Properties of regular languages

27/11/2007

Closure properties

The closure properties tell us which operations let us stay within the class of regular languages, assuming we start from regular languages

Theorem: (Closure under regular operations)

If L_1, L_2 are regular, then so are: $L_1 \cup L_2$

$L_1 \circ L_2$

L_1^*

Proof: since L_1, L_2 are regular, there are R.E.s E_1, E_2 s.t.

$\mathcal{L}(E_1) = L_1$

$\mathcal{L}(E_2) = L_2$

Then: $L_1 \cup L_2 = \mathcal{L}(E_1) \cup \mathcal{L}(E_2) = \mathcal{L}(E_1 + E_2)$ $\Rightarrow$ is regular

$L_1 \circ L_2 = \mathcal{L}(E_1) \circ \mathcal{L}(E_2) = \mathcal{L}(E_1 \circ E_2)$ $\Rightarrow$ is regular

$L_1^* = (\mathcal{L}(E_1))^* = \mathcal{L}(E_1^*)$ $\Rightarrow$ is regular

q.e.d.

Closure under boolean operations:

If L_1 over Σ_1 and L_2 over Σ_2 are regular, then so are

- $L_1 \cup L_2$ (union)
- $\Sigma^* - L_1$ (complement)
- $L_1 \cap L_2$ (intersection)

Note: to define the complement $\overline{L}$ of a language L, we need to specify the alphabet Σ of L: $\overline{L} = \Sigma^* - L$

We may omit to specify Σ when it is clear from the context.

Theorem: (closure under complementation)

If L over Σ is regular, then so is $\overline{L} = \Sigma^* - L$.

Proof:

Since L is regular, there is a DFA

$A_L = (Q, \Sigma, \delta, q_0, F)$ s.t. $\mathcal{L}(A_L) = L$

Construct $\overline{A}_L = (Q, \Sigma, \delta, q_0, Q - F)$

Then $w \in \mathcal{L}(\overline{A}_L)$ iff $\hat{\delta}(q_0, w) \in Q - F$

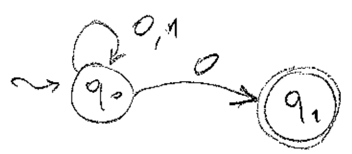
iff $\hat{\delta}(q_0, w) \notin F$

iff $w \notin \mathcal{L}(A_L)$

Hence $\mathcal{L}(\overline{A}_L) = \overline{\mathcal{L}(A_L)} = \overline{L}$, and $\overline{L}$ is regular q.e.d.

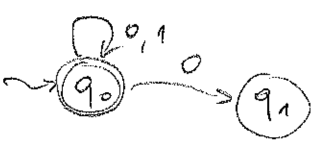
Note: In order to obtain the complement by complementing the set of final states, the automaton has to be deterministic

Example: let A_0 be the NFA



$\mathcal{L}(A_0) = \{w \mid w$ ends with a $0\}$

If we take A_0' with



then $\mathcal{L}(A_0') = \Sigma^* \neq \overline{\mathcal{L}(A_0)}$

Hence, in general, given an NFA A_N, to obtain an automaton for $\overline{\mathcal{L}(A_N)}$ we first have to determinize A_N (e.g., by applying the subset construction). $\Rightarrow$ Exponential blowup

Exercise E4.1 By referring to examples we have seen, prove that in general we cannot do better to compute a DFA for the complement of the language accepted by an NFA.

Theorem (closure under intersection) 27/10/2004 (4.3)

If L_1, L_2 are regular, then so is $L_1 \cap L_2$

Proof: we simply use De Morgan's law

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

and exploit closure under $\cup$ and $^-$.

Note: this proof is constructive, i.e. given e.g. NFA's for L_1 and L_2, it tells us how to construct an NFA for $L_1 \cap L_2$.

What is the cost of this construction? Exponential

In fact, there is a direct construction that computes, given two NFA's A_1, A_2, an NFA $A_{1\cap 2}$ for $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.
If A_1 and A_2 have respectively n_1 and n_2 states, then $A_{1\cap 2}$ has $n_1 \cdot n_2$ states. ($A_{1\cap 2}$ is called product automaton)
See book for details [Exercise]

EXERCISE

Closure under reversal.

Definition:

reversal of a string:

- $\varepsilon^R = \varepsilon$
- if $w = a_1 \cdots a_n$ then $w^R = (a_1 \cdots a_n)^R = a_n \cdots a_1$

reversal of a language: $L^R = \{w^R \mid w \in L\}$

Theorem (closure under reversal)

If L is regular, then so is L^R

Proof: we extend reversal to R.E., inductively

base: $\varepsilon^R = \varepsilon$

$\emptyset^R = \emptyset$

$a^R = a$ for $a \in \Sigma$

induction: $(E_1 + E_2)^R = E_1^R + E_2^R$

$(E_1 \cdot E_2)^R = E_2^R \cdot E_1^R$

$(E_1^*)^R = (E_1^R)^*$

We proof by structural induction that $\mathcal{L}(E^R) = (\mathcal{L}(E))^R$

base: clear

induction:

$\mathcal{L}((E_1 + E_2)^R) = \ldots$ [Def. of reversal for R.E.]

$= \mathcal{L}(E_1^R + E_2^R) =$ [Semantics of +]

$= \mathcal{L}(E_1^R) \cup \mathcal{L}(E_2^R) =$ [I.H.]

$= (\mathcal{L}(E_1))^R \cup (\mathcal{L}(E_2))^R =$

$= \{w^R \mid w \in \mathcal{L}(E_1)\} \cup \{w^R \mid w \in \mathcal{L}(E_2)\} =$

$= \{w^R \mid w \in \mathcal{L}(E_1) \cup \mathcal{L}(E_2)\} =$

$= (\mathcal{L}(E_1) \cup \mathcal{L}(E_2))^R =$ [Semantics of +]

$= (\mathcal{L}(E_1 + E_2))^R$

Other cases: $\boxed{\text{exercise}}$

Example: $E = a b c + b c^* a$

$E^R = c b a + a c^* b$

EXERCISE

Proving languages not to be regular

24/10/2005
23/10/2006

Consider: $L_{alt} = \{ w \mid$ has alternating 0's and 1's $\}$

$L_{eq} = \{ w \mid$ has an equal number of 0's and 1's $\}$

- Claim: L_{alt} is regular

  Proof: easy $E_{alt} = (\varepsilon + 0)(1 \cdot 0)^* \cdot (\varepsilon + 1)$ is such that $\mathcal{L}(E_{alt}) = L_{alt}$

- Claim: L_{eq} is not regular

  How can we prove this?

  Intuition: · DFA with n states can count up to n.

  · to decide whether $w \in L_{eq}$ we need unbounded counting (since w may be arbitrarily long)

Pumping Lemma:

<u>For all</u> regular languages $L \subseteq \Sigma^*$

<u>there exists</u> n (which depends on L) such that

<u>for all</u> $w \in L$ with $|w| \geq n$

<u>there exists</u> a decomposition $w = xyz$ of w s.t.

1) $|y| \geq 1$ (i.e., $y \neq \varepsilon$)
2) $|x \cdot y| \leq n$
3) <u>for all</u> $k \geq 0$, $x y^k z \in L$.

Intuitively, for every $w \in L$, we can find a substring y "near" the beginning of w that can be "pumped", while still obtaining words in L.

OPTIONAL

Proof:

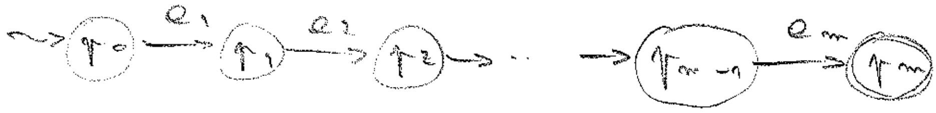
Given regular language L, let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $\mathcal{L}(A) = L$.

We take $n = |Q|$.

Consider any $w = a_1 a_2 \cdots a_m \in L$ with $m = |w| \geq n$.

Since $w \in \mathcal{L}(A)$, we have that $\hat{\delta}(q_0, w) \in F$.

Define $p_i = \hat{\delta}(q_0, a_1 a_2 \cdots a_i)$ $\forall i \in \{1, \dots, m\}$ and $p_0 = q_0$



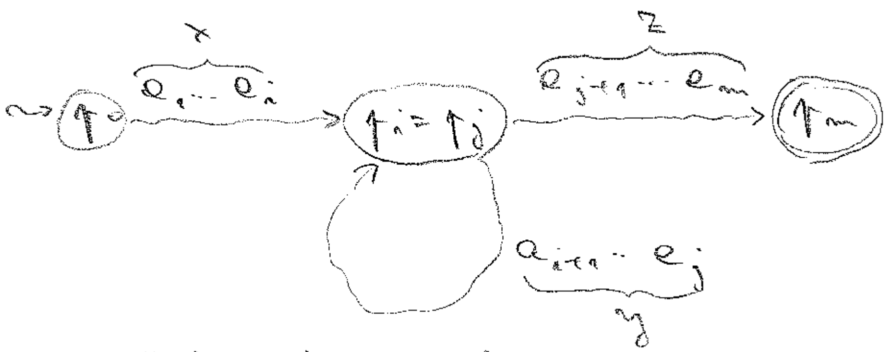
Since $m \geq n$,

- each p_i, $0 \leq i \leq n$ belongs to Q, and
- $|Q| = n$

by the pigeon-hole principle, $p_0, p_1, \dots, p_n$ are not all distinct

Let i, j, with $0 \leq i < j \leq n$ be the least indices such that $p_i = p_j$.

Hence, to accept w, the DFA goes through a cycle:



$\hat{\delta}(p_0, x) = p_i$

$\hat{\delta}(p_i, y) = p_i$

$\hat{\delta}(p_i, z) = p_m$

Observe: $|y| = j - i \geq 1$ (since $i < j$)

- $|xy| = j \leq n$

$\hat{\delta}(q_0, xy^k z) = \hat{\delta}(\hat{\delta}(q_0, x), y^k z) = \hat{\delta}(p_i, y^k z) = \hat{\delta}(\hat{\delta}(p_i, y), y^{k-1} z)$

$= \hat{\delta}(p_i, y^{k-1} z) = \dots = \hat{\delta}(p_i, z) = p_m \in F \Rightarrow xy^k z \in L$

q.e.d.

OPTIONAL END

The pumping lemma states a property of R.L. that can be used to show that a given language is <u>not</u> regular.

<u>Idea</u>: pick $w \in L$ such that we can easily show that
$x y^k z \notin L$ for some choice of k.

Difficulty: we must do so regardless of the choices for n, and the decomposition x, y, z

More precisely: to show that L is <u>not regular</u>,
we have to show that:

<u>for all</u> n

<u>there exists</u> a $w \in L$ with $|w| \geq n$ such that

<u>for all</u> decompositions $w = xyz$ of w ...

with $|y| \leq 1$

$|x \cdot y| \leq n$

<u>there exists</u> $k \geq 0$ s.t. $x \cdot y^k \cdot z \notin L$

We can view the alternation of $\forall$ and $\exists$ as a game between Alice and Ed:

- Ed chooses the language L he wants to show nonregular
- Alice chooses n
- Ed chooses $w \in L$ with $|w| \geq n$
- Alice chooses a decomposition $w = x \cdot y \cdot z$ with $|y| \geq 1$, $|x \cdot y| \leq n$
- Ed chooses $k \geq 0$, and he wins iff $x \cdot y^k \cdot z \notin L$.

Then L is not regular <u>iff</u> Ed has a winning strategy, i.e., he can win whatever moves Alice makes (respecting the rules).

<u>Example</u>: L_{eq} is not regular

Let's play the game and show that Ed can always win.

- Ed chooses L_{eq}
- Alice chooses some n
- Ed chooses $w = 0^n 1^n$

  note that $w \in L$ and $|w| \geq n$
- Alice chooses a decomposition $w = x \cdot y \cdot z$

  with $y \neq \varepsilon$ and $|x \cdot y| \leq n$

  note that, since $|x \cdot y| \leq n$, we have $x \cdot y = 0 \ldots 0$

  $\Rightarrow$ let $x = \underbrace{0 \cdots 0}_{a}$ $\quad y = \underbrace{0 \cdots 0}_{b \geq 1}$

  then $w = \underbrace{0^a}_{x} \cdot \underbrace{0^b}_{y} \cdot \underbrace{0^{n-a-b} \cdot 1^n}_{z}$

- Ed chooses $k = 0$

  then $x \cdot y^0 \cdot z = x z = 0^a \cdot 0^{n-a-b} \cdot 1^n = 0^{n-b} 1^n \notin L$

  and Ed wins

$\Rightarrow$ L_{eq} is not regular

$\boxed{\text{Exercise: E4.2}}$ Let $L_{prime} = \{ w \in \{0\}^* \mid |w| \text{ is prime} \}$.

Show that L_{prime} is not regular.

Notice that the converse of the Pumping Lemma does not hold.

In terms of the game between Alice and Ed

L is not regular $\overset{\Leftarrow}{\not\Rightarrow}$ Ed has a winning strategy

Example: consider $L = L_1 \circ L_2$ with L_1 regular, L_2 not regular

We have that L is not regular

but Ed does not have a winning strategy

Decision problems for regular languages

2/11/2004

Decision problem: Let P be some property of languages

input: regular language L (represented as DFA, NFA, ε-NFA, or R.E.)

output: does L have property P — yes / no

A decision algorithm decides a decision problem:

means: - correct answer
- always terminates in finite time

Emptiness: decide if a regular language L is empty

When L is given as an automaton, then L is not empty iff a final state is reachable from the initial state

This is an instance of graph reachability: recursively
- base: the initial state is reachable
- induction: if q is reachable, and $\delta(q,a)=p$ for some a, then p is reachable

For n states, this takes at most $O(n^2)$
(actually, it takes at most the number of arcs)

Exercise: Emptiness, when L is given as a R.E.

Let us compute empty(E) by structural induction on E

base: empty($\emptyset$) = true
empty(ε) = false
empty(a) = false $\forall a \in \Sigma$

induction: empty(E^*) = false
empty((E)) = empty(E)
empty($E_1 + E_2$) = empty(E_1) $\wedge$ empty(E_2)
empty($E_1 \cdot E_2$) = empty(E_1) $\vee$ empty(E_2)

$\Rightarrow$ linear in E

Membership: given $w \in \Sigma^*$ and $L \subseteq \Sigma^*$ with L regular, decide whether $w \in L$.

2/11/2005
25/10/2006

Algorithm:

- when L is given as a DFA A_D
  - simulate the run of A_D on w
  - if transition table is stored as a 2-dimensional array, each transition takes constant time

    $\Rightarrow$ test takes linear time in $|w|$

- when L is given as an NFA A_N
  - if we compute the equivalent DFA $\Rightarrow$ exponential in $|A_N|$, linear in $|w|$
  - we can also simulate directly the NFA, by computing the sets of states the NFA is in after each input symbol

    $\Rightarrow O(|w| \cdot s^2)$ where s is the number of states of A_N

    at each step at most s states
    each with at most s successors

Equality: given regular languages L_1, L_2
decide whether $L_1 = L_2$

Idea: reduce to emptiness:

consider $L = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$ (symmetric difference)

L is regular, by closure of $\cap$, $\cup$, $^-$

then $L_1 = L_2 \iff L = \emptyset$

Algorithm: 1) Compute representation for L (as DFA or R.E.)
2) Decide emptyness of L

OPTIONAL

Finiteness: given regular language L
decide whether L is finite

Let A_L be a DFA for L with n states.

Theorem: L is infinite iff $\exists w \in L$ s.t. $n \leq |w| < 2n$.

Proof: "$\Leftarrow$": Let $w \in L$ with $n \leq |w|$.
By pumping lemma, $w = x \cdot y \cdot z$ with $y \neq \varepsilon$
and $\forall k \geq 0$, $x \cdot y^k \cdot z \in L$.
Hence L is infinite

"$\Rightarrow$" Suppose L is infinite.
Then $\exists w \in L$ s.t. $|w| \geq n$ (there are only finitely many strings of length $< n$)

Let $\tilde{w}$ be the shortest string in L of length $\geq n$.

Claim: $|\tilde{w}| < 2n$

Proof by contradiction: suppose $|\tilde{w}| \geq 2n$
By pumping lemma, $\tilde{w} = x \cdot y \cdot z$ with $|x \cdot y| \leq n$
$|y| \geq 1$

and $x \cdot y^0 \cdot z = x \cdot z \in L$

We have:

1) $|x \cdot z| = |\tilde{w}| - |y| \geq 2n - n = n$

2) $|x \cdot z| < |\tilde{w}|$, since $|y| \geq 1$

This contradicts choice of $\tilde{w}$ as shortest string, which proves the claim.

Hence, we have a string $\tilde{w} \in L$ with $n \leq |\tilde{w}| < 2n$

q.e.d.

From the theorem we get an algorithm for finiteness

Algorithm: For each $w \in \Sigma^*$ with $n \leq |w| < 2n$,
test whether $w \in L$

↑ OPTIONAL END

$\boxed{\text{Exercise 4.3.3.}}$ Give an algorithm to decide whether a regular language L is universal, i.e. $L = \Sigma^*$

$\boxed{\text{Exercise 4.3.4}}$ Give an algorithm to decide whether two regular languages L_1 and L_2 have at least one string in common.

$\boxed{\text{Exercise E 4.3}}$ Give an algorithm to decide whether a regular language L_1 is contained in another regular language L_2

State minimization

Given DFA $A = (Q, \Sigma, \delta, q_0, F)$, find A' with minimum number of states s.t. $\mathcal{L}(A') = \mathcal{L}(A)$.

<u>Idea</u>: partition Q into equivalence classes and collapse equivalent states

Equivalence relation on states:

$$p \equiv q \quad \text{if for all } w \in \Sigma^* : \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

The equivalence relation induces a partition of Q

$$Q = C_1 \cup C_2 \cup \cdots \cup C_k$$

$$\text{for all } p \in C_i,\ q \in C_j : \quad p \equiv q \iff i = j$$

How do we find the partition? We discover <u>inequivalent</u> states:

$$p \not\equiv q \quad \text{if for some } w \in \Sigma^* \quad \hat{\delta}(p, w) \in F \text{ and } \hat{\delta}(q, w) \notin F$$

or viceversa.

Let $w = a_1 a_2 \cdots a_m$ (i.e. $|w| = m$)

$$p \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \longrightarrow \cdots \xrightarrow{a_{m-1}} p_{m-1} \xrightarrow{a_m} p_m$$
$$q \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \longrightarrow \cdots \xrightarrow{a_{m-1}} q_{m-1} \xrightarrow{a_m} q_m$$

one is final and the other is not

Note: $a_{i+1} \cdots a_m$ is a <u>proof</u> of length $m-i$ <u>of inequivalence</u> of p_i and q_i

Definition: $p \equiv_i q$ if for all w with $|w| \leq i$

$$\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

(intuitively, there is no inequivalence proof of lenght $\leq i$)

The following is immediate to see:

$r \not\equiv_{i+1} q$ if and only if for some $a \in \Sigma$

$\delta(r, a) \not\equiv_i \delta(q, a)$.

$\boxed{10/12/2007}$

<u>Algorithm</u> to compute $\equiv_i$ inductively on i:

step 0: partition $Q = C_1 \cup C_2$ with $C_1 = F$, $C_2 = Q - F$

justified since $r \not\equiv_0 q$ iff one is final and the other not

step $i+1$: determine $r \equiv_{i+1} q$ iff $\forall a \in \Sigma$

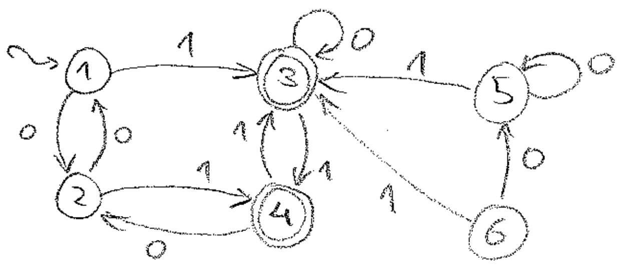
$$\delta(r, a) \equiv_i \delta(q, a)$$

compute refined partition

Algorithm terminates when the refined partition coincides with the one in the previous step (at most $|Q|$ steps)

8/11/2004

<u>Example</u>:



step 0: $C_1^0 = \{1, 2, 5, 6\}$ $C_2^0 = \{3, 4\}$

step 1: $C_1^1 = \{1, 2, 5, 6\}$ $C_2^1 = \{3\}$ $C_3^1 = \{4\}$

step 2: $C_1^2 = \{1, 5, 6\}$ $C_2^2 = \{2\}$ $C_3^2 = \{3\}$ $C_4^2 = \{4\}$

step 3: $C_1^3 = \{1\}$ $C_2^3 = \{2\}$ $C_3^3 = \{3\}$ $C_4^3 = \{4\}$ $C_5^3 = \{5, 6\}$

step 4: no change

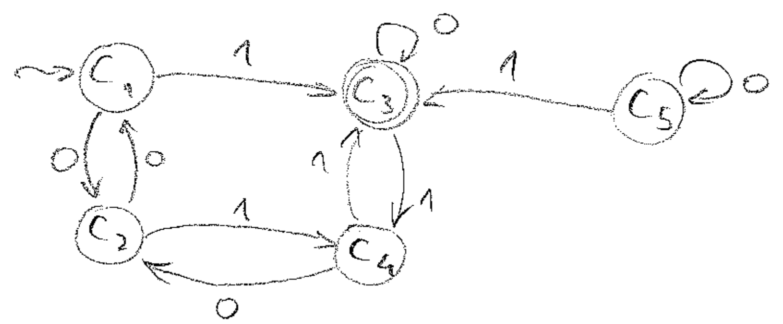
To construct A':

1) Construct partition $Q = C_1 \cup \dots \cup C_k$ of states of A
2) Construct $A' = (Q', \Sigma, \delta', q_0', F')$
   - states $Q' = \{C_1, C_2, \dots, C_k\}$
   - transitions: if $\delta(q, a) = q$ in A
     then $\delta(C[q], a) = C[q]$
     where $C[q]$ is the equivalence class of q
   - start state: $C[q_0]$
   - final states: $\{C[q_f] \mid q_f \in F\}$

We can verify that A' is a well-defined DFA.

Exercise E 4.4

Example:



Note that C_5 is not reachable from the start state and must be removed.

We could show that the DFA constructed in this way is the smallest possible for a given language.

Myhill - Nerode Theorem:

Given $L \subseteq \Sigma^*$, consider the equivalence relation R_L on Σ^* defined as follows: $x R_L y \Leftrightarrow \forall z \in \Sigma^*: xz \in L \Leftrightarrow yz \in L$.
Then L is regular iff R_L induces a finite number of equivalence classes.