Regular Languages: Properties Module - 1

Dr. Mousumi Dutt

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.

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.

Algebraic Laws for Regular Expressions

- 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 & Annihilators

- 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 Law

- 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 expression

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.

Closure under intersection

- 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.

Closure under intersection

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 $a_1 a_2 \ldots a_n$ is the string written backwards $a_n a_{n-1} \ldots a_1$.
- 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

Closure under reversal

1. $E = F + G$. Then $E^R = F^R + G^R$.
   Justification: The reversal of the union of two languages is obtained by computi
   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 $w_1 w_2 \ldots w_n$, where each w_i is in $L(F)$. But $w^R = w_n^R \ldots . w_2^R . w_1^R$ 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 $w_1 w_2 \ldots w_n$ 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 $h : \Sigma^* \rightarrow \Theta^*$, 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 $h : \{0,1\}^* \rightarrow \{a,b\}^*$ be defined by $h(0) = ab$, and $h(1) = \epsilon$. Now

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

Homomorphisms

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 $L(h(E)) = h(L(E))$.

Homomorphisms

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

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

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

Homomorphisms

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

Homomorphisms

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 Homomorphisms

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.

Inverse Homomorphisms: Example

Let $L = L((00 + 1)^*)$.
Let $h : \{a, b\}^* \to \{0, 1\}^*$ 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 $w = (ba)^n$.

Inverse Homomorphisms: 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 $baba \ldots ba$.

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.

Inverse Homomorphisms: Proof

So, let $h(w) \in L$ and suppose $w \notin L((\mathbf{a.b})^*)$. 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 Homomorphisms

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 $h(a)$.**

Closure of Inverse Homomorphisms

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 $h^{-1}(L)$.

Proving a Language Not to be Regular

- Regular languages has at least four different descriptions:
  1. Languages accepted by DFA
  2. Languages accepted by NFA
  3. Languages accepted by ϵ-NFA
  4. Languages defined by regular expressions
- **NOT every language is a regular language**
- Powerful technique for showing certain languages not to be regular: Pumping Lemma

Theorem: Pumping Lemma for Regular Languages

Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $| w | \geq n$, we can break w into three strings, $w = xyz$, such that:

1. $y \neq \epsilon$
2. $| xy | \leq n$
3. For all $k \geq 0$, the string $xy^k z$ is also in L

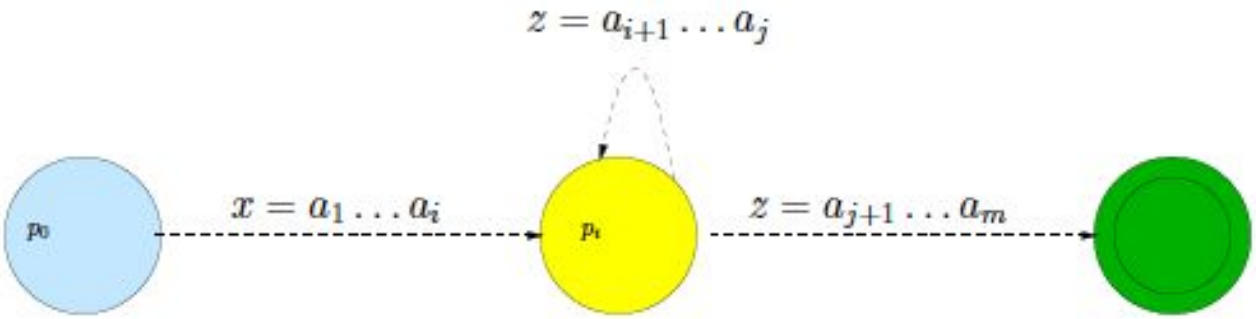
That is, we can always find a nonempty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, or deleting it (case $k = 0$, keeps the resulting string in the language L $\square$

Proof: Pumping Lemma for Regular Languages

- Suppose L is regular and recognized by a DFA A. Suppose A has n states.
- Let w be a string of length n or more, *i.e.*, $w = a_1 a_2 \ldots a_m$ where $m \geq n$ and each a_i is an input symbol.
- For $i = (0, 1, \ldots n)$ define state p_i to be $\hat{\delta}(q_0, a_1 a_2 \ldots a_i)$, where δ is the transition function of A and q_0 is the start state of A.
- **That is, p_i is the state A is after reading the first i symbols of w.** Note that $p_0 = q_0$.
- It is not possible for the $n + 1$ different p_i (for $i = (0, 1, \ldots n)$). to be distinct, since there are only n different states (Pigeonhole principle).
- Thus we can find two different integers i and j (with $0 \leq i < j \leq n$) such that $p_i = p_j$.

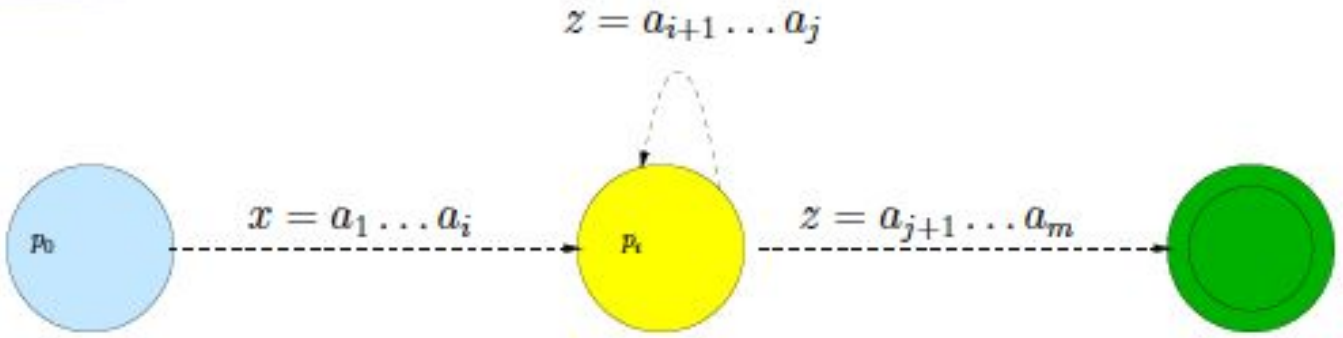
Proof: Pumping Lemma for Regular Languages

- Now we can break $w = xyz$ as follows:
  1. $x = a_1 a_2 \ldots a_i$
  2. $y = a_{i+1} a_{i+2} \ldots a_j$
  3. $z = a_{j+1} a_{j+2} \ldots a_m$



- That is, x takes us to p_i once; y takes us from p_i back to p_i (since p_i is also p_j), and z is the balance of w.
- Note that x and z may be empty.
- However y cannot be empty (i is strictly less than j).

Proof: Pumping Lemma for Regular Languages



- Now consider when A receives input xy^kz for $k \geq 0$:
  - If $k = 0$ then the automaton goes from the start start state to p_i on input x. Since p_i is also p_j, it must be that A goes from p_i to the accepting state on input z. Thus A accepts xz.
  - If $k > 0$ then A goes from q_0 to p_i on input x, circles from p_i to p_i, k times on input y^k, and then goes to the accepting state on input z. Thus for any $k \geq 0$, xy^kz is also accepted by A.

The Pumping Lemma as an Adversarial Game

Theorems whose statement involves several alternatives of *for all* and *there exists* quantifiers can be though of as a game between two players. The pumping lemma is an important example of this type of theorem.

We can see the application of the pumping lemma as a game, in which

1. **Player 1** picks the language L to be proved non regular.
2. **Player 2** picks n, but does not reveal to player 1 what n is; player 1 must devise a play for all possible n.
3. **Player 1** picks w, which may depend on n and which must be of length at least n.
4. **Player 2** divides w into x, y and z, obeying the constraints that are stipulated in the pumping lemma; $y \neq \epsilon$ and $| x.y | \leq n$. Again, player 2 does not have to tell player 1 what x, y and z are, although they must obey the constraints.
5. **Player 1** wins by picking k, which may be a function of n, x, y and z, such that $x.y^k.z$ is NOT in L.

Example

Let $L = \{0^k 1^k : k \in \mathbb{N}\}$. We prove that L is not regular.

[step 1]
By way of contradiction, suppose L is regular.

[step 2]
Let n be as in the Pumping Lemma.

[step 3]
Let $x = 0^n 1^n$.
Then $x \in L$ [definition of L]
and $|x| = 2n \geq n$.

[step 4]
By Pumping Lemma, there are strings u, v, w such that

(i) $x = uvw$,
(ii) $v \neq \epsilon$,
(iii) $|uv| \leq n$,
(iv) $uv^k w \in L$ for all $k \in \mathbb{N}$.

Let y be the prefix of x with length n. I.e., y is the first n symbols of x.
By our choice of x, $y = 0^n$.
By (i) and (iii), $uv = 0^j$ for some $j \in \mathbb{N}$ with $0 \leq j \leq n$.
Combining with (ii), $v = 0^j$ for some $j \in \mathbb{N}$ with $0 < j \leq n$.

By (iv), $uv^2 w \in L$. (#)
Aside: We are picking $k = 2$. Indeed, any $k \neq 1$ will do here.

However, $uv^2 w = uvvw$
$= 0^{n+j} 1^n$
$\notin L$, [definition of L; since $j > 0$, $n + j \neq n$]
which contradicts (#).

Therefore L is not regular. □

Example

Let $L = \{(10)^p 1^q : p, q \in \mathbb{N},\ p \geq q\}$. We prove that L is not regular.

[step 1]
By way of contradiction, suppose L is regular.

[step 2]
Let n be as in the Pumping Lemma.

[step 3]
Let $x = (10)^n 1^n$.
Then $x \in L$ [definition of L]
and $|x| = 3n \geq n$.

[step 4]
By Pumping Lemma, there are strings u, v, w such that

(i) $x = uvw$,
(ii) $v \neq \epsilon$,
(iii) $|uv| \leq n$,
(iv) $uv^k w \in L$ for all $k \in \mathbb{N}$.

Let y be the prefix of x with length n.
By our choice of x, $y = (10)^{\frac{n}{2}}$ if n is even, and $y = (10)^{\frac{n-1}{2}}1$ if n is odd.
By (i) and (iii), uv is a prefix of y, and
$uv = (10)^j$ for some $j \in \mathbb{N}$ with $0 \leq j \leq \frac{n}{2}$, or
$uv = (10)^j 1$ for some $j \in \mathbb{N}$ with $0 \leq j < \frac{n}{2}$.
Combining with (ii) — depending on whether $|uv|$ is even or odd,
v is some nonempty substring of $(10)^j$ for some j where $0 \leq j \leq \frac{n}{2}$, or
v is some nonempty substring of $(10)^j 1$ for some j where $0 \leq j < \frac{n}{2}$.

Example

There are 3 cases to consider:

(a) v starts with 0 and ends with 0.
(b) v starts with 1 and ends with 1.
(c) v starts and ends with different symbols.

For case (a), $uv^0w = uw$ contains 110 as a substring.
Thus $uv^0w \notin L$, [110 is not a substring of any string in L] which contradicts (iv).

Similarly for case (b), uv^0w contains 00 as a substring. *[details left to reader]*

For case (c), $v = (10)^i$ or $v = (01)^i$, where $0 < i$.
So $|v| = 2i$.
Thus $uv^0w = uw = (10)^{n-i}1^n \notin L$, [definition of L; $n - i < n$]
which contradicts (iv).

We reach a contradiction in all cases.

Therefore L is not regular. $\square$

Example

Let C = { w | w has an equal number of 0's and 1's }. Use the pumping lemma to show that this language is not regular.

Assume that C is regular and let p be the pumping length selected by the adversary. Choose s to be the string $(01)^p$. This is clearly of length at least p and is also in the language.

Our adversary splits the string into an x, y, and z. Let's say the adversary splits it into $x=\varepsilon$, $y=01$, and $z=(01)^{p-1}$. Can we find a value k such that xy^kz is not in C? If k=0 then we just get the string xz, which is the string z. z has an equal number of 0's and 1's so it is in C. If k=1 then we get the string xyz, which is also in C. If we pick k=2 then we get the string xyyz, which is also in C. No matter what value of k we pick, each resulting string is still in the language. This means that we didn't pick the right string (or that the language actually is regular).

Example

So let's try again by picking $s=0^p1^p$. This string is also clearly of length at least p and is also in the language. The adversary breaks up the string s into xyz. But we know that since $|xy| \leq p$, then x and y must consist entirely of 0's. Based on condition 3, if we let k=2, then xyyz will have more 0's then there are 1's so this is not in C. Similarly, if we let k=0, then xz will have fewer 0's then 1's so this is also not in C (we can pick k equal to any value except 1).

But what if the adversary picks $x=\varepsilon$ and $z=\varepsilon$? That is, the string y contains the entire string. Then it would seem that xy^kz will still be in C, since y will contain an equal number of 0's and 1's. But since $|xy|$ must be $\leq p$ this selection is not possible for y to equal 0^p1^p, given $s=0^p1^p$.

Example

Let $D = \{ww \mid w \in \{0,1\}^*\}$. In other words, pick w equal to any finite sequence of 0's and 1's. Then allow only those strings that have this word in it back to back. Show that D is non-regular using the pumping lemma.

Assume that D is regular and let p be the pumping length selected by the adversary. Choose s to be the string 0^p0^p. This is clearly of length at least p and is also in the language.

Our adversary splits the string into an x, y, and z. Let's say the adversary splits it into $x=\varepsilon$, $y=00$, and $z=0^{2p-2}$. Can we find a value k such that xy^kz is not in D? The answer is

Example

no, for any value of k the resulting string is still in D. Obviously this was not a good choice of a string s. Let's pick another one.

Choose s to be the string 0^p10^p1. This is clearly a member of D and has length of at least p. Our adversary splits the string into an x,y, and z. Once again, since $|xy| \leq p$, we must have the case that x and y consist entirely of zeros. If we pump y by letting k=2, then we now have more zeros in the first half of the string then in the second half, so the resulting string is no longer in D. Therefore, the language is not regular.

Example

Let F = { 1^{n^2} | n >=0 }. That is, each string consists of 1's and is of length that is a perfect square:

ε, 1, 1111, 111111111 , 1111111111111111, etc. (0, 1, 4, 9, 16, 25, 36, …)

Notice that the gap between the length of the string grows in the sequence. Large members of this sequence cannot be near each other. If we subtract off the difference in length between successive elements, we get 1, 3, 5, 7, 9, 11, 13, etc. For position i where i >0, we get the difference from position i and i-1 as 2*i -1.

Assume that F is regular and let p be the pumping length selected by the adversary. Let $m = p^2$. Choose s to be the string 1^m. This string is clearly in F and is at least of length p.

Example

The adversary picks some strings x,y, and z. Now consider the two strings xy^kz and $xy^{k+1}z$. Both of these strings must be in F. These two strings differ only by a single repetition of y, or |y| which we know must be $\leq p$. However, note that the length of the strings accepted by the language grows in length by 2i-1, not by some fixed amount |y|.

Each time we pump the string we increase the value of i, so we can always find a value of i to make 2i-1 larger than |y|. Consequently, we can always pick a large enough k such that assuming xy^kz is in the language, $xy^{k+1}z$ cannot be in the language because the length differential will be too small to equal to 2i-1.