Formal Language and Automata Theory (CS21004)

Soumyajit Dey
CSE, IIT Kharagpur

Announcements

- The slide is just a short summary
- Follow the discussion and the boardwork
- Solve problems (apart from those we dish out in class)

Table of Contents

Patterns

A *Pattern* captures a family of strings (like FA) : language of the pattern

- $a \in \Sigma$: $L(a) = \{a\}$: matches a single symbol
- ϵ : $\epsilon = \{\epsilon\}$: matches the null string
- ϕ : $L(\phi) = \phi$: matches empty set ϕ
- $\sharp$: $L(\sharp) = \Sigma$: any symbol
- @ : $L(@) = \Sigma^*$: any string

Above *atomic* patterns can be operated with unary $^*, ^+, \neg$ or connected by $\cup/+, \cap, \circ$ (usually not written, kept silent) to generated other valid patterns

Patterns

- x matches $\alpha + \beta$ if x matches either α or β : $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$, similarly you have other rules
- $L(\alpha \cap \beta) = L(\alpha) \cap L(\beta)$, $L(\alpha\beta) = L(\alpha)L(\beta)$, $L(\neg\alpha) = \neg L(\alpha) = \Sigma^* - Ł(\alpha)$,
- Can define $L(\alpha^*)$, $L(\alpha^+)$
- patterns are collections of strings over $\Sigma, \sharp, @, \boldsymbol{\epsilon}, \boldsymbol{\phi}, \neg, ^*, ^+$,

Note 1: $\boldsymbol{\epsilon}, \boldsymbol{\phi}$ and ϵ, ϕ are different. The boldfaces are symbols in the pattern language
Note 2:'$+$' is associative over patterns

Patterns : Applications

- Pattern matching is an important application of FA
- Unix regular exps are basically patterns (Σ ??)
- Unix commands 'grep', 'egrep' use FA inside their implementations

Patterns : examples

- Strings containing at least 3 occurrences of a : @a@a@a@
- all single letters except a : $\# \cap \neg a$
- Strings with no occurrences of a : $(\# \cap \neg a)^*$

Some books call patterns as regular expressions, we shall make a distinction here.

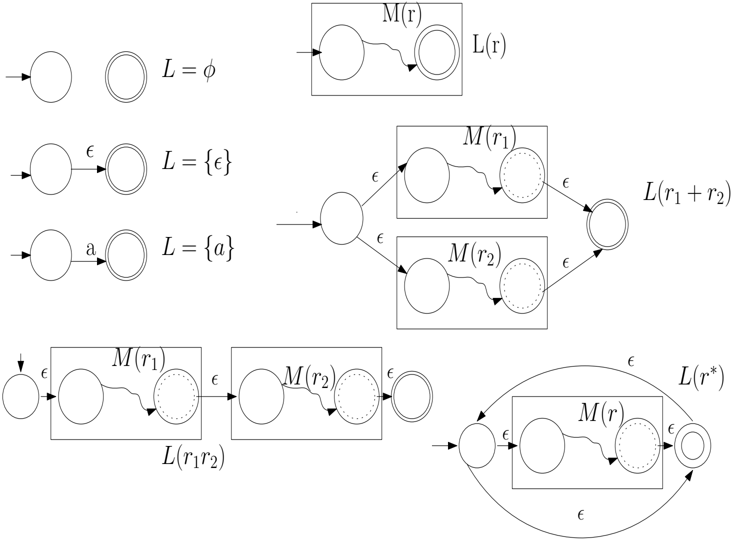
Table of Contents

Regular Expressions

Represents the idea of patterns as regular language (set) generators ($\equiv$ FA) in a minimal way using only Σ, ϵ, ϕ as primitive regular expressions and operators $+, \circ, ^*$ along with the option of using parenthesis ('(' and ')') to denote operator association.

- $L((aa)^*(bb)^*b) = \{a^{2n}b^{2m+1} \mid n, m \geq 0\}$
- Regex for 'at least one pair of consecutive 0's' : $(0+1)^*00(0+1)^*$
- Regex for 'no pair of consecutive 0's' : $(1^*011^*)^*(0+\epsilon) + 1^*(0+\epsilon) \equiv (1+01)^*(0+\epsilon)$ – how to reason about such equivalence ??

Regular Expressions, Languages (sets) and FA



Regular Expressions, Languages (sets) and FA

- Automata for primitive regex can be combined as shown
- We can give a formal method for constructing states/transitions of combined machine from states/transitions of simpler machines
- We can prove language equivalence of regex and automata generated as above by induction on number of operators

DFA to Regex ??

- Let states be $\{1, 2, \cdots, n\}$
- Let $R(k)_{i,j}$ be the Regex whose language is the set of labels of path from i to j without visiting any state with label larger than k.
- Basis : $R(0)_{i,j}$ is basically labels of direct paths from i to j, i.e., $R(0)_{i,j} = a_1 + \cdots + a_n$ if $\delta(i, a_k) = j$ for $1 \leq k \leq n$; Note $R(0)_{i,i} = \phi$ if there is no self-loop
- Induction : $R(k)_{i,j} = R(k-1)_{i,j} + R(k-1)_{i,k} \cdot (R(k-1)_{k,k})^* \cdot R(k-1)_{k,j}$

Overall Regex : $R(n)_{i_0,f_1} + R(n)_{i_0,f_2} + \cdots + R(n)_{i_0,f_k}$ with i_0 being the initial state and $\{f_1, \cdots, f_k\}$ being the set of final states.
COMPLEXITY ??? up to n^3 expressions, each step creates 4 terms for one term.

Using the algorithm, NFA $\Rightarrow$ DFA $\Rightarrow$ Regex AND Regex $\Rightarrow$ NFA $\Rightarrow$ DFA

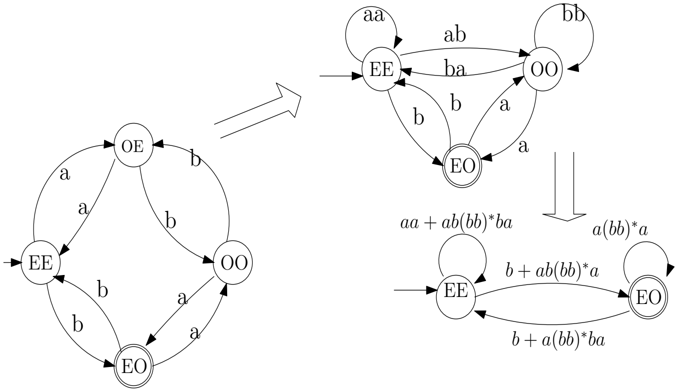
Observation : The algorithm we presented can also be devised for NFAs (check Kozen), use Δ instead of δ.

Alternate Method :

- Collapse states by allowing regex as transition labels
- Derive regex which connect initial and final state pairs
- Overall expression is the union of such regex

Example of collapsing

Consider $\{w \mid n_a(w) \text{ is even}, n_b(w) \text{ is odd}\}$: difficult to conceive regex directly.



Absence of transition between any state pair can be thought of as ϕ

Note : $r\phi = \phi \quad r + \phi = r \quad \phi^* = \lambda$

Table of Contents

Regular Grammars

Regular Language (set) $\equiv$ FA $\equiv$ Regular Grammar

- A grammar $G = (V, \Sigma, S, P)$ is **right linear** if all productions are of the form

$$A \to xB,\ A \to x$$

where $A, B \in V,\ x \in \Sigma^*$

- A grammar $G = (V, \Sigma, S, P)$ is **left linear** if all productions are of the form

$$A \to Bx,\ A \to x$$

where $A, B \in V,\ x \in \Sigma^*$

- A regular grammar is one of the two

Strictly Right Linear Grammars

A grammar $G = (V, \Sigma, S, P)$ is **strictly right linear** if all productions are of the form

$$A \to xB,\ A \to \lambda$$

where $A, B \in V$, $x \in \Sigma \cup \{\lambda\}$

- Any derivation of a word w from S has the form $S \Rightarrow x_1A_1 \Rightarrow x_1x_2A_2 \Rightarrow \cdots \Rightarrow x_1 \cdots x_nA_n \Rightarrow x_1 \cdots x_n$
- some x_i can be λ, connection with NFA ??

♠ For any right-linear grammar G there exists a strictly right-linear grammar H such that $L(G) = L(H)$

Strictly Right Linear Grammars

If G is a strictly right-linear grammar, then $L(G)$ is regular

- Given $G = (V, \Sigma, P, S)$, construct NFA $N = (V, \Sigma, \delta, S, F)$. The set of states is simply the set of non-terminals of G. The start state corresponds to the start variable.
- $\delta(A, x) = \{B \mid A \to xB \in P\}$, $F = \{C \mid C \to \lambda \in P\}$

To show $L(G) = L(N)$

- $L(G) \subseteq L(N)$: by induction on the structure of the derivation
- $L(N) \subseteq L(G)$: by induction on the structure of the computation

Strictly Right Linear Grammars

If A is a regular language, then there is a strictly right-linear grammar G such that $L(G) = A$

- If L is regular, $\exists$ an NFA $N = (Q, \Sigma, \delta, q_0, F)$ for L
- We construct a strictly right-linear grammar $G = (V = Q, \Sigma, P, S = q_0)$ where

$$P = \{A \to xB \,|\, B \in \delta(A, x)\} \cup \{C \to \lambda \,|\, C \in F\}$$

♠ A language A is regular **if and only if** there is a strictly right-linear grammar G such that $L(G) = A$

Regular Grammars

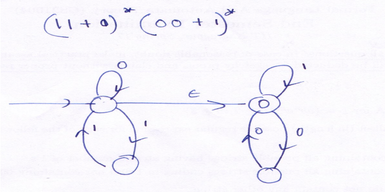
Grammar comprising both left-linear and right-linear rules will not necessarily generate a regular language.
Consider

$$
\begin{aligned}
S &\rightarrow 0A \\
S &\rightarrow 1B \\
S &\rightarrow \lambda \\
A &\rightarrow S0 \\
B &\rightarrow S1
\end{aligned}
$$

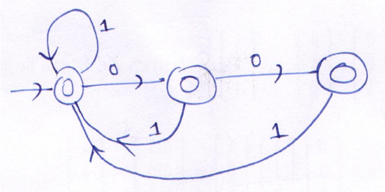
The grammar generates $L = \{ww^R \mid w \in \{0,1\}^*\}$ which is not regular

Table of Contents

Regex example



Set of all strings without strings having two adjacent zeroes.



Regex example

- The second automata is for the language $\Sigma^* \setminus$ "strings having more than 2 consecutive zero-s"
- Regex ?

$(1 + 01 + 001)^*(\epsilon + 0 + 00) \equiv ((\epsilon + 0 + 00)1)^*(\epsilon + 0 + 00) \equiv$
$((\epsilon + 0)(\epsilon + 0)1)^*(\epsilon + 0)(\epsilon + 0)$

How do we perform such equational reasoning, well there exists an underlying algebra

RegEX Simplication Laws

Let α, β be regex; if $L(\alpha) = L(\beta)$, we say $\alpha \equiv \beta$. $\equiv$ is an equivalence relation.

$$\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma \tag{1}$$

$$\alpha + \beta \equiv \beta + \alpha \tag{2}$$

$$\alpha + \boldsymbol{\phi} \equiv \alpha + \alpha \equiv \alpha \tag{3}$$

$$(\alpha\beta)\gamma \equiv \alpha(\beta\gamma) \tag{4}$$

$$\boldsymbol{\epsilon}\alpha \equiv \alpha\boldsymbol{\epsilon} \equiv \alpha \tag{5}$$

$$\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma \tag{6}$$

$$(\beta + \gamma)\alpha \equiv \beta\alpha + \gamma\alpha \tag{7}$$

$$\boldsymbol{\alpha\phi} \equiv \boldsymbol{\phi\alpha} = \boldsymbol{\phi} \tag{8}$$

$$\boldsymbol{\epsilon} + \alpha\alpha^* \equiv \boldsymbol{\epsilon} + \alpha^*\alpha \equiv \alpha^* \tag{9}$$

$$\beta + \alpha\gamma \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma \tag{10}$$

$$\beta + \gamma\alpha \leq \gamma \Rightarrow \beta\alpha^* \leq \gamma \tag{11}$$

Kleene Algebra

$\leq$ in previous slide refers to subset ordering, i.e.
$\alpha \leq \beta \Leftrightarrow L(\alpha) \subseteq L(\beta) \Leftrightarrow L(\alpha + \beta) = L(\beta) \Leftrightarrow \alpha + \beta = \beta$

- Any algebraic structure that satisfies Eq 1 -11 is a Kleene Algebra (Named after Stephen C. Kleene, who invented regular sets)
- Note that the set 2^{Σ^*} (all possible subsets of Σ^*) with constants ϕ, ϵ and operations $\cup, \cdot, *$ forms a Kleene algebra,
- Similarly, the family of all regular subsets, i.e. all Regular expressions (set K say) forms a Kleene Algebra $(K, +, \cdot, *, 1, 0)$ with operations $+, \cdots, *$ and constants $1 = \epsilon$, $0 = \phi$.
- Note: $1a = a1 = a$, $a0 = 0a = 0$ i.e. 1 is an identity for $\cdot$ and 0 is an annihilator for $\cdot$.
- Comment: This is an *idempotent* semiring.

More properties

Many such relations can be derived using the rules listed. Examples:

$$a^* a^* = a^* \tag{12}$$

$$a^{**} = a^* \tag{13}$$

$$(a^* b)^* a^* = (a+b)^* \quad \text{denesting rule} \tag{14}$$

$$a(ba)^* = (ab)^* a \quad \text{shifting rule} \tag{15}$$

$$a^* = (aa)^* + a(aa)^* \tag{16}$$

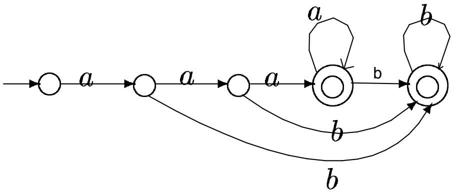
Arden's Theorem: In any Kleene Algebra, $a^* b$ is the $\leq$-least solution of the equation $x = ax + b$. Also, ba^* is the solution of $x = xa + b$.

Arden's Theorem:

We know that set of languages 2^{Σ^*} under the operations union and concatenation is also a Kleene Algebra. Hence Ardens theorem can be stated in terms of languages as follows.
"A^*B is the smallest language that is a solution for X in the linear equation $X = A \cdot X \cup B$ where X, A, B are sets of strings."

Arden's Theorem usage: FA to RegEX

Consider the automaton given below.



Let the states be $\{0, 1, \cdots, 4\}$ left to right. Let A_i be set of strings accepted from state i. We write $A_i = \cup\{\sigma A_j \,|\, \text{state } j \text{ is accessible from state } i \text{ through transition } \sigma\}$. Hence, A_0 is the language of the automaton if 0 is the initial state.

Arden's Theorem usage: FA to RegEX

We can write the following systems of Eqs.

$$
\begin{aligned}
A_0 &= aA_1 \\
A_1 &= aA_2 \cup bA_4 \\
A_2 &= aA_3 \cup bA_4 \\
A_3 &= \epsilon \cup aA_3 \cup bA_4 \\
A_4 &= \epsilon \cup bA_4
\end{aligned}
$$

ϵ is considered for equation written for accepting state (follows from definition of A_i).

Arden's Theorem usage: FA to RegEX

Applying Arden's Thm: $A_4 = b^*\epsilon = b^*$
$A_3 = a^*(b^+\epsilon) = a^*b^*$
$A_2 = a^+b^* \cup b^+$
$A_1 = a(a^+b^* \cup b^+) \cup b^+$
$A_0 = a(a(a^+b^* \cup b^+) \cup b^+) = a^2a^+b^* \cup a^2b^+ \cup ab^+$
Hence, the FA has an equivalent RegEX given by $a^2a^+b^* + a^2b^+ + ab^+$