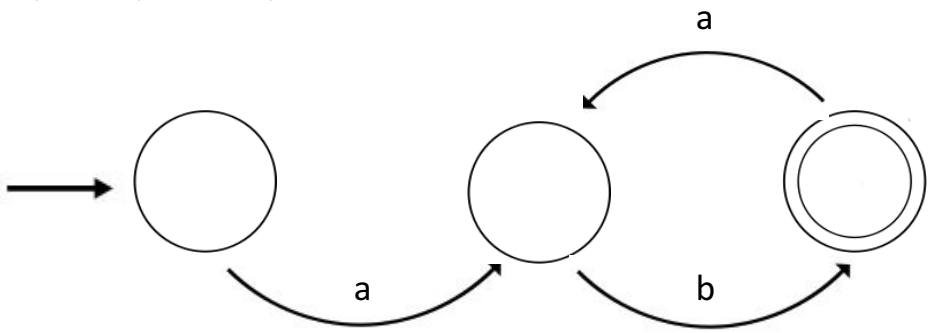
Scribe: Jocelyn Schulz *Date: 9/22/2011*

We started out designing finite state automatons for different sets of strings in the same way that we previously designed a finite state automaton for a noun phrase. A finite state automaton being able to recognize language means that everything that is good ends in a final state and everything that is bad will not.

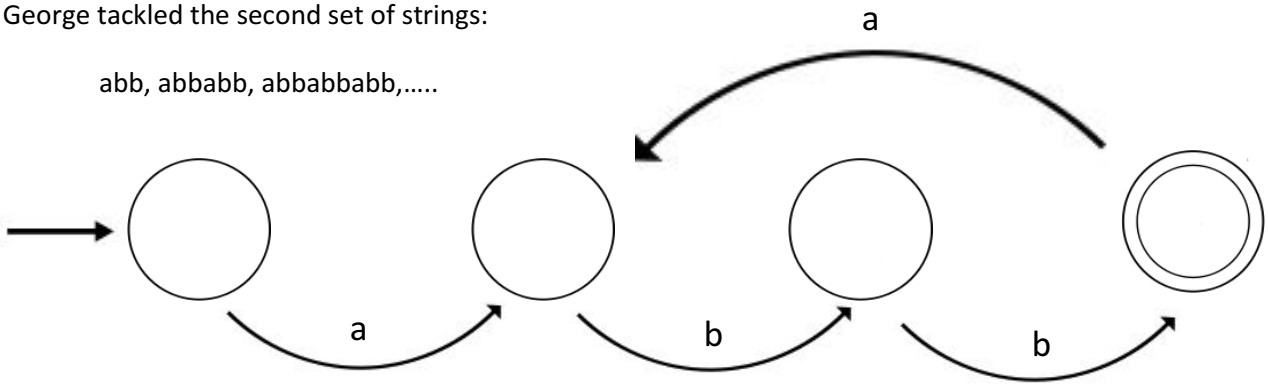
The first set of strings for which Sungwon was told to design a finite state automaton was:

ab, abab, ababab,ababbabab, …..



George tackled the second set of strings:

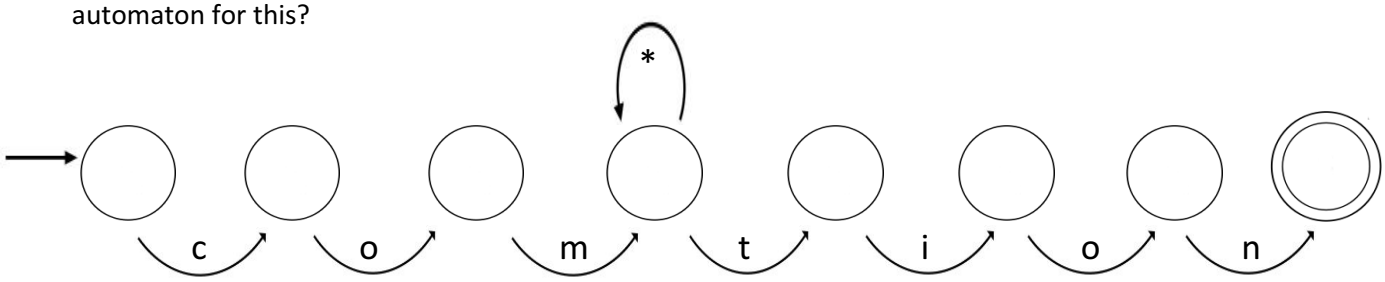
abb, abbabb, abbabbabb,…..



These types of finite state automaton are called **deterministic** because for any state, given the input, there's a clear path to take.

One way in which these are used in practice is with string searches.

For example, say you want any word that starts with 'com' has some number of letters (represented by an asterisk meaning 0 or more letters) and ends in 'tion'. How would you design the finite state automaton for this?



The problem here was, of course, what happens when you get to the asterisk. When the asterisk brings up a 't', which path would you follow from there? This machine is **non-deterministic** finite state automaton because there is at least one state from which, given an input, you could go to two or more possible states.

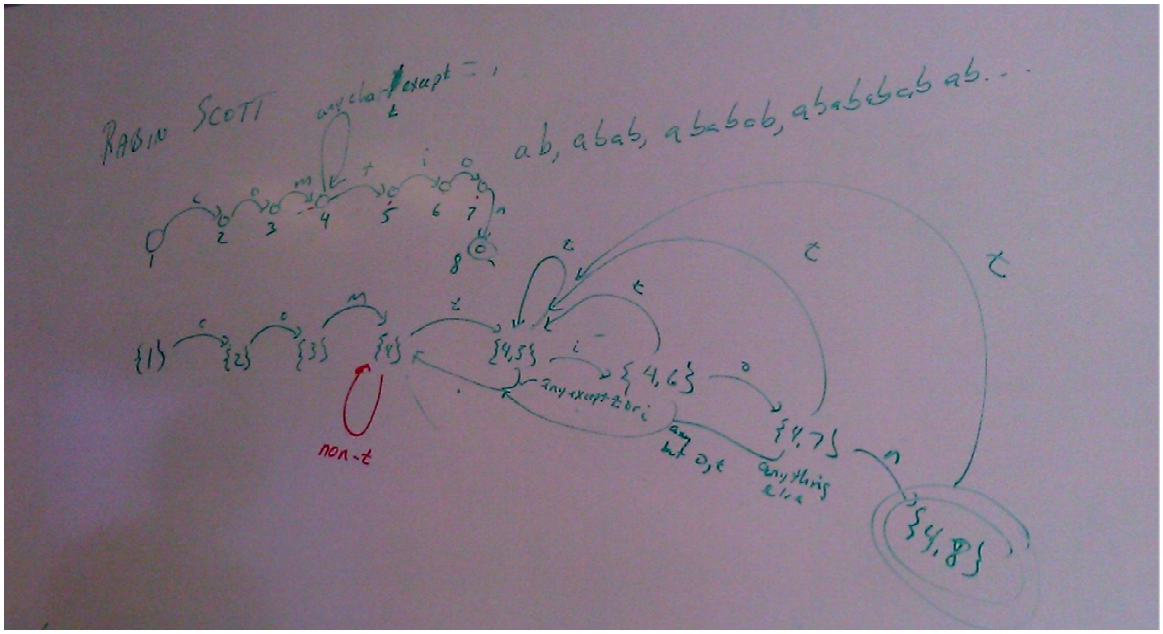
However, there is in fact a way to transform a non-deterministic machine into its kindly brother. This idea came about in the late '50s because of mathematical curiosity and because they wanted to do the same kind of thing. A non-deterministic finite state automaton was represented, but needed to be converted to a deterministic one because those are very easy to program. [Example of the relationship between math and computing]

The equivalence between the two kinds of finite state automata was introduced by Michael Rabin and Dana Scott.

Reminder: Look up the Rabin-Scott algorithm (Rabin-Scott powerset construction) which converts non-deterministic machines to deterministic ones

The actual conversion that we did in class used this kind of **subset construction**. Deterministic finite automata only keep track of their current state. In order for a non-deterministic finite automaton to be transformed into a DFA, it must mimic it. Instead of single states, however, it must keep track of a set of states. To do the conversion, you take the states from the NFA, make more states from them by asking yourself which state you would go to given a certain input and the states will then represent a set of states from what you had in the beginning. Then, you number them for clarification.

The solution is (hopefully) viewable here:



To finish off, we then reviewed two of Dr. Ecco's puzzles: Spacecraft Malfunction and Wrong Number, the solutions for which can be found in the back of the book and Lizzie gave her presentation on Chapter 2: Glenn Reeves and Adrian Stoica – Design for a Faraway Planet.