Properties of Regular Languages

SITE : http://www.info.blois.univ-tours.fr/˜mirian/

Algebraic Laws for Regular Expressions

- Two expressions with variables are equivalent if whatever languages we substitute for the variables the results of the two expressions are the same language.
  Examples in the algebra of arithmetic: $1 + 2 = 2 + 1$ or $x + y = y + x$.
- Like arithmetic expressions, the regular expressions have a number of laws that work for them. Many of these are similar to the laws of arithmetic, if we think of union as additional and concatenation as multiplication.

Associativity and Commutativity

- **Commutativity** is the property of an operator that says we can switch the order of its operands and get the same result.
- **Associativity** is the property of an operator that allow us regroup the operands when the operator is applied twice.

Associativity and Commutativity

- For regular expressions, we have:
  - $L + M = M + L$
    Commutative law for union: we may make the union of two languages in either order
  - $(L + M) + N = L + (M + N)$
    Associative law for union: we may take the union of three languages either by taking the union of the first two initially, or taking the union of the last two initially.
    Together with the commutative law **we can take the union of any collection of languages with any order and grouping, and the result will be the same.**
    Intuitively, a string is in $L_1 \cup L_2 \ldots \cup L_k$ iff it is in one or more of the L_is.
  - $(L.M).N = L.(M.N)$
    Associative law for concatenation: we can concatenate three-languages by concatenating either the first two or the last two initially
  - **Clearly the law $L.M = M.L$ is FALSE**

Identities and Annihilator

- An **identity** for an operator is a value that when the operator is applied to the identity and some other value, the result is the other value.
- An **annihilator** for an operator is value that when the operator is applied to the annihilator and some other value, the result is the annihilator.
- For regular expressions we have:
  - $\emptyset + L = L + \emptyset = L$
  - $\epsilon.L = L.\epsilon = L$
  - $\emptyset.L = L.\emptyset = \emptyset$

Distributive laws

- A distributive law involves two operators and asserts that one operator can be pushed down to be applied to each argument of the other operator individually
- For regular expressions we have:
  - $L.(M+N) = LM + LN$
  - $(M+N).L = ML + NL$

The Idempotent Law

- An operator is idempotent if the result of applying it to two of the same values as arguments is that value.
- Idempotent Law for union: $L + L = L$
  If we take the union of two identical expressions, we can replace them by one copy of the expression.

Laws Involving Closures

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

  Proof: Recall that L^+ is defined to be $L + LL + LLL + \ldots$. Also, $L^* = \epsilon + L + LL + LLL + \ldots$. Thus,

  $$LL^* = L\epsilon + LL + LLL + \ldots$$

  When we remember that $L\epsilon = L$, we see that the infinite expressions for LL^* and L^+ are the same. That proves $L^+ = LL^*$. The proof $L^+ = L^*L$ is similar
- $L^* = L^+ + \epsilon$
- $L? = \epsilon + L$

Discovering Laws for Regular Expressions

- There is an infinite variety of laws about regular expressions that might be proposed. Is there a general methodology that will make our proofs of correct laws easy?
- This technique is closely tied to the regular-expressions operators, and CANNOT be extended to expressions involving some OTHER operators (such as intersection)

The test for a regular expression algebraic law

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

1. Convert E and F to concrete regular expressions (*i.e.*, one with no variables) 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
   Notice that this is an ad-hoc method to decide equality of the pairs or languages. If the languages are *not* the same, than it is sufficient to provide one counterexample: a single string that is in one language but not in the other.
3. Examples: Prove or disprove
   (a) $(L + M)^* = (L^* M^*)^*$
   (b) $L^* = L^* L^*$
   (c) $L + ML = (L + M)L$

Closure Properties of Regular Languages

Let L and M be regular languages. Then the following languages are all regular:

- Union: $L \cup M$
- Intersection: $L \cap M$
- Complement: $\overline{N}$
- Difference: $L \setminus M$
- Reversal: $L^R = w^R : w \in L$
- Closure: L^*
- Concatenation: $L.M$
- Homomorphism:

$$h(L) = \{h(w) \mid w \in L, h \text{ is a homomorphism } \}$$

- Inverse homomorphism:

$$h^{-1}(L) = \{w \in \Sigma \mid h(w)L, h : \Sigma \rightarrow \Delta \text{ is a homomorphism}\}$$

Closure under Union

For any regular languages L and M, then $L \cup M$ is regular.

Proof: Since L and M are regular, they have regular expressions, say:

- Let $L = L(E)$ and $M = L(F)$. Then $L \cup M = L(E + F)$ by the definition of the $+$ operator.

Closure under complementation

If L is a regular language over alphabet Σ then $\overline{L} = \Sigma^* \setminus L$ is also regular.

Proof: Let L be recognized by a DFA

$$A = (Q, \Sigma, \delta, q_0, F).$$

Then $\overline{L} = L(B)$ where B is the DFA

$$B = (Q, \Sigma, \delta, q_0, Q \setminus F).$$

That is, B is exactly like A, but the accepting states of A have become the nonaccepting states of B and vice-versa.

Then w is in $L(B)$ iff $\hat{\delta}(q_0, w)$ is in $Q \setminus F$, which occurs iff w is not in $L(A)$.

Closure under intersection

If L and M are regular languages, then so is $L \cap M$.

Proof: Let L be recognized by the DFA

$$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$

and M by the DFA

$$A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$$

- We assume that the alphabets of both automata are the same ; that is Σ is the union of the alphabets of L and M, if they are different.
- We also assume w.l.o.g. that both automata are deterministic.

- We shall construct an automaton A that simulates both A_L and A_M.
- The states of A are pairs of states, the first from A_L and the second from A_M.
- If A_L goes from state p to state s on reading a, and A_M goes from state q to state t on reading a, then $A_{L \cap M}$ will go from state (p, q) to state (s, t) on reading a.
- The start state of A is the pair of start states of A_L and A_M.
- Since we want to accept iff both automata accept, we select as accepting states of A all pairs (p, q) such that p is an accepting state of A_L and q is an accepting state or A_M.

Formally

$$A_{L\cap M} = (Q_L \times Q_M, \Sigma, \delta, (q_L, q_M), F_L \times F_M),$$

where

$$\delta((p,q),a) = (\delta_L(p,a), \delta_M(q,a))$$

By induction on $|w|$ it is possible to prove that

$$\hat{\delta}_{L\cap M}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w))$$

But A accepts w iff $\hat{\delta}_{L\cap M}((q_L, q_M), w)$ is a pair of accepting states. That is $\hat{\delta}_L(q_L, w)$ must be un F_L and $\hat{\delta}_M(q_M, w)$ must be un F_M.

Thus w is accepted by A iff w is accepted by A_L and by A_M. **Thus A accepts the intersection of L and M.**

Closure under Difference

If L and M are regular languages, then so is $L \setminus M$.

Proof: Observe that $L \setminus M = L \cap \overline{M}$. We already know that regular languages are closed under complement and intersection.

Reversal

- The **reversal** of a string $a_1a_2\dots a_n$ is the string written backwards $a_na_{n-1}\dots a_1$.
- We use w^R for the reversal of string w.
- The reversal of a language L, written L^R, is the language consisting of the reversals of all its strings
- Example:
  $L = \{001, 10, 111\}$
  $L^R = \{100, 01, 111\}$

Closure under reversal

If L is a regular languages, then so is L^R.

Proof 1

Let L be recognized by an fsa A. Turn A into an fsa for L^R , by:

1. Reversing all arcs.
2. Make the old start state the new sole accepting state.
3. Create a new **start state** p_0 , with $\delta(p_0, \epsilon) = F$ (the old accepting states).

Closure under reversal

Proof 2

- Assume L is defined by a regular expression E.
- We show that there is another regular expression E^R such that

$$L(E^R) = (L(E))^R$$

that is, the language of E^R is the reversal of the language of E.

Basis: If E is ϵ, $\emptyset$ of **a**, then $E^R = E$.

Induction: There are three cases, depending on the form of E

Proof (continue)

1. $E = F + G$. Then $E^R = F^R + G^R$.
   Justification: The reversal of the union of two languages is obtained by computing the reversal of the two languages and taking the union of these languages.
2. $E = F.G$. Then $E^R = G^R.F^R$

   Note: We reverse the order of the two languages, as well as reversing the languages themselves.
   Justification: In general, if a word $w \in L(E)$ is the concatenation of $w_1 \in L(F)$ and $w_2 \in L(G)$, then $w^R = w_2^R.w_1^R$.
3. $E = F^*$. Then $E^R = (F^R)^*$
   Justification:
   - Any string $w \in L(E)$ can be written as $w_1 w_2 \dots w_n$, where each w_i is in $L(F)$. But $w^R = w_n^R. \dots .w_2^R.w_1^R$ and each w_i^R is in $L(E^R)$, so w^R is in $L((F^R)^*)$.
   - Conversly, any string in $L((F^R)^*)$ is of the form $w_1 w_2 \dots w_n$ where each w_i is the reversal of a string in $L(F)$.The reversal of this string is in $L(F^*)$ which is in $L(E)$.
     We have shown that a string is in $L(E)$ iff its reversal is in $L((F^R)^*)$.

Homomorphisms

A homomorphism on Σ is a function $h : \Sigma^* \longrightarrow \Theta^*$, where Σ and Θ are alphabets. Let $w = a_1 a_2 \ldots a_n \in \Sigma^*$. Then

$$h(w) = h(a_1)h(a_2)\ldots h(a_n)$$

and

$$h(L) = \{h(w) \mid w \in L\}$$

Example: Let $h : \{0,1\}^* \longrightarrow \{a,b\}^*$ be defined by $h(0) = ab$, and $h(1) = \epsilon$. Now

- $h(0011) = abab$.
- $h(L(10^*1)) = L((ab)^*)$

If L is a regular language over alphabet Σ and h is a homomorphism on Σ, then $h(L)$ is also regular.

Proof

- Let $L = L(R)$ for some RE R.
- We claim that $h(R)$ defines the language $h(L)$.
- In general, if E is the regular expression with symbols in Σ, let $h(E)$ be the expression obtained by replacing each symbol a of Σ by $h(a)$.
- The proof is a structural induction that says whenever we take a subexpression E of R and apply h to it to get $h(E)$, the language $h(E)$ is the same we get if we apply h to the language $L(E)$.
- Formally $L(h(E)) = h(L(E))$.

Proof (cont.)

Basis: If E is ϵ or $\emptyset$, then $h(E) = E$ and $L(h(E)) = L(E) = h(L(E))$.
If E is **a**, then $L(E) = \{a\}$ and

$$L(h(E)) = L(h(\textbf{a})) = \{h(a)\} = h(L(E))$$

Induction: There are three cases, depending on the form of E
Case 1: $E = F + G$

- $h(E) = h(F + G) = h(F) + h(G)$
- We also know that $L(E) = L(F) \cup L(G)$
- $L(h(E)) = L(h(F + G)) = L(h(F) + h(G)) = L(h(F)) \cup L(h(G))$. by the definition of what $+$ means in regular expressions.
- $h(L(E)) = h(L(F) \cup L(G)) = h(L(F)) \cup h(L(G))$ because h is applied to a language by application to each of its strings individually.

By the induction hypothesis $L(h(F)) = h(L(F))$ and $L(h(G)) = h(L(G))$.

Then $L(h(F)) \cup L(h(G)) = h(L(F)) \cup h(L(G))$

Case 2: $E = F.G$

- $h(E) = h(F.G) = h(F).h(G)$
- We also know that $L(E) = L(F).L(G)$
- $L(h(E)) = L(h(F.G)) = L(h(F).h(G)) = L(h(F)).L(h(G))$.
- $h(L(E)) = h(L(F).L(G)) = h(L(F)).h(L(G))$.

By the induction hypothesis $L(h(F)) = h(L(F))$ and $L(h(G)) = h(L(G))$.

Then $L(h(F)).L(h(G)) = h(L(F)).h(L(G))$

Case 3: $E = F^*$

- $h(E) = h(F^*) = h(F)^*$
- $L(h(E)) = L(h(F^*)) = L(h(F)^*) = L(h(F))^*$.
- $h(L(E)) = h(L(F^*)) = h(L(F))^*$.

By the induction hypothesis $L(h(F)) = h(L(F))$.

Then $L(h(F))^* = h(L(F))^*$

Inverse Homomorphism

Let $h : \Sigma^* \to \Theta^*$ be a homomorphism. Let $L \subseteq \Theta^*$, and define

$$h^{-1}(L) = \{w \in \Sigma^* \mid h(w) \in L\}$$

$h^{-1}(L)$ (h inverse of L) is the set of strings w in Σ^* such that $h(w)$ is in L.

Example

Let $L = L((00+1)^*)$.
Let $h : \{a, b\}^* \rightarrow \{0, 1\}^*$ be defined by

- $h(a) = 01$
- $h(b) = 10$

We claim that $h^{-1}(L)$ is the language of regular expression (**b.a**)*, i.e.,
$h^{-1}(L) = L((ba)^*)$.

We shall prove that $h(w) \in L$ iff $w = (ba)^n$.

Proof

(If) Suppose $w = (ba)^n$ for $n \geq 0$.
Note that $h(ba) = 1001$ so $h(w)$ is n repetition of 1001. Thus $h(w) = (1001)^n \in L$.

(Only-If) We assume that $h(w)$ is in L and show that w is of the form $baba \dots ba$.

There are **four** conditions under which a string is NOT of the form, and **we shall show that if any of them hold then $h(w)$ is not in L.**

That is, we prove the contrapositive of the statement we set out to prove.

Proof (cont.)

So, let $h(w) \in L$ and suppose $w \notin L((\textbf{a.b})^*)$. There are 4 cases to consider:

1. w begins with a. Then $h(w)$ begins with 01 and $\notin L((00+1)^*)$, since it has an isolated 0.
2. w ends in b. Then $h(w)$ ends in 10 and $\notin L((00+1)^*)$.
3. w has two consecutive a, i.e., $w = xaay$. Then $h(w) = z0101v$ and $\notin L((00+1)^*)$.
4. w has two consecutive b, i.e., $w = xaay$. Then $h(w) = z1010v$ and $\notin L((00+1)^*)$.

Thus, whenever one of the above cases hole, $h(w)$ is not in L. However, unless at least one of items (1) through (4) hold, then w is of the form $baba \ldots ba$.

To see why, assume none of (1) through (4) hold.
Then (1) tells us w must begin with b and (2) tells us w must end with a.
Statements (3) and (4) tell us that a and b must alternate in w. Thus the logical OR of (1) through (4) is equivalent to the statement w **is not of the form** $baba \ldots ba$.

We have proved that the OR of (1) through (4) is not in L. The statement is the contrapositive of the statement wanted.

Closure of inverse homomorphism

If h is a homomorphism from alphabet Σ to alphabet Θ, and L is a regular language over alphabet Θ, then $h^{-1}(L)$ is also a regular language.

Proof: Let L be the language of the DFA

$$A = (Q, \Theta, \delta, q_0, F)$$

We construct from A and h a DFA for $h^{-1}(L)$. This automaton uses the states of A but translates the input symbols according to h before deciding on the next state. Formally,

$$B = (Q, \Sigma, \gamma, q_0, F)$$

where transition function γ is constructed by the rule

$$\gamma(q, a) = \hat{\delta}(q, h(a))$$

That is **the transition B makes on input a is the result of the sequence of transitions that A makes on string of symbols $h(a)$**.

By an induction on $|w|$ it is possible to show that

$$\hat{\gamma}(q_0, w) = \hat{\delta}(q_0, h(w)))$$

.

Since the accepting states of A and B are the same, B accepts w iff A accepts $h(w)$.

Put another way, B accepts exactly thoses strings w that are in $h^{-1}(L)$.