

CSCI 256: Theory of Computing

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>

B. Juliano © 2002, based on notes by J. Ullman



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.

B. Juliano © 2002, based on notes by J. Ullman



Regular Expressions

- **The Operators of Regular Expressions**
 - The *union* of two languages L and M , denoted $LN\mathbf{M}$, is the set of strings that are either in L or M , or both.
 - The *concatenation* of languages L and M , denoted $L\mathbf{\cdot}M$ 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 .

B. Juliano © 2002, based on notes by J. Ullman



Regular Expressions

- **The Operators of Regular Expressions.**
continued ...
 - 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.

B. Juliano © 2002, based on notes by J. Ullman



Regular Expressions

- **Building Regular Expressions**
 - A regular expression E can be defined as
 - constants ϵ and \emptyset are RE, denoting the languages $\{\epsilon\}$ and \emptyset , respectively. That is, $L(\epsilon)=\{\epsilon\}$, 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.

B. Juliano © 2002, based on notes by J. Ullman



5

Regular Expressions

- **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, $L(E^*) = (L(E))^*$.

B. Juliano © 2002, based on notes by J. Ullman

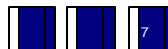


6

Regular Expressions

- **Building Regular Expressions, continued ...**
 - If E is a RE, then (E) , a **parenthesized** E , is a RE, denoting the same language as E . Formally, $L((E))=L(E)$.
- **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.

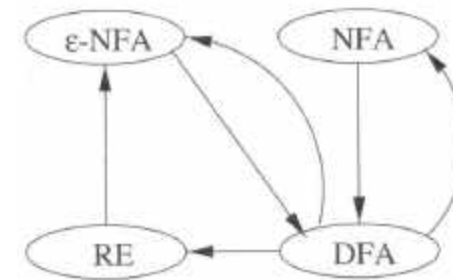
B. Juliano © 2002, based on notes by J. Ullman



7

FAs and REs

- **From DFAs to REs**



B. Juliano © 2002, based on notes by J. Ullman



8

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.

B. Juliano © 2002, based on notes by J. Ullman

9

FAs and REs

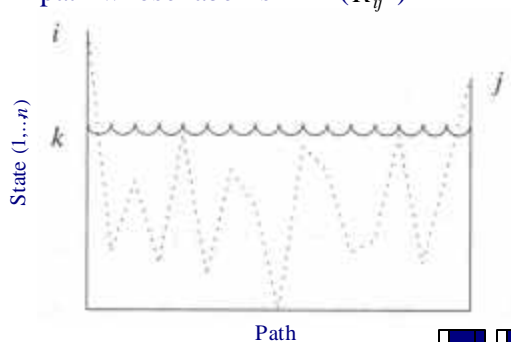
- Theorem 3.4**
 - If $L=L(A)$ for some DFA A , then there is a regular expression R such that $L=L(R)$.
- Proof**
 - Let $Q_A = \{1, 2, \dots, n\}$.
 - Let $R_{ij}^{\Phi \epsilon}$ denote a RE such that $L(R_{ij}^{\Phi \epsilon})$ is the set of strings w such that w is the label of a path from state 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.)

B. Juliano © 2002, based on notes by J. Ullman

10

FAs and REs

- Proof of Theorem 3.4, continued ...**
 - A path whose label is in $L(R_{ij}^{\Phi \epsilon})$



B. Juliano © 2002, based on notes by J. Ullman

11

FAs and REs

- Proof of Theorem 3.4, continued ...**
 - Inductive definition to construct expressions in $L(R_{ij}^{\Phi \epsilon})$
 - BASIS** ($k=0$): (paths with no intermediate states)
 - For $i \neq j$, no $a \in \Sigma$ such that $\delta(i, a) = j$ (null path or loop)
 - There exists $a \in \Sigma$ such that $\delta(i, a) = j$ (arc between i & j)
 Find $a \in \Sigma$ such that $\delta(i, a) = j$
 - If there is no such symbol a , then $R_{ij}^{\Phi \epsilon} = \emptyset$
 - If there is exactly one such symbol a , then $R_{ij}^{\Phi \epsilon} = a$
 - If $\delta(i, a_s) = j$ for $1 \leq s \leq k$, then $R_{ij}^{\Phi \epsilon} = a_1 + a_2 + \dots + a_k$

B. Juliano © 2002, based on notes by J. Ullman

12

FAs and REs

From Figure 3.3 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

- **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 $R_{ij}^{\leq k-1}$.
 - The path goes through state k at least once.



$$R_{ij}^{\leq k} = R_{ij}^{\leq k-1} \cup R_{ik}^{\leq k-1} R_{kk}^{\leq k-1*} R_{kj}^{\leq k-1}$$

- Then, $R = \bigcup_{i,j \in F_A} R_{ij}^{\leq k}$

B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figure 3.4 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

- **Example:**



- $k=0$: $R_{11}^{\leq 0} = \epsilon$, $R_{12}^{\leq 0} = \emptyset$, $R_{21}^{\leq 0} = \emptyset$, $R_{22}^{\leq 0} = \epsilon$

$$R^{\leq 0} = \begin{pmatrix} \epsilon & \emptyset \\ \emptyset & \epsilon \end{pmatrix}$$

B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figure 3.4 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

- **Example:**



- $k=1$: $R_{ij}^{\leq 1} = R_{ij}^{\leq 0} \cup R_{i1}^{\leq 0} R_{1j}^{\leq 0}$

$$R_{11}^{\leq 1} = \epsilon \cup 1^* \epsilon = 1^*$$

$$R_{12}^{\leq 1} = \emptyset \cup 0 \epsilon = 0$$

$$R_{21}^{\leq 1} = \emptyset \cup \emptyset \epsilon = \emptyset$$

$$R_{22}^{\leq 1} = \epsilon \cup \emptyset 1^* \emptyset = \epsilon$$

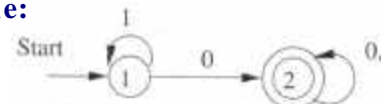
$$R^{\leq 1} = \begin{pmatrix} 1^* & 0 \\ \emptyset & \epsilon \end{pmatrix}$$

B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figure 3.4 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

- **Example:**



- $k=2$: $R_{ij}^{\leq 2} = R_{ij}^{\leq 1} \cup R_{i1}^{\leq 1} R_{1j}^{\leq 1}$

$$R_{11}^{\leq 2} = 1^* \cup 1^* 0 1^* = 1^*$$

$$R_{12}^{\leq 2} = 0 \cup 1^* 0 1^* 0 = 0 \cup 1^* 0 1^*$$

$$R_{21}^{\leq 2} = \emptyset \cup \emptyset 1^* \emptyset = \emptyset$$

$$R_{22}^{\leq 2} = \epsilon \cup \emptyset 1^* \emptyset = \epsilon$$

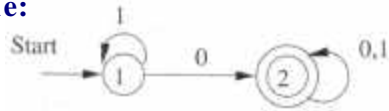
$$R^{\leq 2} = \begin{pmatrix} 1^* & 1^* 0 1^* \\ \emptyset & \epsilon \end{pmatrix}$$

B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figure 3.4 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

- Example:



$$R^{\Phi E} = \begin{matrix} | & 1^* & 1^* 0 \Phi A 1 \hat{E} \\ | & \hat{Y} & \Phi A 1 \hat{E} \end{matrix}$$

- Finally:

$$\text{regular expression, } R = R_{12}^{\Phi E} = 1^* 0 \Phi A 1 \hat{E}$$

B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

- 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.

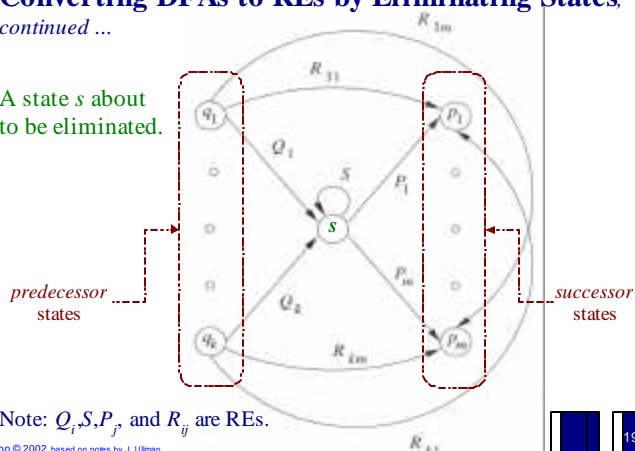
B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figure 3.7 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

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

A state s about to be eliminated.



Note: Q_i, S, P_j , and R_{ij} are REs.

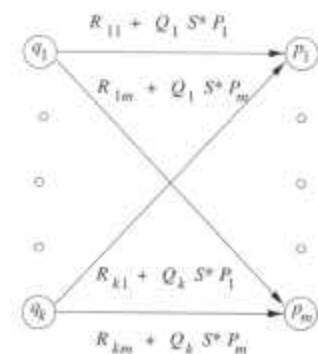
B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figure 3.8 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

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

After state s has been eliminated.



B. Juliano © 2002, based on notes by J. Ullman

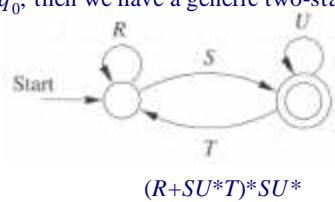
From Figure 3.9 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

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:



B. Juliano © 2002, based on notes by J. Ullman

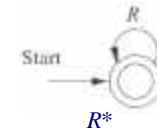
From Figure 3.10 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

FAs and REs

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

- Procedure:

3. If $q_0 \in F$, apply the **reduction process** to get rid of every state but q_0 ; this leaves a generic one-state automaton:



4. The desired regular expression is the sum (union) of all the expressions derived from the reduced automata for each accepting state, by rules (2) and (3).

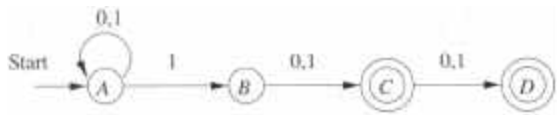
B. Juliano © 2002, based on notes by J. Ullman

From Figures 3.11 and 3.12 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

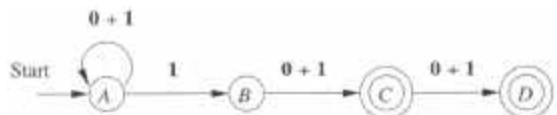
FAs and REs

- **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:



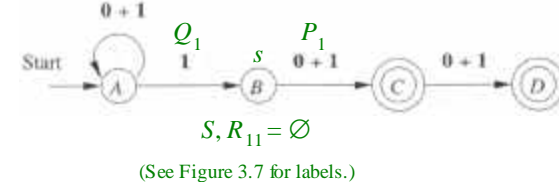
B. Juliano © 2002, based on notes by J. Ullman

From Figures 3.12 and 3.13 of ATLC, Hopcroft, Motwani, & Ullman, 2001.

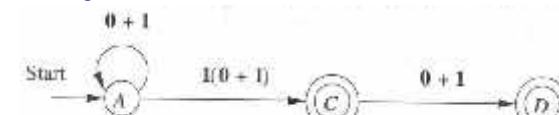
FAs and REs

- **Example, continued ...:**

Eliminate state B with predecessor A and successor C



resulting in:



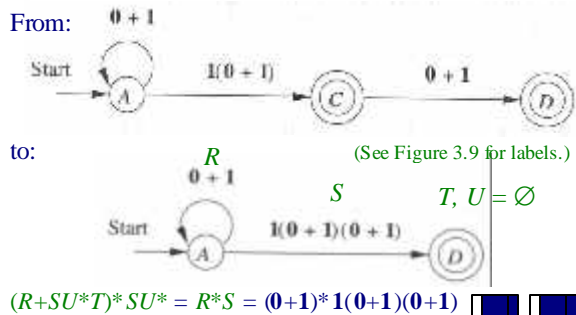
B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figures 3.13 and 3.14 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

▪ **Example, continued ...:**

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



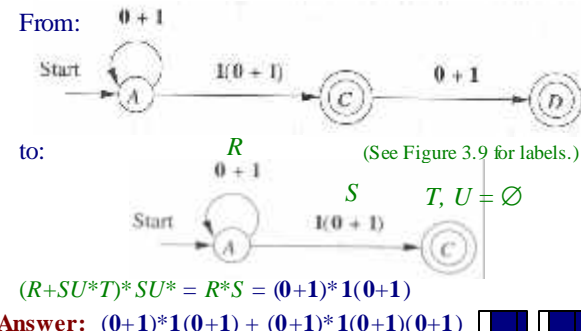
B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figures 3.13 and 3.14 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

▪ **Example, continued ...:**

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



B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

▪ **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 ϵ -NFA

$E=(Q_E, \Sigma, \delta, q_0, F_E)$ where

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

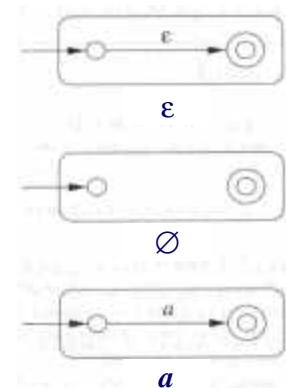
B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figures 3.16 of *ITLC*, Hopcroft, Motwani, & Ullman, 2001.

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

▪ **BASIS:**

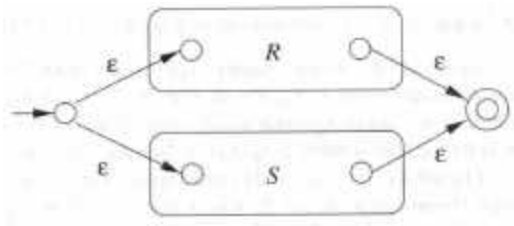


B. Juliano © 2002, based on notes by J. Ullman

FAs and REs

From Figures 3.17 of *ITC*, Hopcroft, Motwani, & Ullman, 2001.

- Converting REs to Automata, *continued* ...
 - INDUCTION:
 - Case 1:** The expression is $R + S$



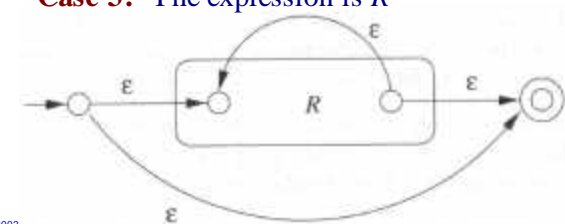
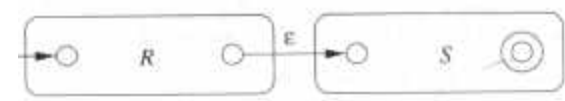
B. Juliano © 2002, based on notes by J. Ullman

29

FAs and REs

From Figures 3.17 of *ITC*, Hopcroft, Motwani, & Ullman, 2001.

- Converting REs to Automata, *continued* ...
 - Case 2:** The expression is RS
 - Case 3:** The expression is R^*



B. Juliano © 2002

30

FAs and REs

- Converting REs to Automata, *continued* ...
 - Case 4:** The expression is (R) ; then, the automaton for R also serves as the automaton for (R) .
 - Example:**
 - $(0+1)01$
 - $(0+1)^*1(0+1)$

B. Juliano © 2002, based on notes by J. Ullman

31

Applications of REs

- Regular Expressions in UNIX
 - Facilitates representation of *character classes*

symbol <code>.</code> (dot)	“any character”
sequence $[a_1, a_2, \dots, a_k]$	RE $a_1 + a_2 + \dots + a_k$
<code>[0-9]</code>	digits
<code>[A-Z]</code>	uppercase letters
<code>[A-Za-z0-9]</code>	digits and letters
<code>[:digit:]</code>	same as <code>[0-9]</code>
<code>[:alpha:]</code>	same as <code>[A-Za-z]</code>
<code>[:alnum:]</code>	same as <code>[A-Za-z0-9]</code>

B. Juliano © 2002, based on notes by J. Ullman

32

Applications of REs

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

B. Juliano © 2002, based on notes by J. Ullman

33

Applications of REs

- 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.

B. Juliano © 2002, based on notes by J. Ullman

34

Applications of REs

- 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\.)'`

B. Juliano © 2002, based on notes by J. Ullman

35

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*
 $(LM)N = L(MN)$

B. Juliano © 2002, based on notes by J. Ullman

36

Algebraic Laws for REs

- Identities and Annihilators

- Identity for Union*

$$\emptyset + L = L + \emptyset = L$$

- Identity for Concatenation*

$$\varepsilon L = L\varepsilon = L$$

- Annihilator for Concatenation*

$$\emptyset L = L\emptyset = \emptyset$$

Algebraic Laws for REs

- Distributive Laws

- Left Distributive Law of Concatenation over Union*

$$L(M + N) = LM + LN$$

- Right Distributive Law of Concatenation over Union*

$$(M + N)L = ML + NL$$

Algebraic Laws for REs

- The Idempotent Law

- Idempotence for Union*

$$L + L = L$$

- Theorem 3.11**

- If L , M , and N are any languages, then

$$L(M N N) = L M N L N$$

Algebraic Laws for REs

- Exponentiation

$$A^n = \begin{cases} \emptyset, & n=0 \\ AA^{n-1}, & n>0 \end{cases}$$

- Kleene star / Kleene closure / Star closure

$$A^C = \bigcup_{i=0}^{\infty} A^i$$

- Positive closure / Plus closure

$$A^A = \bigcup_{i=1}^{\infty} A^i$$

Algebraic Laws for REs

Laws Involving Closures

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

B. Juliano © 2002, based on notes by J. Ullman

41

Algebraic Laws for REs

Closure of Closures

$$\begin{aligned} \mathcal{C}^A \mathcal{C}^A &= A^A \\ \mathcal{C}^A \mathcal{C}^B &= A^B \\ \mathcal{C}^C \mathcal{C}^A &= A^C \\ \mathcal{C}^C \mathcal{C}^B &= A^C \end{aligned}$$

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

B. Juliano © 2002, based on notes by J. Ullman

42

Algebraic Laws for REs

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

B. Juliano © 2002, based on notes by J. Ullman

43

Algebraic Laws for REs

Discovering Laws for Regular Expressions

- **Theorem 3.13**
- Let E be a regular expression with variables L_1, L_2, \dots, L_m . Form concrete RE C by replacing each occurrence of L_i by the symbol a_i , for $i=1, 2, \dots, m$. Then for any languages L_1, L_2, \dots, L_m , every string w in $L(E)$ can be written $w_1w_2w_3 \dots w_k$, where each w_i is in one of the languages, say L_j , and the string $a_{j_1}a_{j_2} \dots a_{j_k}$ is in $L(C)$.

B. Juliano © 2002, based on notes by J. Ullman

44

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 $a_{j_1}a_{j_2}\dots a_{j_k}$, and substituting for each of the a_{j_i} 's any string from the corresponding language L_{j_i} .

Algebraic Laws for REs

- **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, EM97-07, 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).