CS81 Notes Spring 2012 REGULAR LANGUAGES[1]

I. We fix a finite alphabet Σ and let $\Sigma^\star$ denote the set of all finite sequences (strings) whose elements are from Σ. We use ϵ to denote the *empty string*. A (formal) **language** L is a subset of $\Sigma^\star$.

II. Things we can do (function) or ask about (predicate) strings: (a) x^R is the *reverse* of x; (b) x^n is x *repeated* n times; (c) x is a *prefix* of y iff $y = xz$, for some string z; (d) x is a *suffix* of y iff $y = zx$, for some string z; (e) x is a *substring* of y iff $y = uxv$, for some strings u and v; and many others.

III. Examples of languages:

(a) $L = \emptyset$.
(b) $L = \{\epsilon\}$.
(c) $L = \{a, b\}^\star$.
(d) $L = \{w \in \{a, b\}^\star : w \text{ has all } a\text{'s before all } b\text{'s}\}$.
(e) $L = \{w \in \{a, b\}^\star : w \text{ ends with } ab\}$.
(f) $L = \{w \in \{a, b\}^\star : w \text{ contains } ab\}$.
(g) $L = \{w \in \{a, b\}^\star : w \text{ has an odd number of } b\text{'s}\}$.
(h) $L = \{w \in \{a, b\}^\star : w \text{ is a palindrome}\}$.
(i) $L = \{w \in \{a, b\}^\star : w \text{ has more } a\text{'s than } b\text{'s}\}$.
(j) $L = \{w \in \{a, b\}^\star : w = zz, \text{ for some string } z\}$.
(k) $L = \{w \in \{a, b\}^\star : w = zzz, \text{ for some string } z\}$.
(l) $L = \{w \in \{a, b\}^\star : w = a^n b^{n+2}, \text{ where both } n \text{ and } n + 2 \text{ are primes}\}$.
(m) $L = \{w \in \{a, b\}^\star : w \text{ is an encoding of a C program that always halts}\}$.

IV. (Language operators) For languages L, L_1, and L_2, we define the following (derived) languages:

(a) (Union) $L_1 \cup L_2 = \{w : w \in L_1 \text{ or } w \in L_2\}$.
(b) (Intersection) $L_1 \cap L_2 = \{w : w \in L_1 \text{ and } w \in L_2\}$.
(c) (Concatenation) $L_1 L_2 = \{w_1 w_2 : w \in L_1 \text{ and } w \in L_2\}$.
(d) (Kleene star) $L^\star = \{w^n : w \in L, n \in \mathbb{N}\}$.
(e) (Complement) $L^c = \Sigma^\star \setminus L$.
(f) (Reversal) $L^R = \{w^R : w \in L\}$.

V. A **deterministic finite automata** (DFA) M is a five-tuple $M = (Q, \Sigma, \delta, s, F)$ where

(a) Q is a finite set of states,
(b) Σ is a finite alphabet,
(c) δ is a transition function $\delta : Q \times \Sigma \to Q$,
(d) $s \in Q$ is a start (initial) state,
(e) $F \subseteq Q$ is a subset of final (accepting) states.

[1] Last revised: March 07, 2012. Some treatment of the material here are adopted from Hopcroft and Ullman.

VI. We may inductively extend the transition function δ of a DFA to strings as follows. For any state q,

(i) $\delta(q, \epsilon) = q$,

(ii) $\delta(q, wa) = \delta(\delta(q, w), a)$, where $w \in \Sigma^\star$ and $a \in \Sigma$.

A **string** w **is accepted** by a DFA $M = (Q, \Sigma, \delta, s, F)$ iff $\delta(s, w) \in F$. The **language** L **accepted** by a DFA M is the set of all strings accepted by M. We use the notation $L(M) = \{w : \ M \text{ accepts } w\}$. A language is called **regular** iff it is accepted by a DFA.

VII. A **non-deterministic finite automata** (NFA) M is a five-tuple $M = (Q, \Sigma, \delta, s, F)$ where

(a) Q is a finite set of states,

(b) Σ is a finite alphabet,

(c) δ is a transition function $\delta : Q \times (\Sigma \cup \epsilon) \to \mathcal{P}(Q)$,

(d) $s \in Q$ is a start (initial) state,

(e) $F \subseteq Q$ is a subset of final (accepting) states.

VIII. In a similar way, we inductively extend the transition function δ of an NFA to strings. For any state q,

(i) $\delta(q, \epsilon) = \{q\}$,

(ii) $\delta(q, wa) = \bigcup\{\delta(q', a) : q' \in \delta(q, w)\}$, where $w \in \Sigma^\star$ and $a \in \Sigma$.

A **string** w **is accepted** by a NFA $M = (Q, \Sigma, \delta, s, F)$ iff $\delta(s, w) \cap F \neq \emptyset$. The **language** L **accepted** by a NFA M is the set of all strings accepted by M. As before, we denote this as $L(M)$.

IX. (**NFA = DFA**) A language L is regular iff L is accepted by an NFA.

Proof. (sketch) ($\Rightarrow$) If L is regular, then L is accepted by some DFA M. But M is also an NFA (whose ϵ-transitions are trivial).

($\Leftarrow$) If L is accepted by some NFA $M = (Q, \Sigma, \delta, s, F)$. We will convert M to a DFA M' using a *subset* construction. For a state q, the ϵ-CLOSURE of q is the set of all states reachable from q only by ϵ moves. The ϵ-CLOSURE of a set of states A is the union of ϵ-CLOSURE of q for each $q \in A$. The states of M' will be *all subsets* of Q. The start state s' of M' will be the ϵ-closure of s. For a subset K of states and a symbol $a \in \Sigma$, we define

$$\delta'(K, a) = \epsilon\text{-CLOSURE}\left(\bigcup\{\delta(q, a) : q \in K\}\right).$$

Also, we let $F' = \{K \subseteq Q : K \cap F \neq \emptyset\}$. This defines $M' = (\mathcal{P}(Q), \Sigma, \delta', s', F')$.

It remains to show that $w \in L(M)$ iff $w \in L(M')$. $\square$

X. A **regular expression** over Σ is (syntactically) defined *inductively* as follows:

(a) a is a regular expression, for each $a \in \Sigma$,

(b) ϵ is a regular expression,

(c) $\emptyset$ is a regular expression,

(d) (Kleene star) If R is a regular expression, then $(R^\star)$ is a regular expression,

(e) (Union) If R, S are regular expressions, then $(R \cup S)$ is a regular expression,

(f) (Concatenation) If R, S are regular expressions, then (RS) is a regular expression.

We will drop brackets if the context is clear by adopting the precedence order (from high to low) as $\star$ (closure), $\cdot$ (concatenation), and $\cup$ (union). So, $R \cup ST$ means $R \cup (ST)$ and not $(R \cup S)T$.

The semantics of regular expressions is given by a model $\mathfrak{L}$ whose domain is the set of all languages $\mathcal{P}(\Sigma^\star)$ over Σ. The model must give meaning to the function symbols in $\{\star^{(1)}, \cup^{(2)}, {}^{(2)}\}$, where ${}^{(2)}$ is the *invisible* concatenation operator. Thus, we have:

(a) $\mathfrak{L}(a) = \{a\}$.
(b) $\mathfrak{L}(\epsilon) = \{\epsilon\}$.
(c) $\mathfrak{L}(\emptyset) = \emptyset$.
(d) $\mathfrak{L}(R^\star) = \mathfrak{L}(R)^\star$.
(e) $\mathfrak{L}(R \cup S) = \mathfrak{L}(R) \cup \mathfrak{L}(S)$.
(f) $\mathfrak{L}(RS) = \mathfrak{L}(R)\mathfrak{L}(S)$.

The language **generated** by a regular expression R, namely $\mathfrak{M}(R)$, is denoted $L(R)$.

XI. (**RE = DFA**) A language L is regular iff L is generated by a regular expression.

Proof. (sketch)
($\Rightarrow$) Suppose L is accepted by a DFA M. We construct a regular expression R inductively so that $L(R) = L(M)$. Without loss of generality, let $M = (Q, \Sigma, \delta, s, F)$ where $Q = \{1, \ldots, n\}$. Let $R_{i,j}^{(k)}$ be a regular expression which generates all strings w for which M on w moves from state i to state j but passing only through states whose index is at most k (the start state i and the end state j are exempted from this restriction). We may compute $R_{i,j}^{(k)}$ for all values of i, j, and k by dynamic programming:

$$R_{i,j}^{(k)} = \begin{cases} R_{i,k}^{(k-1)}(R_{k,k}^{(k-1)})^\star R_{k,j}^{(k-1)} \cup R_{i,j}^{(k-1)} & \text{if } k > 0 \\ \{a : \delta(i,a) = j\} \cup \epsilon & \text{if } k = 0 \text{ and } i = j \\ \{a : \delta(i,a) = j\} & \text{if } k = 0 \text{ and } i \neq j \end{cases}$$

If 1 is the start state, then $R = \bigcup_{m \in F} R_{1,m}^{(n)}$ is a regular expression for all strings accepted by M.

Remark: An alternative proof of this direction may be obtained by the use of generalized NFA. A **generalized NFA** M is a five-tuple $M = (Q, \Sigma, \delta, s, f)$ where Q is a finite set of states, Σ is a finite alphabet, $\delta : (Q \setminus \{f\}) \times (Q \setminus \{s\}) \to \mathcal{R}$ is the transition function, s is the start state, and f is the accept state. Here $\mathcal{R}$ is the set of all regular expressions over Σ. The following is an algorithm to convert a GNFA to a regular expression (due to Sipser):

input: A GNFA $M = (Q, \Sigma, \delta, s, f)$ with k states
output: A regular expression R so $L(R) = L(M)$
if $k = 2$ **then**
    return the regular expression R labeling the edge (s, f)
else
    select q distinct from s and f

let $M' = (Q', \Sigma, \delta', s, f)$ be a GNFA where

$$Q' = Q \setminus \{q\}$$

and for any $q_i \in Q' \setminus \{f\}$ and $q_j \in Q' \setminus \{s\}$, let

$$\delta'(q_i, q_j) = R_1 R_2^{\star} R_3 \cup R_4$$

where $R_1 = \delta(q_i, q)$, $R_2 = \delta(q, q)$, $R_3 = \delta(q, q_j)$, and $R_4 = \delta(q_i, q_j)$.

recurse on M'

end if

($\Leftarrow$) Suppose that L is generated by a regular expression R. It suffices to show that there is an NFA M which accepts L. We show by induction that any regular expression R is accepted by an NFA.

(a) Case: $R = a$ (where $a \in \Sigma$), or $R = \epsilon$, or $R = \emptyset$
There are simple (one-state or two-state) DFAs accepting $L(R)$.

(b) Case: $R = (R_1 \cup R_2)$
Suppose M_1 and M_2 are NFAs accepting $L(R_1)$ and $L(R_2)$, respectively. The idea is to define two new states s' (new start state) and f' (new final state) Second, we add ϵ-moves from s' to the start states of M_1 and M_2. Finally, we connect any final states of M_1 and M_2 to f' using ϵ-moves.

(c) Case: $R = (R_1 R_2)$
Suppose M_1 and M_2 are NFAs accepting $L(R_1)$ and $L(R_2)$, respectively. The idea is to connect all final states of M_1 to the start state of M_2 using ϵ-moves.

(d) Case: $R = {R_0}^{\star}$
Suppose $M_0 = (Q, \Sigma, \delta, s, F)$ is an NFA accepting $L(R_0)$. The idea is to define a new start state s' which is accepting and connect s' with an ϵ-move to s. Also, we add from each accepting state $f \in F$ an ϵ-move back to s.

The above ideas capture the *closure* properties of regular languages.

□

XII. **Closure** properties:

(a) Any finite language is regular.
(b) There are countably many regular languages.
(c) Regular languages are closed under union, concatenation and Kleene star.
(d) Regular languages are closed under intersection, difference, and reverse.

XIII. (**Pumping Lemma**, Bar-Hillel, Perles, and Shamir [1961])
If a language L is regular, then there is N so for any string $w \in L$, with $|w| \geq N$, there are strings x, y, and z satisfying:

(a) $|y| > 0$.
(b) $w = xy^n z \in L$, for each $n \in \mathbb{N}$.

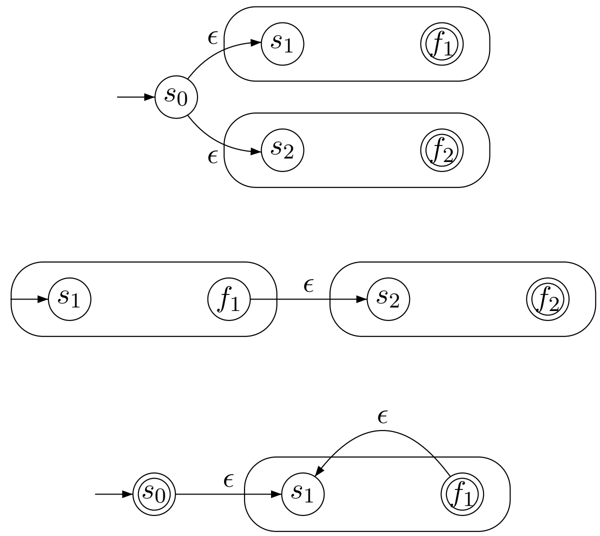


Figure 1: Closure under union, concatenation and Kleene star.

(c) $|xy| \leq N$.

Proof. (sketch) If L is regular, then it is accepted by some DFA $M = (Q, \Sigma, \delta, s, F)$ with say N states. Take any string w so that $|w| \geq N$. Consider the sequence of states on the accepting path from s to a state $f \in F$. By the pigeonhole principle, there is a state q which is repeated. In fact, after the first N moves (or symbols of w), there should be a repeated state. Let q be a state which is repeated after reading the first N symbols of w. So, we may write w as $w = xyz$, where $|xy| \leq N$, x brings s to q (for the first time), y brings q back to q ($y \neq \epsilon$), z brings q (second time at q) to f. □

XIV. (**Decision problems**) Let L be the set of strings accepted by a DFA with N states.

(a) L is **non-empty** iff there is a string $w \in L$ with $|w| < N$.

Proof. ($\Leftarrow$) Obvious.

($\Rightarrow$) Suppose L is non-empty and assume the all strings in L have length at least N. Let w be the shortest string in L. Note $|w| \geq N$ by assumption. Thus, we may apply the Pumping Lemma and write $w = xyz$, for some strings x, y, and z, where $y \neq \epsilon$, and $xz \in L$. This contradicts the choice of w as the shortest string in L. So, there is a string in L whose length is at most N. □

(b) L is **infinite** iff there is a string $w \in L$ with $N \leq |w| < 2N$.

Proof. ($\Leftarrow$) Suppose $w \in L$ with $N \leq |w| < 2N$. By the Pumping Lemma, $w = xyz$, for some strings x, y and z, where $y \neq \epsilon$, and $wy^iz \in L$, for all $i \geq 0$. The latter implies that L is infinite.

($\Rightarrow$) Suppose L is infinite but it contains no string w with $N \leq |w| < 2N$. Let $w \in L$ be the shortest string so that $|w| \geq 2N$. Such a string w exists since L is infinite and there are only finitely many strings of length at most N. By the Pumping Lemma, $w = xyz$ for some strings x, y, and z, where $y \neq \epsilon$ and $xz \in L$, for all $i \geq 0$. The latter contradicts the choice of w. So, L contains a string whose length is between N and $2N - 1$. □

The above shows that there are algorithms to decide if a regular language is empty or finite. There is also an algorithm to decide if two regular languages are equal.

(c) There is an algorithm to decide if two regular languages L_1 and L_2 are **equivalent**.

Proof. Let M_1 accept L_1 and M_2 accept L_2. The symmetric difference of L_1 and L_2 is

$$L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$$

where $A \setminus B = A \cap B^c$. Since regular languages are closed under complementation, union and intersection, they are closed under symmetric difference. But, $L_1 = L_2$ iff $L_1 \triangle L_2 = \emptyset$. Since emptiness of a regular language can be decided, we have an algorithm to decide if $L_1 = L_2$. □

XV. (**Minimization**) For an equivalence relation E, we denote $[x]_E$ as the equivalence class (or cell) corresponding to x; that is, $[x]_E = \{y : xEy\}$. Whenever the context is clear, we will drop the subscript E from $[x]_E$. The *index* of E is the number of cells that it has.

An equivalence relation E is *right-invariant* if xEy implies $(xz)E(yz)$ for all $z \in \Sigma^\star$. Given a DFA M, we may define an equivalence relation xR_My iff $\delta(s, x) = \delta(s, y)$. Note that R_M is right-invariant. Given a language L, we may also define an equivalence relation R_L as xR_Ly iff $xz \in L \Leftrightarrow yz \in L$, for any z.

The following result is called the Myhill-Nerode theorem (due to Nerode [1958]).

Theorem 1. *(Myhill-Nerode) The following are equivalent:*

(i) L is regular.

(ii) L is the union of cells of a right-invariant equivalence relation of finite index.

(iii) R_L has finite index.

Proof. (sketch)
$(i) \to (ii)$: If L is regular, it is accepted by a DFA M. Then, L is the union of cells of R_M corresponding to strings x for which $\delta(s, x) \in F$.

$(ii) \to (iii)$: Suppose L is the union of cells of a right-invariant relation E. It suffices to show that each cell of E is contained in a cell of R_L. For any x, consider $y \in [x]_E$. Since $yz \in [xz]_E$ for any z, we have $yz \in L \;\Leftrightarrow\; xz \in L$. This implies $y \in [x]_{R_L}$.

$(iii) \to (i)$: First, we note R_L is right-invariant. Suppose $y \in [x]_{R_L}$. Thus, $x\alpha \in L \;\Leftrightarrow\; y\alpha \in L$, for all α. We need to show that for any z, we have $yz \in [xz]_{R_L}$. But the latter requires $yzw \in [xzw]_{R_L}$ for all w, so we may simply take $\alpha = zw$. To show that L is regular, we build a DFA M for L defined with $Q = \{[x] : x \in \Sigma^\star\}$, $\delta([x], a) = [xa]$ (which is well-defined since R_L is right-invariant), $s = [\epsilon]$, and $F = \{[x] : x \in L\}$. It remains to check that $L = L(M)$. □

The DFA constructed in the above proof is the minimum state DFA which accepts L and is unique up to isomorphism (renaming of states). To see this, suppose there is DFA M which has fewer states than the number of classes of the right-invariant equivalence relation. Then, by the Pigeonhole principle, there are at least two such classes which are associated with a single state q in M. This is a contradiction.

Next, we consider a DFA minimization algorithm (due to Huffman [1954] and also to Moore [1956]):

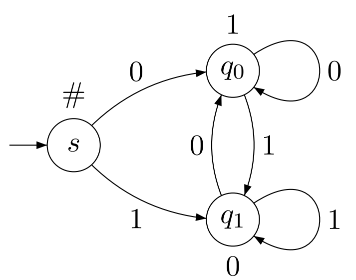
input: A DFA $M = (Q, \Sigma, \delta, s, F)$
output: An equivalent minimum DFA M'

```
for all p ∈ F and q ∈ Q \ F do
    mark (p, q)
end for
for all (p, q) ∈ F × F or (p, q) ∈ (Q \ F) × (Q \ F) do
    if ∃a so that (δ(p, a), δ(q, a)) is marked then
        mark (p, q)
        recursively mark all unmarked pairs on the list for (p, q) and on the list of other pairs marked
        at this step.
    else
        for all input symbol a do
            put (p, q) on the list for (δ(p, a), δ(q, a)) unless δ(p, a) = δ(q, a)
        end for
    end if
end for
```

XVI. (**Transducers**) We describe some models of finite automata with output.

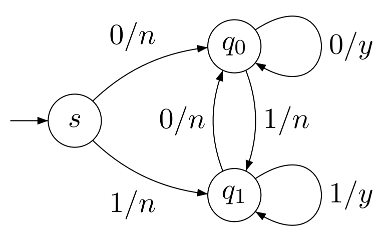
A **Moore machine** M is a six-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, s)$ where Q, Σ, δ and s are similar to a DFA but where Δ is the *output* alphabet and $\lambda : Q \to \Delta$ is the state output function.

Example: Switching 0s and 1s.



A **Mealy machine** M is a six-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, s)$ where each component is similar to a Moore machine but where $\lambda : Q \times \Sigma \to \Delta$ is the transition output function.

Example: $R = (0 \cup 1)^\star(00 \cup 11)$.



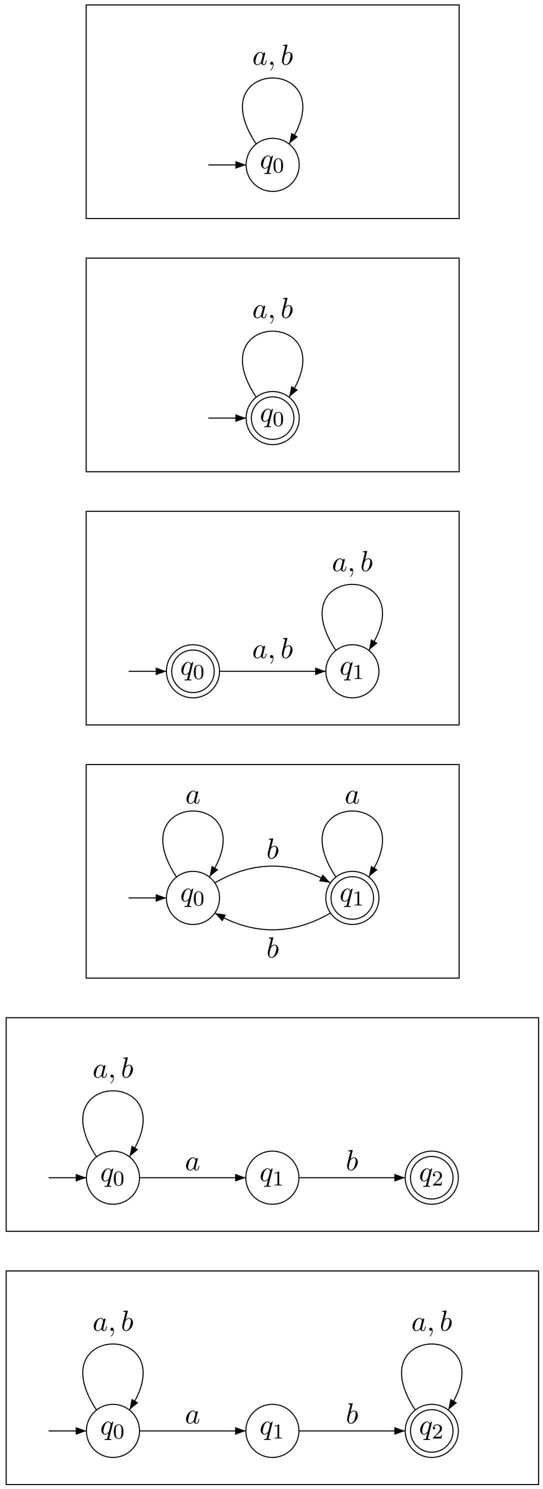


Figure 2: Identify these little automatons!