COMS3261:
Computer Science Theory

Fall 2013

Mihalis Yannakakis

Lecture 8, 9/30/13

Algorithmic problems for regular languages

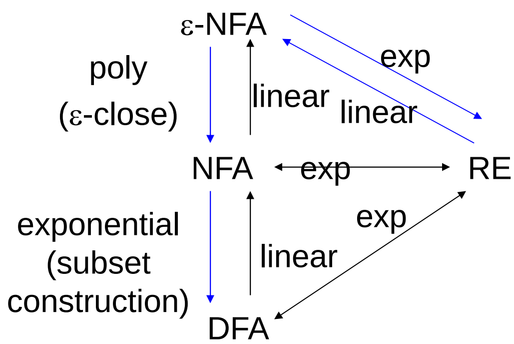
- Converting between representations (DFA,NFA, ε-NFA, RE) Given one representation, construct another
- Emptiness problem: Given representation, is $L = \emptyset$?
- Membership problem: Given representation of L, string w, is $w \in L$?
- Equivalence: Given two descriptions (of the same or different types), do they define the same language?
- Minimization: Given a description construct a minimum equivalent one (with some notion of size, eg. #states for FA)

•Algorithms and complexity may depend on given description.

•Can convert between different representations but at a cost.

Conversion between representations

- Already covered the algorithms



Membership

- Input: Representation E of language L, string w
- Question: w $\in$ L ?
- Algorithm:
- E = DFA: Simulate E on w $\rightarrow$ linear time in |E|+|w|
- E=NFA: Simulate E on w$\rightarrow$ polynomial time in |E|+|w|
  If E has s states, |w|=n, then time $O(ns^2)$
- E=ε-NFA: Simulate E on w$\rightarrow$ polynomial time in |E|+|w|
  If E has s states, |w|=n, then time $O(ns^3)$
- E = RE: Convert to ε-NFA and simulate $\rightarrow$ $O(n|E|^3)$

Emptiness

- Input =DFA or NFA or ε-NFA A: $L(A) \neq \emptyset$ iff some state in F reachable from start state.

Apply a Reachability algorithm (e.g. BFS or DFS) from q_0 .

Time = linear in size of A (number of states and transitions)

- Input = RE

- Convert to ε-NFA and apply the reachability algorithm
  Time = $O(|E|)$
- Alternatively: can check inductively each subexpression
  Basis: $E = \emptyset$ or ε or a in Σ : trivial
  Induction: $E = G1 + G2$: $E \equiv \emptyset$ iff both $G1, G2 \equiv \emptyset$
  $E = G1.G2$: $E \equiv \emptyset$ iff at least one of $G1, G2 \equiv \emptyset$
  $E = G^*$: Then $L(E)$ is not $\emptyset$ (contains at least ε)

Universality

- Input: Representation E of a language L over alphabet Σ
- Question: Is $L = \Sigma^*$?

- Use complementation to reduce to emptiness
- For DFA complementation easy $\rightarrow$ same time complexity
- For NFA, RE, complementation is exponential.
  Translate to a DFA, complement, and apply emptiness algorithm. The translation to DFA blows up the size exponentially in general.

- No better algorithm is known, and it is believed that there does not exist any algorithm to test universality for NFA or RE that runs in polynomial time (they belong to a class of intractable problems, called PSPACE-complete)

Comparing regular languages

- $L \cap M = \emptyset$?
  - If L, M given by DFA or NFA, apply product construction and test for emptiness $\rightarrow$ quadratic time
  - If given by RE, translate first to ε-NFA, eliminate ε transitions and apply the NFA algorithm.

- Application: Verification (model checking)

  L = system executions (given eg. by a NFA)

  M = set of bad (incorrect) executions

  $L \cap M = \emptyset \quad \Leftrightarrow$ all executions are good

  - look up model checking on web (eg. in wikipedia)

Containment

- $L \subseteq M$?

  For example L = executions of system

  M = set of good (correct) executions

- $L \subseteq M \Leftrightarrow L \cap M^c = \emptyset$

  If M given by DFA, then can complement and test in polynomial (quadratic) time.

  Otherwise (M given by NFA or RE), then exponential blowup in complementation

- Universality problem is a special case: $M = \Sigma^* \Leftrightarrow \Sigma^* \subseteq M$

  hard for NFA, RE

Equivalence

- L = M ?
- Equivalent to two containments: L $\subseteq$ M and M $\subseteq$ L
- If L,M given by DFA then polynomial time
- Otherwise (NFA or RE) convert to DFA and test the DFA

$\rightarrow$ exponential blowup. Unavoidable (M=Σ^* ? a special case)

DFA: Can do directly

Method 1: Given DFA A1, A2 for L, M,

apply product construction B = A1 $\times$ A2

L = M $\Leftrightarrow$ $\forall$ reachable state (p1,p2) of B, either both p1,p2 are accepting states of A1, A2 or both rejecting

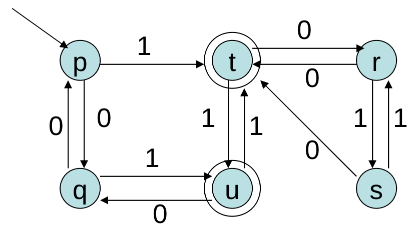
Method 2: State Equivalence algorithm

State Equivalence (for DFA)

- Two states p, q are distinguishable if $\exists$ string w such that one of δ-hat(p,w), δ-hat(q,w) is accepting and the other not.
- p,q are equivalent, p$\equiv$q, iff they are not distinguishable, i.e. for all w, either both δ-hat(p,w), δ-hat(q,w) are accepting or both rejecting.

  - Note: includes case w=ε, so all accepting states are distinguishable from all rejecting
- Equivalence relation:
  - p $\equiv$ p for all p (reflexive)
  - p $\equiv$ q $\Rightarrow$ q $\equiv$ p (symmetric)
  - p $\equiv$ q and q $\equiv$ r $\Rightarrow$ p $\equiv$ r (transitive)
- Partition of Q into equivalence classes: Two states are equivalent iff they are in the same class

Example



- p, r are distinguishable:

  on input 1 p goes to t $\in$F , r goes to s $\notin$F

- p, q are distinguishable:

  on input 101 p goes to s $\notin$F, q goes to u $\in$F

State Equivalence/Partition Algorithm

(Different than table filling algorithm in book)

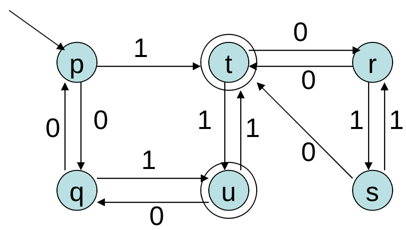
Initialize: partition { F , Q-F}

Loop: while $\exists$ block (class) B in current partition and input symbol a in S such that δ(B,a) = { δ(p,a) | p $\in$B } contains states from two or more blocks then split B into subblocks, where each subblock contains those states p of B such that δ(p,a) is in the same block

Return final partition

Final partition π* has the property that every block B is stable: $\forall$a$\in\Sigma$, δ(B,a) $\subseteq$ B' for some block B'

Example

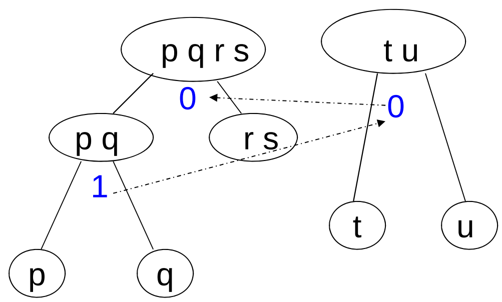


{p,q,r,s} {t,u}

{p,q} {r,s} {t,u}

{p,q} {r,s} {t} {u}

{p} {q} {r,s} {t}, {u}

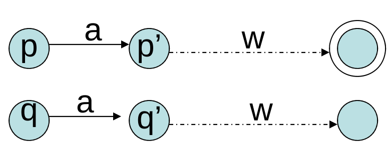


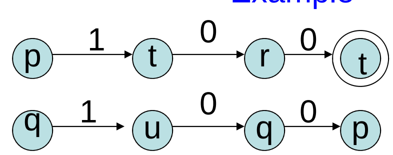
Correctness

- One direction: By induction on step of the algorithm
- Induction hypothesis: If two states p, q are in different blocks, then distinguishable (and can construct inductively string w that distinguishes them)
- Basis: Initialization. string = ε
- Induction step: We split B because of a symbol a

  If states p,q of B assigned to different subblocks, then p'=δ(p,a) and q'=δ(q,a) were in different blocks $\Rightarrow \exists$ string w that distinguishes p',q' $\Rightarrow$ aw distinguishes p,q

Example





Correctness ctd.

- Converse: If two states p,q are distinguishable then they end up in different blocks.
- Proof: By induction on the length of the shortest string w that distinguishes the states.
- Basis: length =0 $\Rightarrow$ w=ε $\Rightarrow$ one of p,q is in F, the other not
- Induction step: w=ax, where a$\in\Sigma$, x$\in\Sigma^*$

  Let p'=δ(p,a) and q'=δ(q,a). Then p',q' distinguished by x (shorter string) $\Rightarrow$ in different blocks of final partition $\Rightarrow$ p,q cannot be in the same block B because a can split B

Complexity Analysis

- Every iteration splits at least one block
- Started with 2 blocks, end with at most n=#states blocks

  $\Rightarrow$ at most n-2 iterations
- Each iteration: at worst have to examine each block B, each symbol a , and check the next states of the states of B $\rightarrow$ time O(dn) where d=|Σ|, n=|Q|
- Total time $O(dn^2)$
- Improvement (Hopcroft): O(d n logn)
- Corollary of analysis: If two states can be distinguished, then they can be distinguished by a string of length < n

Application1: Equivalence of DFA

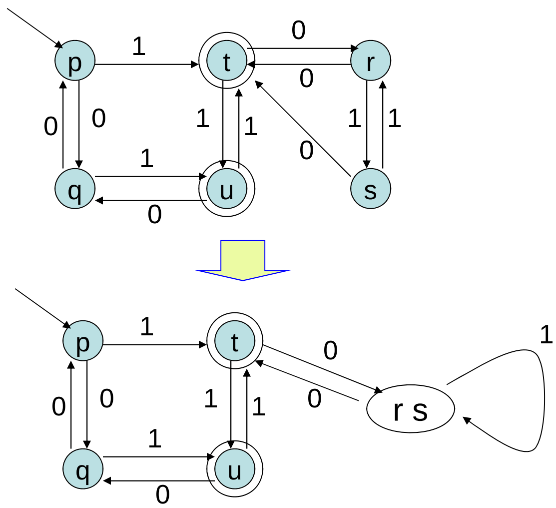
- Input: DFA A1, A2
- Consider A = disjoint union of A1, A2
  (two initial states but ignore it, does not matter)
- Compute the state equivalence partition
- L(A1) = L(A2) iff the two start states are equivalent

Application 2: Minimization of DFA

- Reduced DFA: no two states equivalent (i.e. state equivalence partition has all blocks singleton) and all states reachable from start state
- Algorithm: Given any DFA A, construct A_m as follows
- Delete unreachable states
- Compute state equivalence partition
- Q_m = ‘reachable’ blocks of partition of A
- start state = block with start state of A
- accepting states = blocks with accepting states of A
- transition function well-defined: $\delta(B,a)$ = block B’ that contains $\delta(p,a)$ for any (all) p in B

- The DFA A_m is reduced: for any two blocks B_1, B_2, if p,q are states of A in B_1,B_2 then $\exists$w that distinguishes them $\Rightarrow$ w distinguishes the blocks B_1, B_2

Example



Isomorphism of Reduced DFA

- If M, M' are reduced DFA for same language L then every state of M has an equivalent state in M' and vice-versa
- Proof: initial states q_0, q'_0 equivalent because L(M)=L(M')
- For any other state p of M, take a path from q_0 to p (exists because p reachable), let w be the label of the path,

  i.e. δ-hat(q_0,w)=p, and let p'= δ-hat(q'_0,w)

  If p,p' distinguishable, say by some string x,

  i.e. δ-hat(p,x) $\in$F and δ-hat(p',x) $\notin$F' (or vice-versa) then wx $\in$ L(M) and wx $\notin$L(M') (or vice-versa)

  Therefore p$\equiv$p' .
- Since no two states of M are equivalent, and same for M', $\equiv$ is a (1-1 onto) bijection between the states of M,M'
- $\Rightarrow$ M, M' have the same # of states

  and the bijection $\equiv$ preserves the transitions

Properties of Reduced DFA

- Every DFA that is equivalent to a reduced DFA A has at least one state equivalent to each state of A

  $\Rightarrow$ Any reduced DFA has the minimum number of states among all DFA that accept the same language

- All reduced DFA for a regular language are isomorphic = same except for the names of the states $\Rightarrow$

  Can say:
- THE minimum DFA for a language: unique (up to the names of the states)

Min DFA and the Myhill-Nerode theorem

- Recall: Two strings x,y are called distinguishable with respect to L if $\exists$ string z such that one of xz, yz is in L and the other is not; otherwise x,y are called equivalent.
- Given DFA A for language L, definitions $\Rightarrow$ two strings x,y are equivalent iff the states δ^(q_0,x), δ^(q_0,y) are equivalent
- Partition of strings into equivalence classes $\leftrightarrow$ partition of reachable states of A into equivalence classes $\leftrightarrow$ states of reduced DFA
- Index of language L (= # equivalence classes of strings)

  = # states of minimum DFA for L

NFA don't have unique minimal NFA

- Example: $\Sigma^*0\Sigma^*$, where $\Sigma=\{0,1\}$