# Lecture 23

Andrea LaPaugh

April 30, 1996

# Finite State Machines

(a.k.a finite automata)

work on sequences of inputs

outputs depend on history of inputs seen

BUT only fixed, finite number of states (N)

means can only depend on history

for fixed time in past

i.e. for sequence going back only so far

# Finite State Acceptors

Simplies kind of finite state machines

RULES: I) One input, one output (binary)

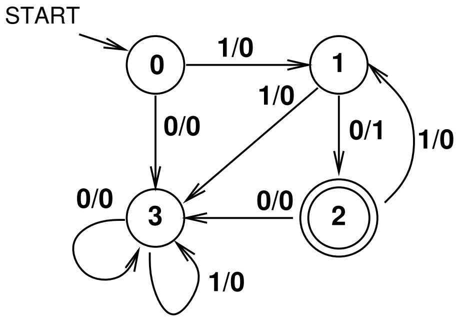
II) If output is "1" when change
to a certain state, then
output is "1" whenever change
to that state (accepting state)

Look at set of "accepted" input sequences
when output becomes "1", accept input
string defined from first input
to point where accept

Say FSA "recognizes" input string or
"decides" whether input string is
in the "language" it can recognize

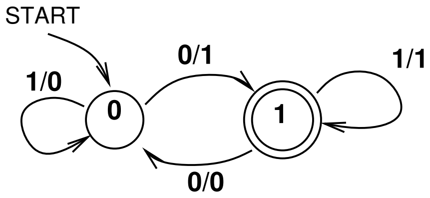
# Examples

one or more repetitions of 10




10101010?

odd number of 0's




0001110?

Accepting state has double circle.

In general, can be more than one
accepting state.

## Limitations of Finite State Acceptors

FSAs can't "count"

Theorem: No finite state acceptor can decide whether or not its input has the same number of 0's and 1's.

Proof:

Suppose an N state machine does it.

Give it N+1 0's followed by N+1 1's

Some state must be revisited

Machine would accept the same string without the intervening 0's

That string doesn't have the same number of 0's and 1's

Contradiction.

0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

## Regular Expressions

In fact, we can write all the sets of binary strings (languages) accepted by FSAs using a pattern matching language called "regular expressions" (REs)

Formal Definition of Regular Expression (essential combinations)

1) symbol 0 or 1
2) ab   where a and b are regular expressions
   = concatenate
3) a+b   where a and b are regular expressions
   = OR
4) a*   where a is a regular expression
   = repetition

# REs: Example and notes

Examples:

0110

0110 + 1001

(10)*

(00+11)*

1(00)*1

Regular expressions have lots of practical uses

editors

unix "grep"

Can use other "alphabets" than {0,1}

# FSAs are equivalent to REs

Finite State Acceptors and Regular Expressions are ``equally powerful`` ways of doing the same thing

For any RE, there is an FSA that

* accepts strings matching the RE
* rejects strings not matching the RE

Can construct FSA automatically from RE in simple fashion

For any FSA, there is a RE that

* matches all strings accepted by the FSA
* matches no other strings

This is called a "one-to-one correspondence" between FSAs and REs