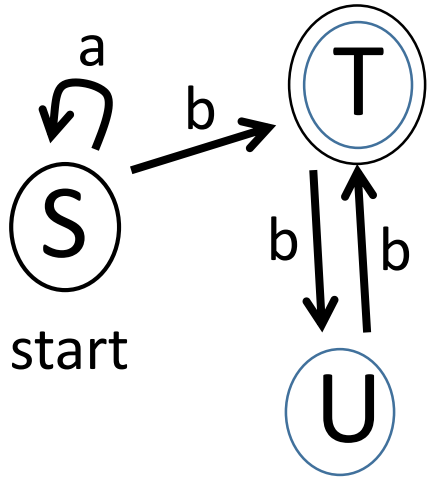
Finite Automata

See Sections 2.1 and 2.2

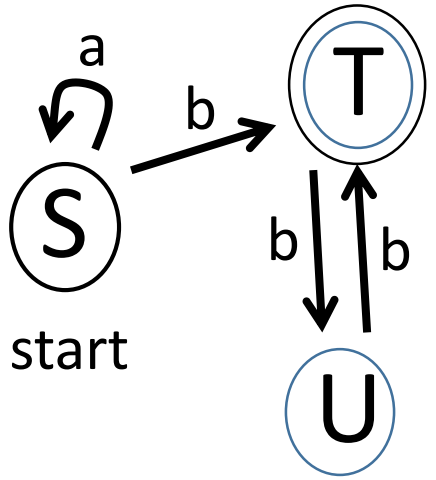
Let's start with an example:



Here you see labeled circles that are *states*, and labeled arrows that are *transitions*. One of the states is marked "start". One of the states has a double circle; this is a *terminal state* or *accept state*.

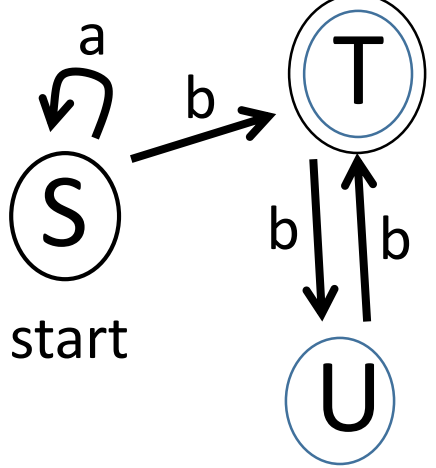
To process a string with this automaton, begin in the start state at the start of the string. Walk through the letters of the string, taking the transitions labeled with the letters. If you are in an accept state at the end of the string, accept it; otherwise reject it.

For example, on string "abbb" we go through the states SSTUT and end in an accept state, so we accept "abbb". On the other hand, with string "bb" we go through states STU and we do not accept.



With string "abab" we start in state S, stay in S on input 'a', go to state T on input 'b' and then have nowhere to go on input 'a'. Again, we do not accept the string. The only strings accepted are those for which the automaton consumes all of the letters in getting to an accept state.

The usual question for a finite automaton is "What is the language accepted by the automaton?"



The language accepted by this automaton is the set of strings with any number of a's followed by an odd number of b's.

In general, a "Deterministic Finite Automaton" or DFA is a quintuple $(Q, \Sigma, \delta, s, F)$ where

Q is a finite set of states

Σ is an alphabet of symbols

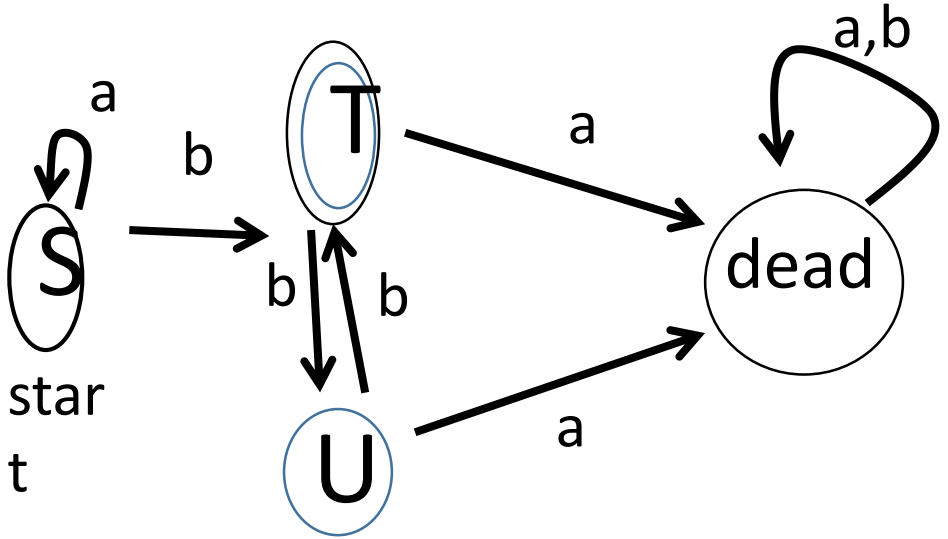
δ is a *transition function* whose inputs are a state and an element of Σ and whose output is a state. This is represented by the arrows in our diagrams.

s is one of the states in Q. This is our *start* state. Note that there is only one start state in an automaton.

F is the set of accept, or *final* states.

A DFA processes strings in Σ^* as follows: Let $w=a_0a_1...a_{n-1}$ be a string in Σ^*. Let q_0 be s (the start state), and for $i > 0$ let $q_i = \delta(q_{i-1},a_{i-1})$ The last state is q_n. If q_n is an element of F accept the string w.

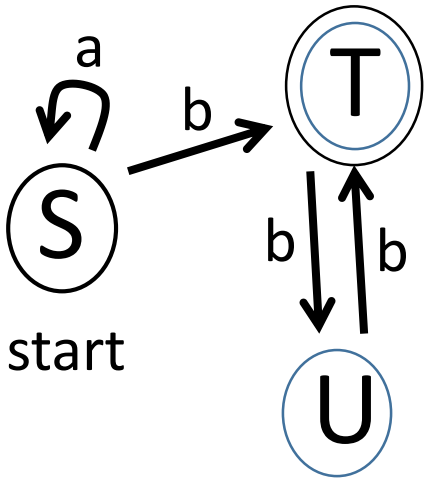
This definition assumes there is a transition from every state on every element of S. We can make any automaton fit this by adding for any missing transition a transition to a "dead" state from which there is no exit.



It is useful to use the *Language of Regular Expressions* to describe the strings accepted by an automaton:

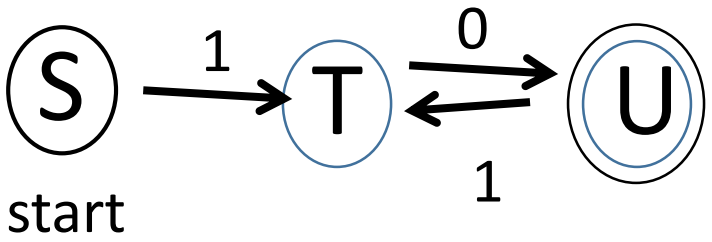
1. Any particular string w represents the language {w}
2. If expressions E and F represent languages $\mathcal{L}_1$ and $\mathcal{L}_2$ then expression E+F represents $\mathcal{L}_1 \cup \mathcal{L}_2$.
3. If expressions E and F represent languages $\mathcal{L}_1$ and $\mathcal{L}_2$ then expression EF represents the language of strings formed by concatenating a string from $\mathcal{L}_2$ onto the end of a string from $\mathcal{L}_1$.
4. If expression E represents language $\mathcal{L}$ then expression E^* represents the language of strings formed by concatenating 0 or more strings from $\mathcal{L}$ together.
5. If expression E represents language $\mathcal{L}$ then expression E^+ represents the language of strings formed by concatenating 1 or more strings from $\mathcal{L}$ together. $E^+=EE^*$

Example 1.



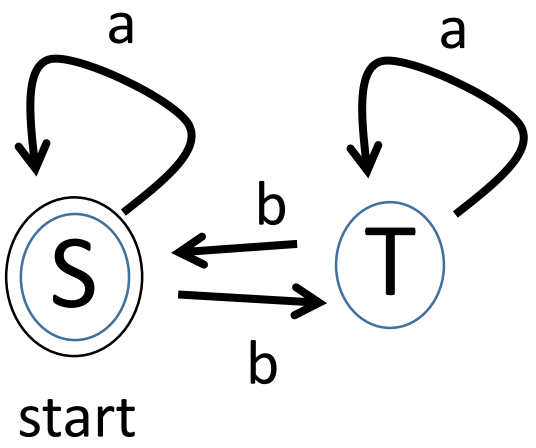
We said before that this accepts the language of any number of a's followed by an odd number of b's. This language is represented by the regular expression $a^*b(bb)^*$

Example 2



This accepts $10(10)^* = (10)^+$

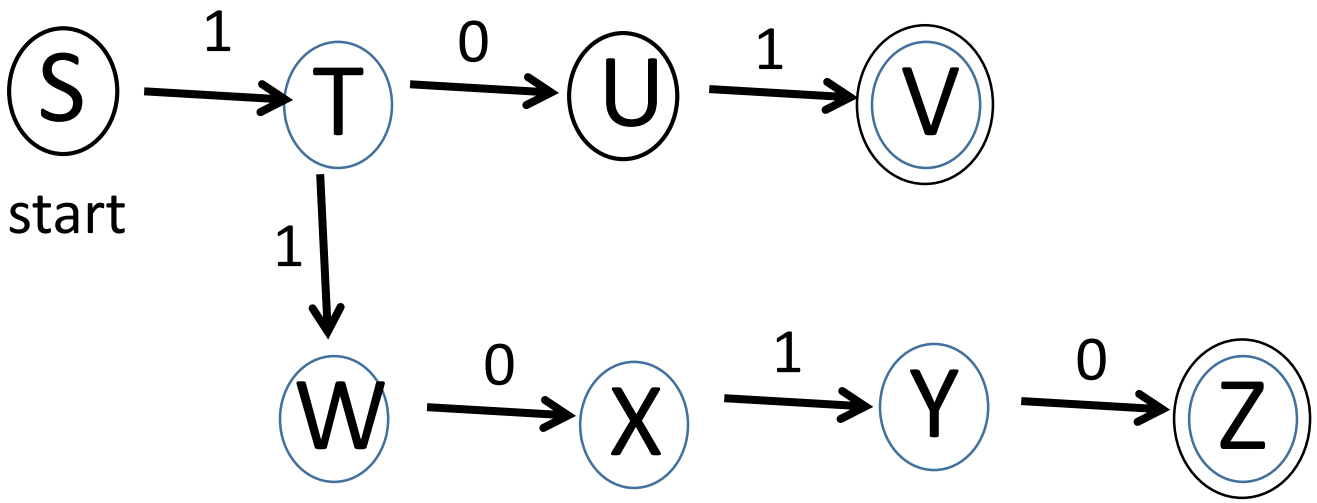
Example 3



This accepts strings with an even number of b's:

$(a^*(ba^*b)^*)^*$

Example 4: Find a DFA that accepts 101, 11010, and nothing else.



Note that by imitating this example you could make a DFA that accepts any particular finite language.

Note that it is easy to write a program that simulates a DFA's actions on a string. You might have one variable that represents the current state and a for loop that iterates through the letters of the string. For a simple DFA with only a few states you might have a big conditional statement that checks the various cases: if you are in state foo with input letter bar change the state to fee. For a more general automaton represent the transition function by a 2D table whose rows are indexed by the states and whose columns are indexed by the alphabet letters. Table[foo][bar] is the state to transition to if you are in state foo and see input letter bar.

Definition: We say that a language is *regular* if it is the language accepted by some DFA.

For examples:

- Σ^* is regular for any finite alphabet Σ.
- Every finite language is regular.
- 10^*1 is regular.
- Strings of 0's and 1's with an even number of 1's is regular.

Which of the following are regular? For each let the alphabet be {0,1}
Note that we can show a set *is* regular by producing the DFA that accepts it. We don't have a way to show a set isn't regular. We will.

a) Strings of length 2? yep
b) Strings of even length? yep
c) Strings of prime length?
d) Strings with the same number of 0's as 1's?
e) Strings with more 0's than 1's?
f) Strings that contain 010 as substrings? yep
g) Strings with no more than 2 0's? yep
h) Strings with at least 2 0's? yep
i) Strings that are palindromes?
j) Strings that are palindromes of length 6 or less? yep

Of course, we would like some way to characterize the languages that are regular so we could determine if a language is *not* regular.