SAN JOSE STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING

CS 154 Formal Languages and Computability Spring 2012
Section 1 Room MH 225 Class 3: 02-01-12

T. Howell
Summary of Class 2

M = (Q, Σ, δ, s, F) (a 5-tuple)

Other ways to write down a finite automaton:

Table:

		a	b
→	0	1	0
	1	2	1
	2	3	2
	*3	3	3

Transition diagram:

Example 1
accepted any string with 3 or more a's.
Example 2
accepted any string with always 1, 2, or 3 0's between consecutive 1's.

The <u>language</u> accepted by M is the set of strings accepted by M and is denoted L(M).
A subset $A \subseteq \Sigma^*$ is <u>regular</u> if A = L(M) for some finite automaton M.

<u>New Material</u>
Example 3: <u>(might have been covered last class)</u>
Consider $\{x \in \{0,1\}^* \mid x$ represents a multiple of 3 in binary$\}$
0, 11, 110, 1001, 1100, 1111, 10010, ... should be accepted. Other strings should be rejected.
Here is an automaton accepting this set:

		0	1
→	*0	0	1
	1	2	0
	2	1	2

Talk about how it works.
State **i** means the number represented by the symbols seen so far modulo 3 is **i**.

We prove by induction that is works.
Let N(x) denote the number represented by string x. (Note N(ε) = 0.)

We want to show that
$\hat{\delta}(0,x) = N(x) \bmod 3$ (1)
We will use a simple numerical fact: N(xc) = 2N(x) + c for c $\in$ {0,1}. (2)
Notice that in our table, δ(q, c) = (2q + c) mod 3. (3)

Basis
For x = ε, $\hat{\delta}(0,\varepsilon) = 0 = N(\varepsilon) \bmod 3$.

Inductive step
Our inductive hypothesis is $\hat{\delta}(0,x) = N(x) \bmod 3$ for all strings x of length at most n.
Assuming (1) is true for some string x, we show it is also true for xc, where c $\in$ {0,1}.

$\hat{\delta}(0,xc) = \delta(\hat{\delta}(0,x),c)$	by def of $\hat{\delta}$
= δ(N(x) mod 3, c)	by inductive hypothesis
= (2(N(x) mod 3) + c) mod 3	by (3)
= (2N(x) + c) mod 3	by properties of numbers, mod
= N(xc) mod 3	by (2)

So statement (1) holds for all strings x of length at most n+1.
By induction, the hypothesis is true for all strings x $\in$ $\{0,1\}^*$.

<u>Nondeterminism</u>
Nondeterminism means there is more than one choice for the next state is a computation. It is useful in describing distributed computations in which the exact sequence of actions is not fully determined. It often provides a simpler or more natural way of describing a language.

We will soon show that nondeterminism does <u>not</u> increase the computational power of a finite automaton.

A nondeterministic FA has a transition table with a set of states for each entry instead of just a single state. There is no rule to say which state to choose. If there is some way to choose states that leads to an accepting state, the input is accepted. Informally, we sometimes say the machine "guesses" the state to go to.

Example 4:
A = {x $\in$ $\{0,1\}^*$ | the fifth symbol from the right is 1.}

		0	1
→	u	{u}	{u, v}
	v	{w}	{w}
	w	{x}	{x}
	x	{y}	{y}
	y	{z}	{z}
	*z	∅	∅

If the input string has 1 in the 5th position from the right, there is a legal sequence of transitions leading to an accept state. If the input string has 0 in the 5th position from the right, there is no legal sequence of transitions leading to an accept state.

We can think of the machine as "guessing" when to use the transition to state v. After guessing, the remaining states "check" that the guess was correct. The string is accepted if there is a way, however lucky, for the machine to "guess" a sequence of legal transitions leading to an accepting state.

Example 5 (from HMU)
B = {x ∈ {0,1}* | x ends with 01.}

		0	1
→	u	{u, v}	{u}
	v	∅	{w}
	*w	∅	∅

At each step we update the *set of states* the NFA can be in. For input 10010 the sequence of states goes {u}, {u}, {u, v}, {u, v}, {u, w}, {u, v}. Since {u, v}∩ F = ∅, this string is rejected.

For input 1001, however, the sequence is {u}, {u}, {u, v}, {u, v}, {u, w}, and {u, w}∩ F = {w}, and the string is accepted.

<u>Converting NFA to DFA</u>
There is a simple algorithm for converting an NFA to a DFA. It is called the subset construction.

Create a state in the DFA for every subset of states in the NFA. The start state of the DFA is {s}. The successor state for subset T and input x is the union of the successor state sets for the elements of T with input a:

$$\delta_D(T, a) = \bigcup_{p \in T} \delta_N(p, a)$$

The accepting states of the DFA are all subsets that contain an accepting state of the NFA.

The DFA for the NFA of Example 5 looks like this:

		0	1
→	{u}	{u, v}	{u}
	{u, v}	{u, v}	{u, w}
	*{u, w}	{u, v}	{u}

Note that we did not need to create states for all 8 subsets of {u, v, w}. We created the ones we needed as we went along, and we only needed three of the eight. This worked out very nicely for Example 5. Students should try the procedure for Example 4 to practice the algorithm and to see why I did not do it in class.