Lecture 2 (~~[illegible]~~)

An alphabet Σ is a finite set of symbols.

$\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, c\}$, ..

A word over Σ is a finite sequence of symbols from Σ. Thus if $\Sigma = \{a, b, c\}$ words are, for example, a, ba, $abba$, $cabac$, ε : the empty string. The set of all possible words is Σ^*. It is always infinite when $\Sigma \neq \phi$. ~~If~~ We have $\phi^* = \{\varepsilon\}$. Note: $\varepsilon \neq \phi$.

A language is just a subset of Σ^*. If we take $\Sigma = \{a, b, c\}$ examples of languages are:

(1) $\{\varepsilon, a, aa, aaa, aaaa, \ldots\} = \{a\}^*$

(2) $\{\varepsilon$, [illegible] $\ldots\}$ — what pattern do I have in mind

We need a way of describing languages: these ~~are~~ will be studied later.

How do we recognize patterns?

Def A deterministic finite automaton (or finite-state machine) is a 4-tuple $A = (\cancel{\Sigma}, S, s_0, \delta : S \times \Sigma \to S, F \subseteq S)$

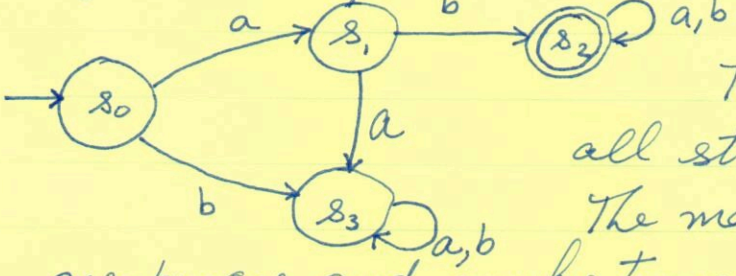
where S is a finite set of states

$s_0 \in S$ is the initial state

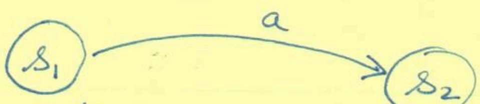
$\delta : S \times \Sigma \to S$ is the transition function

$F \subseteq S$ are the final states or accepting states.

To give examples we draw pictures:



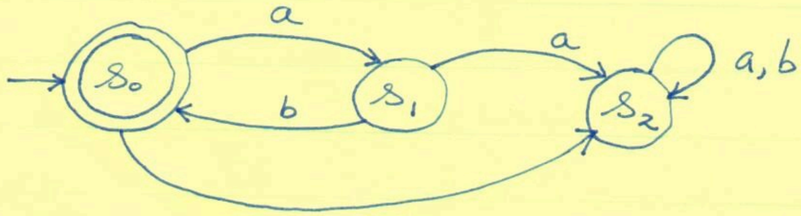
This machine accepts all strings that start with "ab". The machine reads symbols one-by-one and makes transitions. A word is accepted if the machine is in an accept state at the end of the word



state another state

$\delta(s_1, a) = s_2$

<u>Example</u>



Accepts ϵ, ab, abab, ababab, ... $(ab)^n$, ...
and <u>nothing</u> else.

<u>Basic cycle</u> (1) read a letter (2) change state (3) read next letter (4) if there are no more symbols STOP (5) If the m/c is in a state in F then accept else reject.

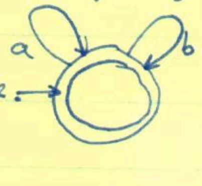
So machines or automata define <u>languages</u>.

<u>Def</u> A language that can be recognized by a DFA is called a <u>regular</u> <u>language</u>.

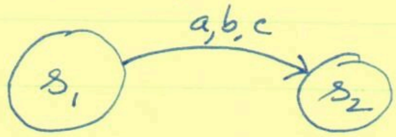
When you design a machine to recognize $L \subseteq \Sigma^*$ it must (1) accept every word in L and (2) reject every word not in L. If you do (1) but not (2) you get 0; you do not get 1/2 credit.

The "size" of a language has almost nothing to do with how hard it is to recognize it.

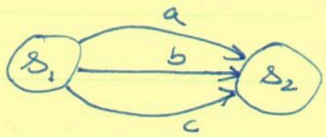
This mickey mouse machine recognizes Σ^* the biggest language.

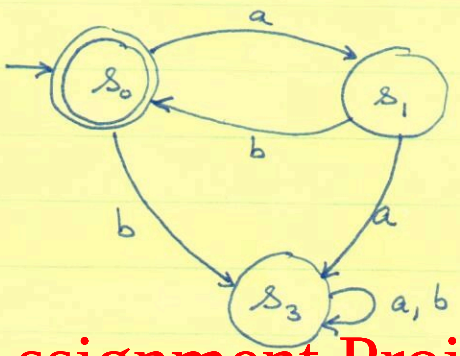


<u>Every</u> state has to have an arrow labelled with <u>every</u> symbol. Sometimes we write



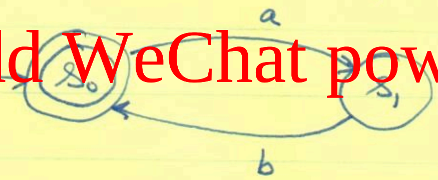
for





S_3 is a dead state: we can never get from it to an accept state. We cannot even get out of S_3.

For shorthand we may leave out the dead state and the arrows that go to it:



<u>Example</u> $\Sigma = \{1, 2, 3, 4, 5, \cancel{6}\}$ combination 45213

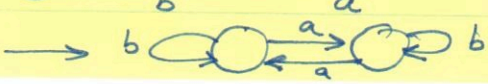
Drawing the dead state would introduce a whole lot of extra arrows.

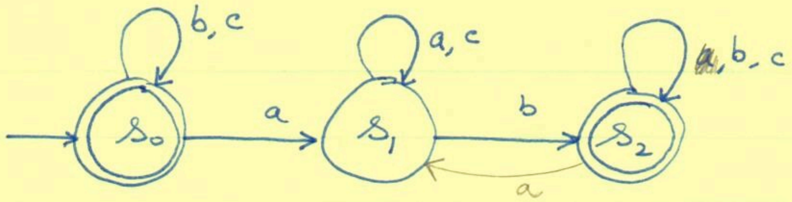
<u>Prop</u> Every finite language is regular.

<u>Example</u> We could in principle have <u>unreachable</u> states

We will assume they are removed.

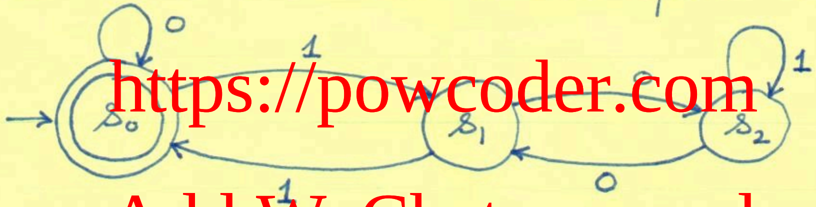
<u>Useless</u>!





If there is an "a" there must be a "b" sometime later. aaaccaabccb is OK ababa is <u>not</u> OK. What must we remember? What do the states represent? They represent whatever we need to remember.

<u>Example</u> $\Sigma = \{0, 1\}$ L = { strings which when interpreted as binary numbers are divisible by 3 : L = {11, ε, 110, 1001, 1100, ..} but not 10 or 1 or 001. Interpret ε as 0



We read left to right

$\underbrace{\sqcup\sqcup\sqcup\sqcup\sqcup}_{\text{part of the word seen so far}}$ $\sqcup$ ← new symbol

If x is the value so far then when we read another 0 we get $2x$; if we read 1 we get $2x+1$. We only need to do the arithmetic mod 3 : s_0 means $x \equiv 0 \pmod 3$ s_1 means $x \equiv 1 \pmod 3$ & s_2 means $x \equiv 2 \pmod 3$.

$0 \times 2 = 0$ so $\delta(s_0, 0) = s_0$ $\quad 0 \times 2 + 1 = 1$ so $\delta(s_0, 1) = s_1$.

<u>Lesson</u> The states should mean something, they encode your finite memory.

Prop Any machine to recognize L must have at least 3 states.

Proof Suppose M has only 2 states one must be an accept state and the other one a non-accept state. Now consider the strings 100, 101 & 110. The string 110 must go to the accept state and 100 & 101 must both go the other state call it B. Once the machine is in B it does not matter how it got there the subsequent actions are determined. So 1001 & 1011 must go to the same state but 1001 should be accepted & 1011 must be rejected. Thus 1001 & 1011 cannot reach the same state ⊗.

General strategy: If we want to prove that there must be at least n states. Find n strings such that they all have to go to different states. For each pair say u, v show that there is some string x s.t. ux is accepted & vx is rejected. This proves that u, v cannot reach the same state. Do this for all pairs.

Some math Given $M = (S, s_0, \delta, F)$ we define

$\delta^* : S \times \Sigma^* \to S$ by induction

$$\delta^*(s, \varepsilon) = s$$

$$\delta^*(s, ax) = \delta^*(\delta(s,a), x)$$

$$L_M = \{x \mid \delta^*(s_0, x) \in F\}$$

A set with a binary associative operation $\bullet$ and an identity ε : $(M, \bullet, \varepsilon)$ is called a monoid.