CHAPTER 3: Regular Expressions and Languages

Dr. Benjoe A. Juliano

Tel. 530 898-4619 (office)
530 898-6442 (dept)
Fax. 530 898-5995

Juliano@ecst.csuchico.edu
http://www.ecst.csuchico.edu/~juliano

Regular Expressions

Definition

Regular expressions are algebraic descriptions of languages (as opposed to machine-like descriptions such as DFAs and NFAs).

Regular expressions denote languages.

The Operators of Regular Expressions

The union of two languages $L$ and $M$, denoted $L \cup M$, is the set of strings that are either in $L$ or $M$, or both.

The concatenation of languages $L$ and $M$, denoted $L \cdot M$ or simply $LM$, is the set of strings that can be formed by taking any string in $L$ and concatenating it with any string in $M$.

The closure (or star, or Kleene closure) of a language $L$, denoted $L^*$, represents the set of strings that can be formed by taking any number of strings from $L$, possibly with repetitions, and concatenating all of them.
Building Regular Expressions

A regular expression $E$ can be defined as
- constants $\varepsilon$ and $\emptyset$ are RE, denoting the languages $\{\varepsilon\}$ and $\emptyset$, respectively. That is, $L(\varepsilon) = \{\varepsilon\}$, and $L(\emptyset) = \emptyset$.
- If $a$ is any symbol, then $a$ is a regular expression denoting the language $\{a\}$. That is, $L(a) = \{a\}$.
- A variable, usually capitalized and italic such as $L$, is a variable, representing any language.

Building Regular Expressions, continued ...

If $E$ and $F$ are RE, then $E+F$ is a RE denoting the union of $L(E)$ and $L(F)$. So, $L(E+F) = L(E) \cup L(F)$.
- If $E$ and $F$ are RE, then $EF$ is a RE denoting the concatenation of $L(E)$ and $L(F)$. That is, $L(EF) = L(E)L(F)$.
- If $E$ is a RE, then $E^*$ is a RE, denoting the closure of $L(E)$. That is, $L(E^*) = (L(E))^*$.

Precedence of RE operators

The star operator is of highest precedence.
- The concatenation or “dot” operator is next in precedence.
- Finally, all unions (+ operators) are grouped with their operands.

From DFAs to REs

- From Figure 3.1 of [10], Hopcroft, Motwani, & Ullman, 2001.
FAs and REs

From DFAs to REs, continued...

Hence, to show that REs define the same class of languages as DFA, NFA, and ε-NFA, one must show that

Every language defined by one of these automata is also defined by a regular expression. For this proof, one can assume the language is accepted by some DFA.

Every language defined by a regular expression is defined by one of these automata. For this part of the proof, the easiest is to show that there is an ε-NFA accepting the same language.

Proof of Theorem 3.4, continued...

A path whose label is in \( L(R^{\delta \in L}_{\delta}) \)

State (1,…,n)

Path

Proof of Theorem 3.4

If \( L = L(A) \) for some DFA \( A \), then there is a regular expression \( R \) such that \( L = L(R) \).

Proof

Let \( Q_n = \{1,2,…,n\} \).

Let \( R^{\delta \in L}_{\delta} \) denote a RE such that \( L(R^{\delta \in L}_{\delta}) \) is the set of strings \( w \) such that \( w \) is the label of a path from state \( i \) to state \( j \) in \( A \), and that path has no intermediate node whose number is greater than \( k \). (Note: States \( i \) and \( j \) are not “intermediate” nodes.)

Inductive definition to construct expressions in \( L(R^{\delta \in L}_{\delta}) \)

BASIS (\( k=0 \)): (paths with no intermediate states)

1. For \( i \neq j \), no \( a \in \Sigma \) such that \( \delta(i,a)=j \) (null path or loop)

2. There exists \( a \in \Sigma \) such that \( \delta(i,a)=j \) (arc between \( i \) & \( j \))

Find \( a \in \Sigma \) such that \( \delta(i,a)=j \)

a. If there is no such symbol \( a \), then \( R^{\delta \in L}_{\delta} = \emptyset \)

b. If there is exactly one such symbol \( a \), then \( R^{\delta \in L}_{\delta} = a \)

c. If \( \delta(i,a)=j \) for \( 1 \leq s \leq k \), then \( R^{\delta \in L}_{\delta} = a_1 + a_2 + w + a_k \)
Proof of Theorem 3.4, continued ...

INDUCTION: Suppose there is a path from state $i$ to state $j$ that goes through no state higher than $k$.
- The path does not go through state $k$ at all; hence, the label of the path is in the language $R^k_{i j}$.
- The path goes through state $k$ at least once.

Then, $R = N_{jg} R^k_{i j}$

Example:

$k = 0$:

$R^0_{11} = \varepsilon A 1$

$k = 1$:

$R^0 = 1 \varepsilon A 1 0$

$R^1_{11} = A 1 A 0 A 1 E 0 A 1 E 0$

$R^1_{12} = 0 A 0 A 0 A 1 E 0 0 1$

$R^1_{21} = 0 A 0 A 1 E 0 A 1 E 0$

$R^1_{22} = A 0 A 1 E 0 0 1$

$R^0 = A 0 A 0 A 1 E$

Example:

$k = 2$:

$R^2_{11} = A 1 0 A 0 A 1 E 0 0 A 1 E$

$R^2_{12} = 0 A 0 A 0 A 0 A 1 E 0 0 A 1 E$

$R^2_{21} = 0 A 0 A 0 A 0 A 1 E 0 A 1 E$

$R^2_{22} = A 0 A 0 A 1 E 0 A 1 E 0 0 A 1 E$

$R^2 = A 0 A 0 A 1 E$
FAs and REs

Example:

\[
R \in \Sigma = 1^* 0 \Phi A 1^* \epsilon
\]

Finally:

regular expression, \( R = R_{12} \in \epsilon = 1^* 0 \Phi A 1^* \epsilon \)

Converting DFAs to REs by Eliminating States

Motivation: Method of Theorem 3.4 to construct REs is expensive ...

- Have to construct \( n^3 \) expressions for \( n \)-state FA
- Length of expression can grow by a factor of 4, on the average, with each of the \( n \) inductive steps — expressions can reach on the order of \( 4^n \) symbols.

Method: Consider automata that have regular expressions as labels.

- Can be accomplished by systematically eliminating states.
FAs and REs

Converting DFAs to REs by Eliminating States, continued ...

Procedure:

1. For all \( q \in F \), apply the reduction process to produce an equivalent automaton with RE labels on the arcs. Eliminate all states except \( q \) and start state \( q_0 \).

2. If \( q \neq q_0 \) then we have a generic two-state automaton:

\[
(R + SU^*)T^*SU^*
\]

From Figure 3.9 of IATLC, Hopcroft, Motwani, & Ullman, 2001.

FAs and REs

Example:

Consider an NFA that accepts all strings of 0's and 1's such that either the second or third position from the end has a 1.

With RE labels:

\[
\begin{align*}
\text{Start} & \quad 0 + 1 \\
A & \quad 0, 1 \quad B \\
C & \quad 0, 1 \quad D
\end{align*}
\]

Eliminate state \( B \) with predecessor \( A \) and successor \( C \)

\[
S, R_{11} = \emptyset
\]

(See Figure 3.7 for labels.)

resulting in:

\[
\begin{align*}
\text{Start} & \quad 0 + 1 \\
A & \quad 1 \quad B \\
C & \quad 0 + 1 \quad D
\end{align*}
\]
**FAs and REs**

**Example, continued ...:**

*(Branch 1)* Eliminate state $C$ to derive a two-state automaton with states $A$ and $D$.

From: $0 + 1$

Start $\xrightarrow{1(0+1)} C \xrightarrow{0+1} D$

To: $\xrightarrow{R} 0+1$

$S \xrightarrow{T,U} = \emptyset$

$(R^+SU^*T)^*SU^* = R^*S = (0+1)^*1(0+1)(0+1)$

*(Branch 2)* Eliminate state $D$ to derive a two-state automaton with states $A$ and $C$.

From: $0 + 1$

Start $\xrightarrow{1(0+1)} C \xrightarrow{0+1} D$

To: $\xrightarrow{R} 0+1$

$S \xrightarrow{T,U} = \emptyset$

$(R^+SU^*T)^*SU^* = R^*S = (0+1)^*1(0+1)$

**Answer:** $(0+1)^*1(0+1)(0+1) + (0+1)^*1(0+1)(0+1)$

---

**Theorem 3.7**

Every language defined by a regular expression is also defined by a finite automaton.

**Converting REs to Automata**

Suppose $L = L(R)$ for a RE $R$. By structural induction on $R$, $L = L(E)$ for some $\varepsilon$-NFA $E = (Q, \Sigma, \delta, q_0, F_E)$ where

- $|F_E| = 1$ (say $F_E = \{f\}$)
- There is no $a \in \Sigma$ such that for $q \in Q, \delta(q,a) = q_0$
- There is no $a \in \Sigma$ such that for $q \in Q, \delta(f,a) = q$

---

**Converting REs to Automata, continued ...**

**Basis:**

$\varepsilon$

$\emptyset$

$a$
### FAs and REs

**Converting REs to Automata, continued ...**

**INDUCTION:**

**Case 1:** The expression is $R + S$

\[
\begin{array}{c}
  \varepsilon \\
  R \\
  S \\
  \varepsilon
\end{array}
\]

**Case 2:** The expression is $RS$

**Case 3:** The expression is $R^*$


### Applications of REs

**Regular Expressions in UNIX**

- Facilitates representation of *character classes*
  - Symbol . (dot) “any character”
  - Sequence $[a_1, a_2, \ldots, a_k]$ RE $a_1 + a_2 + \ldots + a_k$
  - $[0-9]$ digits
  - $[A-Z]$ uppercase letters
  - $[A-Za-z0-9]$ digits and letters
  - $[:digit:]$ same as $[0-9]$
  - $[:alpha:]$ same as $[A-Za-z]$
  - $[:alnum:]$ same as $[A-Za-z0-9]$
Applications of REs

- **Regular Expressions in UNIX, continued ...**
  - Sampling of operators
    - union
    - ? “zero or one of”
      - Example: $R?$ means “$\varepsilon + R$”
    - + “one or more of”
      - Example: $R+$ means “$RR*$”
    - {n} “n copies of”
      - Example: $R\{5\}$ means “$RRRRR$”

- **Lexical Analysis**
  - A **lexical analyzer** is the component of a compiler that scans the source program and recognizes all tokens (e.g. keywords and identifiers).
  - Examples:
    - Unix `lex` command
    - GNU `flex` command
  - Process: Use RE-to-DFA conversion to generate an efficient function that breaks source programs into tokens.

- **Finding Patterns in Text**
  - Example: Street addresses typically end in “Street” (or its abbreviation), “Avenue,” etc.
    - `Street|St\.|Avenue|Ave\.|Road|Rd\.`
  - Example: Names of streets
    - `'[A-Z][a-z]* ( [A-z][a-z]*)*'`
  - Example: Putting things together ...
    - `'[A-Z][a-z]* ( [A-z][a-z]*)* (Street|St\.|Avenue|Ave\.|Road|Rd\.)'`

- **Algebraic Laws for REs**
  - **Associativity and Commutativity**
    - **Commutative Law for Union**
      - $L+M = M+L$
    - **Associative Law for Union**
      - $(L+M)+N = L+(M+N)$
    - **Associative Law for Concatenation**
      - $(LM)N = L(MN)$
Algebraic Laws for REs

- **Identities and Annihilators**
  - **Identity for Union**
    \[ \emptyset + L = L + \emptyset = L \]
  - **Identity for Concatenation**
    \[ \epsilon L = L \epsilon = L \]
  - **Annihilator for Concatenation**
    \[ \emptyset L = L \emptyset = \emptyset \]

- **Distributive Laws**
  - **Left Distributive Law of Concatenation over Union**
    \[ L(M + N) = LM + LN \]
  - **Right Distributive Law of Concatenation over Union**
    \[ (M + N)L = ML + NL \]

- **The Idempotent Law**
  - **Idempotence for Union**
    \[ L + L = L \]
  - **Theorem 3.11**
    - If \( L, M, \) and \( N \) are any languages, then
    \[ L(M \cap N) = LM \cap LN \]

- **Exponentiation**
  - **Kleene star / Kleene closure / Star closure**
    \[ A^n = \begin{cases} 1 & n = 0 \\ AA^{n-1}B & n \in \mathbb{N} \end{cases} \]
  - **Positive closure / Plus closure**
    \[ A^+ = \bigcup_{j=1}^{\infty} A^j \]
Algebraic Laws for REs

Laws Involving Closures

- \((L^*)^* = L^*\)
- \(\emptyset^* = \epsilon\)
- \(\epsilon^* = \epsilon\)
- \(L^+ = LL^* = L^*L\)
- \(L^* = L^+ + \epsilon\)
- \(L? = \epsilon + L\)

Example: Equivalences involving closures

- \((a+b)^* = (a+b)^* + (a+b)^*\)
- \((a+b)^* = (a+b)^* + a^*\)
- \((a+b)^* = (a+b)^*(a+b)^*\)
- \((a+b)^* = a(a+b)^* + b(a+b)^* + \emptyset^*\)
  
  - All strings that start with an \(a\)
  - All strings that start with an \(b\)
- \((a+b)^* = (a+b)^*ab(a+b)^* + b^*a^*\)
  
  - All strings that contain \(ab\) as a substring
  - All strings without \(ab\) as a substring.

Closure of Closures

- \(\emptyset A^\epsilon A^{\emptyset} = A^\emptyset\)
- \(\emptyset A^\epsilon A^{\epsilon} = A^\epsilon\)

No new strings are added to either \(A^*\) or \(A^+\) by Kleene or Plus closure ...

Discovering Laws for Regular Expressions

Theorem 3.13

Let \(E\) be a regular expression with variables \(L_1, L_2, \ldots, L_m\). Form concrete RE \(C\) by replacing each occurrence of \(L_i\) by the symbol \(a_i\) for \(i = 1, 2, \ldots, m\).

Then for any languages \(L_1, L_2, \ldots, L_m\), every string \(w\) in \(L(E)\) can be written \(w_1w_2w_3w_k\), where each \(w_i\) is in one of the languages, say \(L_j\), and the string \(a_1a_2a_3\ldots a_k\) is in \(L(C)\).
Algebraic Laws for REs

Discovering Laws for Regular Expressions

Theorem 3.13

Less formally, construct $L(E)$ by starting with each string in $L(C)$, say $a_1 a_2 w a_3$, and substituting for each of the $a_i$’s any string from the corresponding language $L_i$.

Test for a Regular Expression Algebraic Law

To test whether $E=F$ is true, where $E$ and $F$ are two regular expressions with the same set of variables

1. Convert $E$ and $F$ to concrete regular expressions $C$ and $D$, respectively, by replacing each variable by a concrete symbol.
2. Test whether $L(C) = L(D)$. If so, then $E=F$ is a true law, and if not, then the “law” is false.

Copyright and Intellectual Property Notice

This document and its contents are the Intellectual Property (IP) of Dr. Benjoe A. Juliano of the Department of Computer Science at California State University, Chico (CSUC). Dr. Juliano claims exclusive moral rights of ownership under current Copyright Laws (Title 17 of the United States Code and 1998 Digital Millennium Copyright Act) and IP Policies/Guidelines (CSUC EM83-08, EM87-07, and Article 39 of the CFA/CSU Contract) including, but not limited to:

- the exclusive right to copy, reproduce, and/or distribute this document;
- the right to be identified as the creator of this work (the right of attribution);
- the right to take action against false attribution; and
- the right to object to derogatory treatment of this work (the right of integrity).