

Regular Expressions

We have seen that there are infinite languages.

We are interested in

infinite languages with finite representations.

Regular expressions are the simplest finite representation.

They correspond to and are accepted by

“Finite State Automata” (singular: automaton).

Slide Lecture 2 –39

Languages in lexicographic order are **well-founded**, and this allows us to use induction to prove properties of languages.

Operations on Languages

Let L_1 and L_2 be languages over the alphabet Σ

The **concatenation** of L_1 and L_2 , denoted L_1L_2 , is the set:

$$\{w \mid w = xy, x \in L_1, y \in L_2\}$$

Slide Lecture 2 –40

The **positive closure** of L , denoted L^+ , is the set of strings obtained by concatenating one or more strings of L

(i.e., it does not contain λ)

The **Kleene closure** of L , denoted by L^* , is

$$L^+ \cup \{\lambda\}$$

The **complement** of L , denoted \bar{L} , is the set

$$\Sigma^* - L$$

all strings in Σ^* not contained in L

Slide Lecture 2 –41

This finite notation allows us to view a computing machine as a “string processor”, and a computer program as a finite string of some programming language.

Slide Lecture 2 –42

Finite State Machines with deterministic behavior are called **Deterministic Finite State Automata** (DFA).

A DFA is defined as a 5-tuple

$$M = (K, \Sigma, \Delta, S, F)$$

1. K is a finite set of states
2. Σ is an alphabet
3. Δ is the deterministic state transition function that maps a state and an input to another state

$$K \times \Sigma \longrightarrow K$$

Slide Lecture 2 -43

4. S is the initial state, $S \in K$
5. F is the set of final states, $F \subseteq K$

Slide Lecture 2 -44

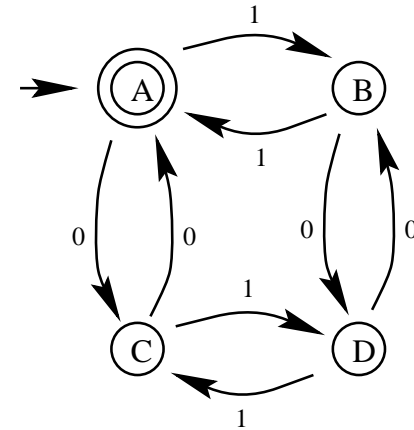


Figure 1: A Finite-State Automaton

Slide Lecture 2 -45

What does this machine do?

Accepts 11, 00, 1010, 0101, 1001, 100001

Accepts strings with an even number of 1's
and an even number of 0's.

Slide Lecture 2 -46

$K = \{ABCD\} \quad \Sigma = \{0, 1\}$

The Transition Function

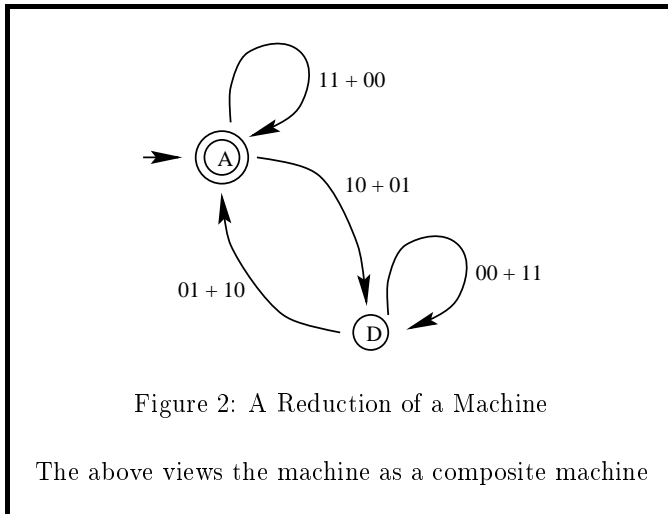
		Inputs	
		0	1
State	A	C	B
	B	D	A
	C	A	D
	D	B	C

$S = A \quad F = \{A\}$

Slide Lecture 2 -47

$(11 + 00 + [(10 + 01)(00 + 11) * (10 + 01)])*$

Slide Lecture 2 -49



Slide Lecture 2 -48

Regular Expressions to describe finite state machines are defined inductively:

1. λ and ϕ are regular expressions
2. every symbol $x \in \Sigma$ is a regular expression
3. if R_1 and R_2 are regular expressions, so are:
 - $R_1 + R_2$ (union)
 - $R_1 R_2$ (concatenation)
 - $(R_1)^*$ (Kleene closure)

Slide Lecture 2 -50

WHAT ABOUT THESE?

1. All words $\in \Sigma^*$ containing 2 consecutive a's or 2 consecutive b's
2. All words $\in \Sigma^*$ containing n a's followed by n b's
(Can we do this with a FSA?)

Two regular expressions are **equivalent** iff

$$\tilde{R}_1 = \tilde{R}_2$$

Example:

$$b(ab)^* = (ba)^*b$$

Slide Lecture 2 –55

Inductive proof:

Step 1

Both generate 'b'

Choosing 1 iteration

bab

bab

Step 2

Assume $b(ab)^n = (ba)^nb$

Slide Lecture 2 –56

Step 3 show for $n + 1$

$$b(ab)^nab = (ba)^nbab$$

$$[b(ab)^n]ab = [(ba)^nb]ab$$

Also note there are $n + 1$ a's in both cases

Slide Lecture 2 –57

Properties of Regular Expressions

Let R, S, T be regular expressions over Σ

1. $R + \Phi = \Phi + R$
2. $R + S = S + R$
3. $R + R = R$
4. $R + (S + T) = (R + S) + T$

(+ is commutative, idempotent, and associative)

Slide Lecture 2 –58

$$1. R \lambda = \lambda R = R$$

$$2. R \Phi = \Phi R = \Phi$$

$$3. (RS) T = R (ST)$$

(concatenation has identity and cancellation elements,
concatenation is also associative)

Note: $RS \neq SR$ for arbitrary R and S

Slide Lecture 2 -59

8. Arden's Rule:

Assume $\lambda \notin \tilde{S}$. Then

$$\text{a. } R = SR + T \text{ iff } R = S^* T$$

$$\text{b. } R = RS + T \text{ iff } R = T S^*$$

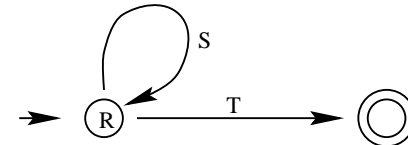


Figure 3: Arden's Rule

Slide Lecture 2 -61

$$1. R(S + T) = RS + RT$$

What about $R + (ST) \stackrel{?}{=} (R + S)(R + T)$

$$2. R^* = R^* R^* = (R^*)^* = (\lambda + R)^*$$

$$3. (R + S)^* = (R^* + S^*)^*$$

$$4. (R + S)^* = (R^* S^*)^* = (R^* S)^* R^*$$

Note: $(R + S)^* \neq R^* + S^*$

$$5. R^* R = R R^*$$

$$6. R(SR)^* = (RS)^* R$$

$$7. (R^* S)^* = \lambda + (R + S)^* S$$

Slide Lecture 2 -60

MEALY and MOORE machines

We have been looking at Moore Machines, where an output is associated with a state: hence, these are also known as **State-Assigned Machines**

Note that a Moore Machine has an output for the initial state, and hence an output for λ .

Slide Lecture 2 -62

A Mealy Machine is transition assigned.
 Note that a Mealy Machine does not have an output associated with the initial state, and thus has no output for λ .

Slide Lecture 2 -63

A transition assigned (Mealy) machine, M_t , and a state assigned (Moore) machine, M_s , are *similar* is, for any input, the output of M_s is exactly the same as the output of M_t *preceded by one arbitrary but fixed symbol*.

This is because M_s has an output for the initial state.

Slide Lecture 2 -65

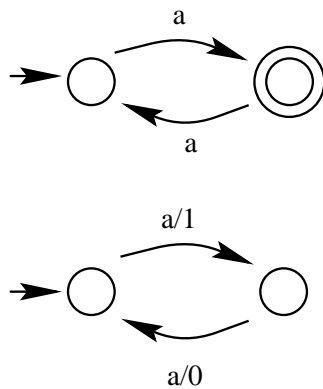


Figure 4: Equivalent Mealy and Moore Machines

Slide Lecture 2 -64

One can easily convert a state assigned machine into a transition assigned machine.

For each computation

$$\delta(K_x, \alpha) \longrightarrow K_y$$

Move the output associated with K_y to the transition between K_x and K_y .

Slide Lecture 2 -66

To convert a transition assigned machine into a state assigned machine, one must move the output associated with the computation

$$\delta(K_x, \alpha) \longrightarrow K_y$$

from the transition to the state K_y .

Note that we may now need multiple copies of K_y .

E.G. $K_{y-outputs-1}$ $K_{y-outputs-0}$

Slide Lecture 2 -67

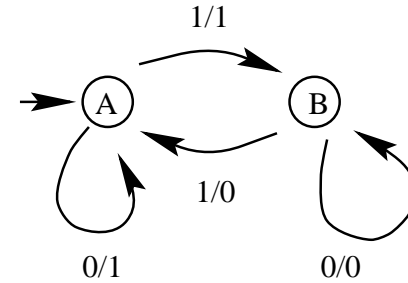


Figure 5: A Mealy Transition Assigned Machine

Slide Lecture 2 -69

For each output i and state K_y , we construct a new state,

$$K_{y-outputs-i}$$

Each transition that goes to K_y and outputs i is mapped to state $K_{y-outputs-i}$.

Each new state $K_{y-outputs-i}$ inherits all of the outgoing transitions associated with K_y .

Slide Lecture 2 -68

	0	1
A0	A1	B1
A1	A1	B1
B0	B0	A0
B1	B0	A0

Note that there are now two initial states. You can keep two initial states (a nondeterminism!) or you can pick either initial state.

Slide Lecture 2 -70