Introduction to Regular Expressions

• In arithmetic, we use addition and multiplication operations in expressions such as $(5 + 3) \times 4$ to describe a *quantity*
  – The value of this expression is the number 32

• Similarly, we use union, concatenation, and * operations in expressions such as $(0 \cup 1)^*0$ to describe a *language*
  – The value of this expression is the language consisting of all strings starting with a 0 or a 1 and ending with a 0
Introduction to Regular Expressions

• If $\Sigma = \{a, b\}$, then the following are regular expressions (i.e., languages) over $\Sigma$:
  $\emptyset$
  $\epsilon$
  $a$
  $(a \cup b)^*$
  $abba \cup \epsilon$

• Note that $a$ is shorthand for $\{a\}$, $\epsilon$ is shorthand for $\{\epsilon\}$, and $(a \cup b)^*$ is shorthand for $\{a, b\}^*$
Operations on Regular Expressions

• Regular expressions use three operations: union, concatenation, and Kleene star

• The **union** of regular expressions is the same as for the union of sets, since regular expressions represent languages, which are sets

• Example: \{01, 111, 10\} \cup \{00, 01\} = \{01, 111, 10, 00\}
Operations on Regular Expressions

- The **concatenation** of regular expressions $R_1$ and $R_2$ is denoted $R_1R_2$
- The concatenation of regular expressions $R_1$ and $R_2$ contains every string $wx$ such that $w$ is in $R_1$ and $x$ is in $R_2$
- Example: $\{01, 111, 10\} \{00, 01\} = \{0100, 0101, 11100, 11101, 1000, 1001\}$
Operations on Regular Expressions

• The *Kleene star* of a regular expression R is denoted R*
  
  – R* is the set of strings formed by concatenating zero or more strings from R, in any order

• \[ R^* = \epsilon \cup R \cup RR \cup RRR \cup \ldots \]

• Example: \((0 \cup 10)^* = \{\epsilon, 0, 10, 00, 010, 100, 1010, \ldots\}\)
Algebraic Laws for Regular Expressions

• Let $\alpha$, $\beta$, and $\gamma$ represent any regular expressions:
  – The union of regular expressions is associative
    • $(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$
  – The union of regular expressions is commutative
    • $\alpha \cup \beta = \beta \cup \alpha$
  – Concatenation is associative
    • $(\alpha \beta) \gamma = \alpha (\beta \gamma)$
  – Concatenation is not commutative
    • $\alpha \beta \neq \beta \alpha$
  – Concatenation distributes over union
    • $\alpha (\beta \cup \gamma) = \alpha \beta \cup \alpha \gamma$
Precedence of Operators

• Just as with arithmetic operations, parentheses may be used wherever needed to influence the grouping of operators.
• Order of precedence is * (highest), then concatenation, then ∪ (lowest).
Kleene Star and Regular Expressions

- Let $\alpha$ and $\beta$ represent any regular expressions:
  - $\emptyset^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $(\alpha^*)^* = \alpha^*$
  - $\alpha^* \alpha^* = \alpha^*$
  - $(\alpha \cup \beta)^* = (\alpha^* \beta^*)^*$
Identities and Annihilators

- If $R$ is a regular expression, then
  - $R \cup \emptyset = R$
    - $\emptyset$ is the identity for union
  - $\epsilon R = R \epsilon = R$
    - $\epsilon$ is the identity for concatenation
  - $\emptyset R = R \emptyset = \emptyset$
    - $\emptyset$ is the “annihilator” for concatenation
Examples of Regular Expressions

• In all of these examples, assume $\Sigma = \{0, 1\}$
  – $0^*10^*$ = $\{w : w$ contains a single 1$\}$
  – $\Sigma^*1\Sigma^*$ = $(0 \cup 1)^*1(0 \cup 1)^*$ = $\{w : w$ has at least one 1$\}$
  – $\Sigma^*001\Sigma^*$ = $\{w : w$ contains the string 001$\}$
  – $(01^+ \cup 1)^*$ = $\{w :$ every 0 in $w$ is followed by at least one 1$\}$
  – $(\Sigma\Sigma)^*$ = $\{w : w$ is a string of even length$\}$
  – $(\Sigma\Sigma\Sigma)^*$ = $\{w :$ the length of $w$ is a multiple of 3$\}$
  – $01 \cup 10$ = $\{01, 10\}$
Examples of Regular Expressions

• In all of these examples, assume $\Sigma = \{0, 1\}$
  - $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w : w$ starts and ends with the same character$\}$
  - $(0 \cup \epsilon)1^* = 01^* \cup 1^*$
    • Note: the expression $0 \cup \epsilon$ describes the language $\{0, \epsilon\}$, so the concatenation operation adds either $0$ or $\epsilon$ before every string in $1^*$
  - $(\epsilon \cup 1)(\epsilon \cup 0) = \{\epsilon, 0, 1, 10\}$
  - $1^*\emptyset = \emptyset$
    • Note: concatenating the empty set to any set yields the empty set
Examples of Regular Expressions

Details matter:
\[ a^* \cup b^* \neq (a \cup b)^* \]
\((a \cup b)^*\) contains the string \(ab\), whereas \(a^* \cup b^*\) does not. Every string in \(a^* \cup b^*\) contains only \(a\)’s or only \(b\)’s.

\((ab)^* \neq a^*b^*\)
\(a^*b^*\) contains the string \(aaabbb\), whereas \((ab)^*\) does not. \((ab)^*\) contains the string \(abab\), whereas \(a^*b^*\) does not.
Examples of Regular Expressions

• Matching decimal encoding of numbers:
  \(-?[0-9]+(\.[0-9]*)? \mid 0\.[0-9]+\)

• Legal passwords:
  \(((a-z) \cup (A-Z)) ((a-z) \cup (A-Z) \cup (0-9) \cup _) \{8,16\}\)

• IP addresses:
  \(((0-9){1,3} (\.[0-9]{1,3})) \{3\}\)
Why Are Regular Expressions Useful?

• Regular expressions are significant for two reasons:
  1. They are a useful way to define patterns
  2. The languages that can be defined with regular expressions are exactly the *regular languages*
Examples of Regular Languages

• \( L(01) = \{01\} \)
• \( L(01 \cup 0) = \{01, 0\} \)
• \( L(0(1 \cup 0)) = \{01, 00\} \)
  – Note the order of precedence of operators
• \( L(0^*) = \{\varepsilon, 0, 00, 000,\ldots\} \)
• \( L((0 \cup 10)^*(\varepsilon \cup 1)) = \) all strings of 0’s and 1’s that do not have two consecutive 1’s
Analyzing a Regular Language

\[ L((a \cup b)^*b) = L((a \cup b)^*) L(b) \]
\[ = (L(a \cup b))^* L(b) \]
\[ = (L(a) \cup L(b))^* L(b) \]
\[ = (\{a\} \cup \{b\})^* \{b\} \]
\[ = \{a, b\}^* \{b\} \]

So the meaning of the regular expression \((a \cup b)^*b\) is the set of all strings over the alphabet \{a, b\} that end in \(b\).
Analyzing a Regular Language

\[ L((a \cup b) (a \cup b)a(a \cup b)^*) = (L(a \cup b) L(a \cup b) L(a) L((a \cup b)^*)) = (L(a \cup b)) (L(a \cup b)) \{a\} (L((a \cup b)^*)) \]
\[ = L(a \cup b) L(a \cup b) \{a\} \{a, b\}* \]
\[ = \{a, b\} \{a, b\} \{a\} \{a, b\}* \]

So the meaning of the regular expression \((a \cup b) (a \cup b)a(a \cup b)^*\) is the set of all strings over the alphabet \{a, b\} such that there exists an \(a\) in the third location of the string.

Alternatively, we could say \(L = \{xa y : x\text{ and } y \text{ are strings of } a\text{'s and } b\text{'s\ and } |x| = 2\}\).
Analyzing a Regular Language

Question: Find a regular expression that represents the language $L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$:

$$((a \cup b)(a \cup b))^*$$

Another regular expression that fits the language is:

$$(aa \cup ab \cup ba \cup bb)^*$$
Analyzing a Regular Language

Question: Find a regular expression that represents the language $L = \{w \in \{a, b\}^*: w$ has an odd number of $a$’s$\}$

$$b^*(ab^*ab^*)^*ab^*$$

Another regular expression that fits the language is:

$$b^*ab^*(ab^*ab^*)^*$$
Regular Languages and Automata

- Each of the three types of automata (DFA, NFA, $\epsilon$-NFA) that we have discussed, and regular expressions as well, define exactly the same set of languages: *the regular languages*

- Theorem: A language is *regular* if and only if it is recognized by some finite automaton
Kleene’s Theorem

- Kleene’s Theorem: if a language is regular then it is recognized by a finite automaton, and conversely, if a language is recognized by a finite automaton then it is regular.

- This proof of this theorem has two directions:
  - Given a regular language, we can construct a finite automaton that recognizes it.
  - Given a finite automaton, we can find a regular language that recognizes it.
  - We will use a constructive proof to do this.
Proof: RE to NFA Construction

Given a regular expression \( \alpha \), construct a NFA called \( M \) such that \( L(\alpha) = L(M) \). Start with the *primitive* regular expressions, \( \emptyset \), \( c \), and \( \epsilon \):

1. DFA for \( \emptyset \):

2. DFA for a single character \( c \) of \( \Sigma \)

3. DFA for \( \epsilon \)
RE to NFA Construction: Union

$\epsilon$-NFA for $R_1$

$\epsilon$-NFA for $R_2$

$\epsilon$-NFA for $R_1 \cup R_2$
RE to NFA Construction: Concatenation

ε-NFA for $R_1$ → ε-NFA for $R_2$ → ε-NFA for $R_1R_2$
RE to NFA Construction: Star

$\epsilon$-NFA for $R$

$\epsilon$-NFA for $R^*$
RE to NFA Construction: Example 1

Construct an NFA to represent the regular expression \((b \cup ab)^*\)

Step 1:  
- NFA for \(b\):  
- NFA for \(a\):  
- NFA for \(b\):  

- NFA for \(ab\):
RE to NFA Construction: Example 1

Construct an NFA to represent the regular expression \((b \cup ab)^*\)

Step 2: NFA for \((b \cup ab)\):
RE to NFA Construction: Example 1

Construct an NFA to represent the regular expression \((b \cup ab)^*\)

**Step 3:** NFA for \((b \cup ab)^*\):

New start state

![NFA Diagram](image-url)
Construct an NFA to represent the regular expression

$$(a \cup bb)^* (ba^* \cup \epsilon)$$
Construct an NFA to represent the regular expression

\[(a \cup bb)^* (ba^* \cup \epsilon)\]
NFA to RE Construction

• Goal: Given an NFA called M, find an equivalent machine M’ such that:
  – M’ has only two states: a start state and an accepting state, along with a single transition from the start state to the accepting state

• Idea: Instead of limiting the transitions to a single character or an ε, allow the entire regular expression to be the transition label
NFA to RE Construction

• Procedure: Given an arbitrary NFA M, M’ will be built by starting with M and then removing, one at a time, all of the states that lie in-between the start state and an accepting state.

• As each such state is removed, the remaining transitions will be modified so that the set of strings that can drive M’ from its start state to some accepting state remains unchanged.
NFA to RE Construction

For example, let $M$ be:

Suppose we “rip out” state 2:

From this NFA, we see that the equivalent RE is $ab^*a$. 

NFA to RE Construction: Example 1

Let M be:

[Diagram of an NFA with transitions labeled as follows: 1 to 2 with label 'a', 2 to 1 with label 'b', 3 to 1 with label 'b', 1 to 3 with label 'a', and 3 to itself with label 'a'.]
NFA to RE Construction: Example 1

1. Create a new initial state, 4, and a new, unique accepting state, 5, neither of which is part of a loop.

Before

After
NFA to RE Construction: Example 1

2. Repeatedly remove states and edges and replace with edges labeled with larger and larger regular expressions.

Before

4  ε  1  a  2  b
  3  b  a
  5  ε  ε  ε

After

4  ε  1  a  2  b
  5  aa*b

After removal of state 3
NFA to RE Construction: Example 1

To remove state 2, notice that an $a$ transitions into state 2, and a $b$ and $aa^*b$ transition out of state 2 to state 1. Also note that an $\varepsilon$ transitions out of state 2 to state 5.

Before

\[
\begin{array}{ccc}
4 & \varepsilon & 1 \\
& a & 2 \\
& b & \\
& \varepsilon & 5 \\
\end{array}
\]

After

\[
\begin{array}{ccc}
4 & \varepsilon & 1 \\
& ab \cup aaa^*b & \\
\end{array}
\]

After removal of state 2
NFA to RE Construction: Example 1

Before

\[
\begin{array}{c}
4 \\
\varepsilon \\
1 \\
\varepsilon \\
5 \\
\end{array}
\xrightarrow{\text{ab} \cup \text{aaa*}b} 
\begin{array}{c}
4 \\
5 \\
\end{array}
\]

After

\[
\begin{array}{c}
4 \\
\end{array}
\xrightarrow{(\text{ab} \cup \text{aaa*}b)^*(\text{a} \cup \varepsilon)} 
\begin{array}{c}
5 \\
\end{array}
\]

After removal of state 1
NFA to RE Procedure Summary

• First, standardize the NFA:
  – Start by removing from M any states that are unreachable from the start state. If the start state is part of a loop, create a new start state and connect it to M’s start state with an epsilon transition.
  – If there is more than one accepting state of M, or if there is just one but there are any transitions out of it, create a new accepting state and connect each of M’s accepting states to it via an epsilon transition. Remove the old accepting states from the set of accepting states.
  – If there is more than one transition between states p and q, collapse them into a single transition.
  – If there is a pair of states p, q, and there is no transition between them and p is not the accepting state and q is not the start state, then create a transition from p to q labeled ∅.
NFA to RE Procedure Summary

• Next, create the RE from the NFA:
  1. If M has no accepting states, then halt and return the simple RE $\emptyset$
  2. If M has only one state, then halt and return the simple RE $\epsilon$
  3. Repeat until only the start state and the accepting state remain:
     3.1 Select some state $rip$ of M. Any state except the start state or the accepting state may be chosen.
     3.2 For every transition from some state $p$ to some state $q$, if both $p$ and $q$ are not $rip$ then, using the current labels given by the expression R, compute the new label $R'$ for the transition from $p$ to $q$ using the formula
        $R'(p, q) = R(p, q) \cup R(p, rip) R(rip, rip)^* R(rip, q)$
     3.3 Remove $rip$ and all transitions into and out of it
  4. Return the regular expression that labels the one remaining transition from the start state to the accepting state
Showing a Language is Regular

• Question: Is the language \( L = \{w \in \{a,b\}^*: awa\} \) regular?
  – To show that this (or any other language) is regular, all we have to do is find a DFA that recognizes it.
  – This DFA must check whether a string begins and ends with an \( a \).
  – What is between the starting and ending \( a \) does not matter as long as each character is in the alphabet \( \{a, b\} \).
Showing a Language is Regular

• Question: Is the language \( L = \{ w \in \{a,b\}^* : awa \} \) regular?
• Yes, \( L \) is regular because we can construct a DFA that recognizes \( L \)
• Here it is:
Question: Is the language $L = \{w_1, w_2 \in \{a, b\}^* : aw_1aaw_2a\}$ regular?

– The DFA from the previous slide can be used as the starting point, but the state $q_3$ has to be modified (it can no longer be final)

– Replicate the states from the first part (with new names) and have $q_3$ as the beginning of the new part
Showing a Language is Regular

• Since the complete string can be broken into its constituent parts whenever $aa$ occurs, we let the first occurrence of two consecutive $a$’s be the “trigger” that gets the DFA into its second part
  – We can do this by making $\delta(q_3, a) = q_4$

• Since this DFA recognizes $L$, $L$ must be regular
Showing a Language is Regular

- Show that $L = \{w \in \{a, b\}^* : \text{every } a \text{ is immediately followed by a } b\}$ is regular

- Since this DFA recognizes $L$, $L$ must be regular
Showing a Language is Regular

• Show that \( L = \{w \in \{0, 1\}^* : w \text{ has odd parity}\} \) is regular

• Since this DFA recognizes \( L \), \( L \) must be regular

Odd parity = there is an odd number of 1 bits in the string.
Showing a Language is Regular

• Show that $L = \{w \in \{a, b\}^* : w$ has at most one $b\}$ is regular

• Since this DFA recognizes $L$, $L$ must be regular
Showing a Language is Regular

- Show that $L = \{w \in \{a, b\}^* : \text{every } a \text{ region in } w \text{ is of even length}\}$ is regular

- Since this DFA recognizes $L$, $L$ must be regular
Closure Properties

• Recall that a collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection

• The class of regular languages is closed under the union operation
  – If $L_1$ and $L_2$ are regular languages, then $L_1 \cup L_2$ is also a regular language
Closure Under Union: Proof

• Let \( L_1 \) and \( L_2 \) be regular languages that recognize regular expressions \( r_1 \) and \( r_2 \) respectively. Then \( L_1 \cup L_2 \) is also a regular language. Proof:

• Let \( M(r_1) = (Q_1, \Sigma, \delta_1, q_1, F_1) \) and \( M(r_2) = (Q_2, \Sigma, \delta_2, q_2, F_2) \) be finite automata such that \( M(r_1) = L_1 \) and \( M(r_2) = L_2 \)

• Construct a finite automaton \( M \) such that \( M = M(r_1) \cup M(r_2) \)
Closure Under Union: Proof

• On input $w$, $M$ will keep track of the states of both $M(r_1)$ and $M(r_2)$

• Let $M = (Q, \Sigma, \delta, q_0, F)$ where:

  $Q = Q_1 \times Q_2$
  $\delta$ is defined as follows: for all $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
  $q_0 = \{q_1, q_2\}$
  $F = \{r_1, r_2\}$ where $(r_1 \in F_1$ or $r_2 \in F_2)$
Closure Under Concatenation and *

- The regular languages are closed under concatenation and Kleene star.
- The proofs use same ideas as for union:
  - Concatenation: Let $M(r_1)$ be the language that recognizes regular expression $r_1$, and let $M(r_2)$ be the language that recognizes regular expression $r_2$, then $M = M(r_1) \cdot M(r_2)$ is a regular expression whose language is $r_1r_2$.
  - Kleene star: $R^*$ is a regular expression whose language is $L^*$. 
Closure Under Intersection

• The regular languages are closed under intersection
  – If L and M are regular languages, then so is $L \cap M$
• Proof: Let A and B be DFAs whose languages are L and M, respectively
• Construct C, the cross-product automaton of A and B
• Make the final states of C be the pairs consisting of the final states of both A and B
Cross-Product DFA for Intersection

Cross-product of the states \{A, B\} and \{C, D\} = \{AC, AD, BC, BD\}

For the cross-product transitions, state AC on 0 goes to state AD because state A on 0 goes to state A and C on 0 goes to D.
Closure Under Difference

• Regular languages are closed under difference
  – If \( L \) and \( M \) are regular languages, then so is \( L - M \) (the strings in \( L \) but not \( M \))

• Proof: Let \( A \) and \( B \) be DFAs whose languages are \( L \) and \( M \), respectively

• Construct \( C \), the product automaton of \( A \) and \( B \)

• The final states of \( C \) are the pairs whose A-state is final but whose B-state is not
Cross-Product DFA for Difference

In the original two DFAs, state B is final, but state D is not.
Closure Under Complementation

• Regular languages are closed under complement
  – The *complement* of a language L (with respect to an alphabet \( \Sigma \) such that \( \Sigma^* \) contains L) is \( \Sigma^* - L \)

• Proof: since \( \Sigma^* \) is surely regular, and we know that regular languages are closed under difference, we immediately know that the complement of any regular language is also regular.
Closure Under Reversal

• Regular languages are closed under reversal
  – Given language \( L \), \( L^R \) is the set of strings whose reversal is in \( L \)
  – For example, if \( L = \{0, 01, 100\} \) then \( L^R = \{0, 10, 001\} \)
• Proof: Let \( E \) be a regular expression for \( L \)
  – Use proof by induction to show how to reverse \( E \) to provide a regular expression \( E^R \) for \( L^R \)
Reversal of a Regular Expression

• Basis: If E is a symbol $a$, $\epsilon$, or $\emptyset$, then $E^R = E$

• Induction:

If E is composed of regular expressions F and G such that

- $E = F \cup G$, then $E^R = F^R \cup G^R$
- $E = FG$, then $E^R = G^R F^R$
- $E = F^*$, then $E^R = (F^R)^*$

• Example: Let $E = 01^* \cup 10^*$

\[
E^R = (01^* \cup 10^*)^R = (01^*)^R \cup (10^*)^R \\
= (1^*)^R 0^R \cup (0^*)^R 1^R \\
= (1^R)^* 0 \cup (0^R)^* 1 \\
= 1^* 0 \cup 0^* 1
\]

Thus, $(01^* \cup 10^*)^R = 1^* 0 \cup 0^* 1$