Regular Expressions
Definitions
Equivalence to Finite Automata

RE’s: Introduction
◆ Regular expressions are an algebraic way to describe languages.
◆ They describe exactly the regular languages.
◆ If E is a regular expression, then L(E) is the language it defines.
◆ We’ll describe RE’s and their languages recursively.

RE’s: Definition
◆ Basis 1: If a is any symbol, then a is a RE, and L(a) = {a}.
  ◆ Note: {a} is the language containing one string, and that string is of length 1.
◆ Basis 2: \( \epsilon \) is a RE, and L(\( \epsilon \)) = \{\( \epsilon \}\).
◆ Basis 3: \( \phi \) is a RE, and L(\( \phi \)) = \( \phi \).

RE’s: Definition – (2)
◆ Induction 1: If E₁ and E₂ are regular expressions, then \( E₁ + E₂ \) is a regular expression, and 
  L(E₁ + E₂) = L(E₁) \cup L(E₂).
◆ Induction 2: If E₁ and E₂ are regular expressions, then \( E₁E₂ \) is a regular expression, and 
  L(E₁E₂) = L(E₁)L(E₂).

Concatenation: the set of strings \( wx \) such that w is in L(E₁) and x is in L(E₂).

RE’s: Definition – (3)
◆ Induction 3: If E is a RE, then \( E^* \) is a RE, and L(\( E^* \)) = (L(E))*. 
  ◆ Note: when \( n = 0 \), the string is \( \epsilon \).

Precedence of Operators
◆ Parentheses may be used wherever needed to influence the grouping of operators.
◆ Order of precedence is * (highest), then concatenation, then + (lowest).
Examples: RE's

- \( L(01) = \{01\} \).
- \( L(01+0) = \{01, 0\} \).
- \( L(0(1+0)) = \{01, 00\} \).
  - Note order of precedence of operators.
- \( L(0^*) = \{\epsilon, 0, 00, 000, \ldots \} \).
- \( L((0+10)^*(\epsilon+1)) \) = all strings of 0's and 1's without two consecutive 1's.

Equivalence of RE's and Automata

- We need to show that for every RE, there is an automaton that accepts the same language.
  - Pick the easiest to program automaton type: the \( \epsilon \)-NFA.
- And we need to show that for every automaton, there is a RE defining its language.
  - Pick the most restrictive type: the DFA.

Converting a RE to an \( \epsilon \)-NFA

- Proof is an induction on the number of operators (+, concatenation, *) in the RE.
- We always construct an automaton of a special form (next slide).

An \( \epsilon \)-NFA Subroutine

- No arcs from outside, no arcs leaving.
- Start state: Only 1 state with external predecessors: the only entry.
- Final state: Only 1 state with external successors: the only exit.

RE to \( \epsilon \)-NFA: Basis

- Symbol \( a \):
- \( \epsilon \):
- \( \emptyset \):

RE to \( \epsilon \)-NFA: Induction 1 – Union

- Sub for \( E_1 \):
- Sub for \( E_2 \):
For \( E_1 \cup E_2 \)
**DFA-to-RE**

- A strange sort of induction.
- States of the DFA are assumed to be \(1, 2, \ldots, n\).
- We construct RE's for the labels of restricted sets of paths.
  - **Basis**: single arcs or no arc at all.
  - **Induction**: paths that are allowed to traverse next state in order.

**Restricted k-Paths**

- A k-path is a path through the graph of the DFA that goes through no state numbered higher than k.
- Endpoints are not restricted; they can be any state.

**Example: k-Paths**

0-paths from 2 to 3: RE for labels = 0.
1-paths from 2 to 3: RE for labels = 0 + 11.
2-paths from 2 to 3: RE for labels = \((10)^*0 + 1(01)^*1\)
3-paths from 2 to 3: RE for labels = ??

**k-Path Induction**

- Let \(R_{ij}^k\) be the regular expression for the set of labels of k-paths from state i to state j.
- **Basis**: \(k=0\). \(R_{ij}^0 = \) sum of labels of arc from i to j.
  - \(\emptyset\) if no such arc.
  - But add \(\epsilon\) if \(i=j\).
Example: Basis

- $R_{12}^0 = \emptyset$.
- $R_{11}^0 = \emptyset + \varepsilon = \varepsilon$.

k-Path Inductive Case

- A k-path from i to j either:
  1. Never goes through state k, or
  2. Goes through k one or more times.

$$R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}.$$  

Illustration of Induction

Final Step

- The RE with the same language as the DFA is the sum (union) of $R_{ij}^n$, where:
  1. n is the number of states; i.e., paths are unconstrained.
  2. i is the start state.
  3. j is one of the final states.

Example

- $R_{23}^3 = R_{23}^2 + R_{23}^2(R_{33}^2)^* R_{33}^2 = R_{23}^2(R_{33}^2)^* R_{23}^2$.
- $R_{23}^2 = (10)^*0 + 1(01)^*1$.
- $R_{33}^2 = 0(01)^*(1+00) + 1(10)^*(0+11)$.
- $R_{23}^3 = [(10)^*0 + 1(01)^*1] [(0(01)^*(1+00) + 1(10)^*(0+11))]*$.

Summary

- Each of the three types of automata (DFA, NFA, $\varepsilon$-NFA) we discussed, and regular expressions as well, define exactly the same set of languages: the regular languages.
Algebraic Laws for RE’s

- Union and concatenation behave sort of like addition and multiplication.
  - + is commutative and associative; concatenation is associative.
  - Concatenation distributes over +.
    - $E_1 (E_2 + E_3) = E_1 E_2 + E_1 E_3$
  - Exception: Concatenation is not commutative.

Identities and Annihilators

- $\varnothing$ is the identity for +.
  - $R + \varnothing = R$.
- $\varepsilon$ is the identity for concatenation.
  - $\varepsilon R = R \varepsilon = R$.
- $\varnothing$ is the annihilator for concatenation.
  - $\varnothing R = R \varnothing = \varnothing$. 