Expressions
From NFAs to Regular Expressions

Note: We’re using concatenation and Kleene closure in order to skip this state.
From NFAs to Regular Expressions

\[
\begin{align*}
\delta(q_s, \epsilon) &= q_1 \\
\delta(q_1, R_{11}) &= q_2 \\
\delta(q_2, R_{12}) &= q_f \\
\delta(q_f, \epsilon) &= q_f
\end{align*}
\]

Regular expressions:

\[
\epsilon R_{11}^* R_{12}
\]

\[
R_{21} R_{11}^* R_{12}
\]
From NFAs to Regular Expressions

Note: We're using union to combine these transitions together.
From NFAs to Regular Expressions

\[
R_{11} \ast R_{12} (R_{22} \mid R_{21} R_{11} \ast R_{12}) \ast \varepsilon
\]
From NFAs to Regular Expressions

\[ R_{11}^* R_{12} \ (R_{22} \mid R_{21} R_{11}^* R_{12})^* \]
The Construction at a Glance

- Start with an NFA for the language $L$.
- Add a new start state $q_s$ and accept state $q_f$ to the NFA.
  - Add $\varepsilon$-transitions from each original accepting state to $q_f$, then mark them as not accepting.
- Repeatedly remove states other than $q_s$ and $q_f$ from the NFA by “shortcutting” them until only two states remain: $q_s$ and $q_f$.
- The transition from $q_s$ to $q_f$ is then a regular expression for the NFA.
There's another example!

Check the appendix to this slide deck.
Our Transformations

DFA

direct conversion

subset construction

NFA

state elimination

recursive transform

Regexp
Regular Languages

• A language $L$ is regular iff
  • $L$ is accepted by some DFA.
  • $L$ is accepted by some NFA.
  • $L$ is described by some regular expression.

• What constructions on regular languages can we do with regular expressions?
String Homomorphism

- Let $\Sigma_1$ and $\Sigma_2$ be alphabets.
- Consider any function $h : \Sigma_1 \rightarrow \Sigma_2^*$ that associates symbols in $\Sigma_1$ with strings in $\Sigma_2^*$.
- For example:
  - $\Sigma_1 = \{0, 1\}$
  - $\Sigma_2 = \{0, 1\}$
  - $h(0) = \varepsilon$
  - $h(1) = 1$
String Homomorphism

- Given a function $h : \Sigma_1 \rightarrow \Sigma_2^*$, the function $h^* : \Sigma_1^* \rightarrow \Sigma_2^*$ is formed by applying $h$ to each character of a string $w$.
- This function is called a string homomorphism.
  - From Greek “same shape.”
String Homomorphism, Intuitively

- Example: Let $\Sigma_1 = \{0, 1, 2\}$ and consider the string 0121.
- If $\Sigma_2 = \{A, B, C, ..., Z, a, b, ..., z, ', [, ], . \}$, define $h : \Sigma_1 \to \Sigma_2^*$ as
  - $h(0) = \text{That's the way}$
  - $h(1) = \text{[Uh huh uh huh]}$
  - $h(2) = \text{I like it}$
- Then $h^*(0121) = \text{That's the way } [\text{Uh huh uh huh}]\text{ I like it } [\text{Uh huh uh huh}]$
- Note that $h^*(0121)$ has the same structure as 0121, just expressed differently.
Homomorphisms of Languages

• If $L \subseteq \Sigma_1^*$ is a language and $h^* : \Sigma_1^* \rightarrow \Sigma_2^*$ is a homomorphism, the language $h^*(L)$ is defined as

$$h^*(L) = \{ h^*(w) | w \in L \}$$

• The language formed by applying the homomorphism to every string in $L$. 
Homomorphisms of Regular Languages

- **Theorem:** If $L$ is a regular language over $\Sigma_1$ and $h^* : \Sigma_1^* \rightarrow \Sigma_2^*$ is a homomorphism, then $h^*(L)$ is a regular language.

- **Proof sketch:** Transform a regular expression for $L$ into a regular expression for $h^*(L)$ by replacing all characters in the regular expression with the value of $h$ applied to that character.

- Examples at the end of these slides.
The Big List of Closure Properties

• The regular languages are closed under
  • Union
  • Intersection
  • Complement
  • Concatenation
  • Kleene Closure
  • String Homomorphism
  • **Plus a whole lot more!**
The Limits of Regular Languages
Is every language regular?
An Important Observation
An Important Observation
Visiting Multiple States

- Let $D$ be a DFA with $n$ states.
- Any string $w$ accepted by $D$ that has length at least $n$ must visit some state twice.
  - Number of states visited is equal to the length of the string plus one.
  - By the pigeonhole principle, some state is duplicated.
- The substring of $w$ between those revisited states can be removed, duplicated, tripled, etc. without changing the fact that $D$ accepts $w$. 
Intuitively

\begin{center}
\begin{tikzpicture}
    \node (start) [draw, circle, inner sep=0.3cm] {start};
    \node (x) [draw, circle, inner sep=0.3cm] at (2,0) {$x$};
    \node (y) [draw, circle, inner sep=0.3cm] at (2,-1) {$y$};
    \node (z) [draw, circle, inner sep=0.3cm] at (2,-2) {$z$};
    \node (end) [draw, circle, inner sep=0.3cm] at (2,-3) {};\draw[->,dashed] (y) to (x);
    \draw[->,dashed] (x) to (z);
    \draw[->,dashed] (z) to (end);
    \draw[->,dashed] (end) to (x);
\end{tikzpicture}
\end{center}
Informally

- Let $L$ be a regular language.
- If we have a string $w \in L$ that is “sufficiently long,” then we can split the string into three pieces and “pump” the middle.
- We can write $w = xyz$ such that $xy^0z$, $xy^1z$, $xy^2z$, ..., $xy^n z$, ... are all in $L$.
  - **Notation**: $y^n$ means “$n$ copies of $y$.”
The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

  For any regular language $L$,

  There exists a positive natural number $n$ such that

  For any $w \in L$ with $|w| \geq n$,

  There exists strings $x, y, z$ such that

  For any natural number $i$,

  $$w = xyz,$$

  $$y \neq \varepsilon$$

  $$xyz^i \in L$$

  This number $n$ is sometimes called the pumping length.
The Weak Pumping Lemma

• The **Weak Pumping Lemma for Regular Languages** states that

  **For any** regular language $L$,

  **There exists** a positive natural number $n$ such that

  **For any** $w \in L$ with $|w| \geq n$,

  **There exists** strings $x, y, z$ such that

  **For any** natural number $i$,

  $w = xyz$,

  $y \neq \varepsilon$

  $xy^iz \in L$

Strings longer than the pumping length must have a special property.
The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that

  For any regular language $L$,

  There exists a positive natural number $n$ such that
  
  For any $w \in L$ with $|w| \geq n$,

  There exists strings $x, y, z$ such that
  
  For any natural number $i$,

  $w = xyz$, $w$ can be broken into three pieces,

  $y \neq \varepsilon$ where the middle piece isn't empty,

  $xy^iz \in L$ where the middle piece can be replicated zero or more times.
The Weak Pumping Lemma

- Let $\Sigma = \{0, 1\}$ and $L = \{ w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring.} \}$
- Any string of length 3 or greater can be split into three pieces, the second of which can be “pumped.”
The Weak Pumping Lemma

- Let $\Sigma = \{0, 1\}$ and $L = \{ \varepsilon, 0, 1, 00, 01, 10, 11 \}$

- Any string of length 3 or greater can be split into three pieces, the second of which can be “pumped.”

The weak pumping lemma holds for finite languages because the pumping length can be longer than the longest string!
The equality problem is defined as follows: **Given two strings x and y, decide if x = y.**

Let Σ = {0, 1, ?}. We can encode the equality problem as a string of the form x?y.

- “Is 001 equal to 110 ?” would be 001?110
- “Is 11 equal to 11 ?” would be 11?11
- “Is 110 equal to 110 ?” would be 110?110

Let $EQUAL = \{ w?w \mid w \in \{0, 1\}^* \}$

**Question**: Is $EQUAL$ a regular language?
What's Going On?

• The weak pumping lemma says that for “sufficiently long” strings, we should be able to pump some part of the string.

• We can't pump any part containing the ?, because we can't duplicate or remove it.

• We can't pump just one part of the string, because then the strings on opposite sides of the ? wouldn't match.

• Can we formally show that $EQUAL$ is not regular?
Theorem: \textit{EQUAL} is not regular.

Proof: By contradiction; assume that \textit{EQUAL} is regular. Let \( n \) be the pumping length guaranteed by the weak pumping lemma. Let \( w = 0^n?0^n \). Then \( w \in \textit{EQUAL} \) and \(|w| = 2n + 1 \geq n\). Thus by the weak pumping lemma, we can write \( w = xyz \) such that \( y \neq \varepsilon \) and for any \( i \in \mathbb{N}, xy^iz \in \textit{EQUAL} \). Then \( y \) cannot contain \( ? \), since otherwise if we let \( i = 0 \), then \( xy^iz = xz \) does not contain \( ? \) and would not be in \textit{EQUAL}. So \( y \) is either completely to the left of the \( ? \) or completely to the right of the \( ? \). Let \(|y| = k\), so \( k > 0 \). Since \( y \) is completely to the left or right of the \( ? \), then \( y = 0^k \). Now, we consider two cases:

\textit{Case 1:} \( y \) is to the left of the \( ? \). Then \( xy^2z = 0^{n+k}?0^n \notin \textit{EQUAL} \), contradicting the weak pumping lemma.

\textit{Case 2:} \( y \) is to the right of the \( ? \). Then \( xy^2z = 0^n?0^{n+k} \notin \textit{EQUAL} \), contradicting the weak pumping lemma.

In either case we reach a contradiction, so our assumption was wrong. Thus \textit{EQUAL} is not regular. ■
Nonregular Languages

- The weak pumping lemma describes a property common to all regular languages.
- Any language \( L \) which does not have this property cannot be regular.
- What other languages can we find that are not regular?
A Canonical Nonregular Language

- Consider the language \( L = \{ 0^n1^n \mid n \in \mathbb{N} \} \).
  \[ L = \{ \varepsilon, 01, 0011, 000111, 00001111, \ldots \} \]

- \( L \) is a classic example of a nonregular language.

- Intuitively: If you have only finitely many states in a DFA, you can't "remember" an arbitrary number of 0s.

- How would we prove that \( L \) is nonregular?
The Pumping Lemma as a Game

- The weak pumping lemma can be thought of as a game between you and an adversary.
- **You win** if you can prove that the pumping lemma fails.
- **The adversary wins** if the adversary can make a choice for which the pumping lemma succeeds.
- The game goes as follows:
  - **The adversary** chooses a pumping length $n$.
  - **You** choose a string $w$ with $|w| \geq n$ and $w \in L$.
  - **The adversary** breaks it into $x$, $y$, and $z$.
  - **You** choose an $i$ such that $xy^iz \notin L$ (if you can't, you lose!)
The Pumping Lemma Game

**ADVERSARY**

Maliciously choose pumping length $n$.

Maliciously split $w = xyz$, $y \neq \varepsilon$

Grrr! Aaaargh!

**YOU**

Cleverly choose a string $w \in L$, $|w| \geq n$

Cleverly choose $i$ such that $xy^i z \notin L$
Theorem: \( L = \{0^n1^n \mid n \in \mathbb{N}\} \) is not regular.

Proof: By contradiction; assume \( L \) is regular. Let \( n \) be the pumping length guaranteed by the weak pumping lemma. Consider the string \( w = 0^n1^n \). Then \(|w| = 2n \geq n\) and \( w \in L \), so we can write \( w = xyz \) such that \( y \neq \varepsilon \) and for any \( i \in \mathbb{N} \), we have \( xy^iz \in L \).

We consider three cases:

Case 1: \( y \) consists solely of 0s. Then \( xy^0z = xz = 0^{n-|y|}1^n \), and since \( |y| > 0 \), \( xz \notin L \).

Case 2: \( y \) consists solely of 1s. Then \( xy^0z = xz = 0^n1^{n-|y|} \), and since \( |y| > 0 \), \( xz \notin L \).

Case 3: \( y \) consists of \( k > 0 \) 0s followed by \( m > 0 \) 1s. Then \( xy^2z \) has the form \( 0^m1^m0^k1^n \), so \( xy^2z \notin L \).

In all three cases we reach a contradiction, so our assumption was wrong and \( L \) is not regular. ■
Counting Symbols

• Consider the alphabet $\Sigma = \{ 0, 1 \}$ and the language

$$BALANCE = \{ w \in \Sigma^* \mid w \text{ contains an equal number of } 0\text{s and } 1\text{s.} \}$$

• For example:
  • $01 \in BALANCE$
  • $110010 \in BALANCE$
  • $11011 \notin BALANCE$

• **Question:** Is $BALANCE$ a regular language?
An Incorrect Proof

Theorem: BALANCE is regular.

Proof: We show that BALANCE satisfies the condition of the pumping lemma. Let \( n = 2 \) and consider any string \( w \in \text{BALANCE} \) such that \(|w| \geq 2\). Then we can write \( w = xyz \) such that \( x = z = \varepsilon \) and \( y = w \), so \( y \neq \varepsilon \). Then for any natural number \( i \), \( xy^i z = w^i \), which has the same number of 0s and 1s. Since BALANCE passes the conditions of the weak pumping lemma, BALANCE is regular. ■
The Weak Pumping Lemma

- The **Weak Pumping Lemma for Regular Languages** states that
  
  **For any** regular language \( L \),
  
  **There exists** a positive natural number \( n \) such that
  
  **For any** \( w \in L \) with \( |w| \geq n \),
  
  **There exists** strings \( x, y, z \) such that
  
  **For any** natural number \( i \),
  
  \( w = xyz \), \( w \) can be broken into three pieces,
  
  \( y \neq \varepsilon \) where the middle piece isn’t empty,
  
  \( xy^i z \in L \) where the middle piece can be replicated zero or more times.

This says **nothing** about languages that aren’t regular!
Caution with the Pumping Lemma

• The weak and full pumping lemmas describe a necessary condition of regular languages.
  • If $L$ is regular, $L$ passes the conditions of the pumping lemma.

• The weak and full pumping lemmas are not a sufficient condition of regular languages.
  • If $L$ is not regular, it still might pass the conditions of the pumping lemma!

• If a language fails the pumping lemma, it is definitely not regular.

• If a language passes the pumping lemma, we learn nothing about whether it is regular or not.
BALANCE is Not Regular

• The language BALANCE can be proven not to be regular using a stronger version of the pumping lemma.
• To see the full pumping lemma, we need to revisit our original insight.
An Important Observation
Pumping Lemma Intuition

- Let $D$ be a DFA with $n$ states.
- Any string $w$ accepted by $D$ that has length at least $n$ must visit some state twice within its first $n$ characters.
  - Number of states visited is equal $n + 1$.
  - By the pigeonhole principle, some state is duplicated.
- The substring of $w$ in-between those revisited states can be removed, duplicated, tripled, quadrupled, etc. without changing the fact that $w$ is accepted by $D$. 
The Pumping Lemma

For any regular language $L$,

There exists a positive natural number $n$ such that

For any $w \in L$ with $|w| \geq n$,

There exists strings $x, y, z$ such that

For any natural number $i$,

$$w = xyz, \quad w \text{ can be broken into three pieces},$$

$$|xy| \leq n, \quad \text{where the first two pieces occur at the start of the string},$$

$$y \neq \varepsilon \quad \text{where the middle piece isn't empty},$$

$$xy^iz \in L \quad \text{where the middle piece can be replicated zero or more times}.$$
Why This Change Matters

• The restriction $|xy| \leq n$ means that we can limit where the string to pump must be.

• If we specifically craft the first $n$ characters of the string to pump, we can force $y$ to have a specific property.

• We can then show that $y$ cannot be pumped arbitrarily many times.
Theorem: BALANCE is not regular.

Proof: By contradiction; assume that BALANCE is regular. Let \( n \) be the length guaranteed by the pumping lemma. Consider the string \( w = 0^n1^n \). Then \( |w| = 2n \geq n \) and \( w \in BALANCE \). Therefore, there exist strings \( x, y, \) and \( z \) such that \( w = xyz \), \( |xy| \leq n \), \( y \neq \varepsilon \), and for any natural number \( i \), \( xy^iz \in BALANCE \). Since \( |xy| \leq n \), \( y \) must consist solely of 0s. But then \( xy^2z = 0^{n+|y|}1^n \), and since \( |y| > 0 \), \( xy^2z \notin BALANCE \).

We have reached a contradiction, so our assumption was wrong and BALANCE is not regular. ■
Summary of the Pumping Lemma

- Using the pigeonhole principle, we can prove the **weak pumping lemma** and **pumping lemma**.

- These lemmas describe essential properties of the regular languages.

- Any language that fails to have these properties cannot be regular.
Next Time

- **Beyond Regular Languages**
  - Context-free languages.
  - Context-free grammars.
Appendix: From Regular Expressions to NFAs
A Marvelous Construction

- To show that any language described by a regular expression is regular, we show how to convert a regular expression into an NFA.

- **Theorem**: For any regular expression $R$, there is an NFA $N$ such that
  - $\mathcal{L}(R) = \mathcal{L}(N)$
  - $N$ has exactly one accepting state.
  - $N$ has no transitions into its start state.
  - $N$ has no transitions out of its accepting state.
Base Cases

Automaton for $\varepsilon$

Automaton for $\emptyset$

Automaton for single character $a$
Construction for $R_1 R_2$
Construction for $R_1 \mid R_2$
Construction for R*
Appendix: Homomorphisms of Regular Languages
Homomorphisms of Regular Languages

- Consider the language defined by the regular expression \((0120)^*\) and the function
  - \(h(0) = n\)
  - \(h(1) = y\)
  - \(h(2) = a\)
- Then \(h^*( (0120)^* ) = ( (n) (y) (a) (n) )^* \)
Homomorphisms of Regular Languages

- Consider the language $011^*$ and the function
  - $h(0) = \text{Here}$
  - $h(1) = \text{Kitty}$
- Then $h^*(011^*) = (\text{Here})(\text{Kitty})(\text{Kitty})^*$