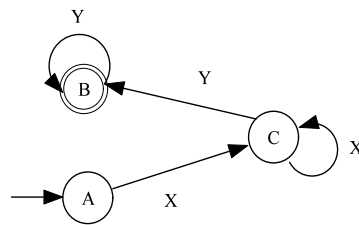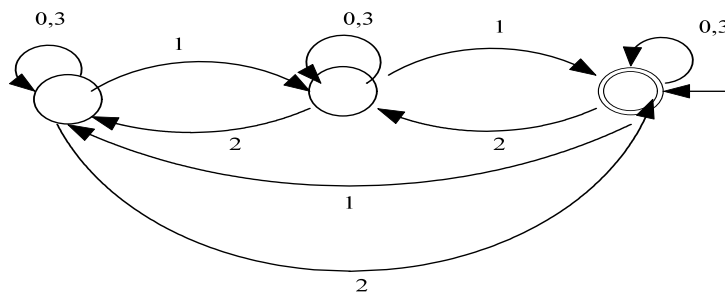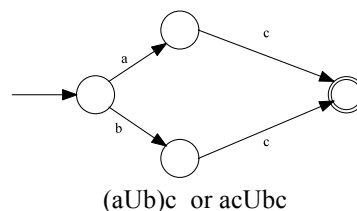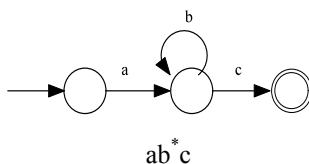# Regular Expressions and FSAs

A Finite State Automation (FSA) has four components: an input alphabet (those letters or strings which are legal inputs); a set of transition rules to advance from state to state (given a current state and an element of the input alphabet, what is the next state); a unique start state; and one or more final states. We can draw the FSA, as shown below, by representing each state as a circular node; the final state as a double circle; the start state as the only node with an incoming arrow; and the transition rules by the strings on the edges connecting the nodes. When labels are assigned to states, they appear inside the circle representing the state.

If there is a path from the start state to a final state by which an input string can be parsed, then the input string is said to be "accepted" by the FSA. The FSA above will accept strings composed of one or more x's followed by one or more y's (e.g., xy, xxy, $xxxyy$, $xyyy$). A more complicated example is given below. The input alphabet is the digits 0, 1, 2 and 3. This FSA accepts those strings whose base 4 value is a multiple of 3. It does this by summing the value of the digits: when in the left node, the running sum of the digits has a value, modulo 3, of "1"; in the middle node, "2", and in the right node, "0". A base four number, like a base ten number, is a multiple of 3 only when its digits sum to a multiple of 3.

Just like Boolean Algebra is a convenient algebraic representation of Digital Electronic Circuits, a regular expression is an algebraic representation of an FSA. For example, the regular expression corresponding to the first FSA given above is xx^*yy^*. The regular expression for the second FSA is extremely complex! The following simple FSAs and REs illustrate the correspondence:

ab^*c

$(aUb)c$ or $acUbc$

The rules for forming a regular expression (RE) are as follows:

        [1] The null string (λ) is a RE.
        [2] If the string a is in the input alphabet, then it is a RE.
        [3] if the strings a and b are both REs, then so are the strings built up using the following rules:

                [3a] CONCATENATION. "*ab*" (*a* followed by *b*).
                [3b] UNION. "*a***U***b*" (*a* or *b*).
                [3c] CLOSURE. "a^*" (*a* repeated zero or more times).

If we have a regular expression, then we can mechanically build an FSA to accept the strings which are generated by the regular expression. Conversely, if we have an FSA, we can mechanically develop a regular expression which will describe the strings which can be parsed by the FSA. For a given FSA or regular expression, there are many others which are equivalent to it. A "most simplified" regular expression or FSA is not always well defined.

Identities for regular expressions appear below. The order of precedence for regular expression operators is: Kleene Star, concatenation; and then union. The Kleene Star binds from right to left; concatenation and union both bind from left to right.

Typical problems in the category will include: translate an FSA to/from a regular expression; simplify an FSA or regular expression (possibly to minimize the number of states or transitions); create an FSA to accept certain types of strings.
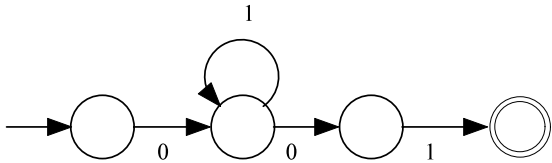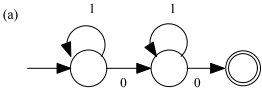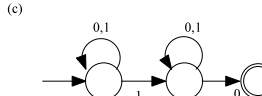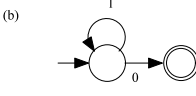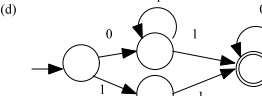
References
FSAs are usually covered in books on compilers. Sedgewick's *Algorithms* covers this topic rather nicely in the context of Pattern Matching and Parsing (Chapters 20 and 21).

Basic Identities

1. $(a^*)^* = a^*$
2. $aa^* = a^*a$
3. $aa^*\mathbf{U}\lambda = a^*$
4. $a(b\mathbf{U}c) = ab\mathbf{U}ac$
5. $a(ba)^* = (ab)^*a$
6. $(a\mathbf{U}b)^* = (a^*\mathbf{U}b^*)^*$
7. $(a\mathbf{U}b)^* = (a^*b^*)^*$
8. $(a\mathbf{U}b)^* = a^*(ba^*)^*$

Sample Problems

Find a simplified regular expression for the following FSA: 	The expression 01*01 is read directly from the FSA. It is in its most simplified form.

List all the following FSAs which represent 1*01*0.
 	
	Only choice (a) is correct: the other FSAs correspond to the following regular expressions:
	  (b) 1*0
	  (c) (0**U**1)*1(0**U**1)*0
	  (d) 01*10***U**10*10*
	Note that choices (c) and (d) can be rewritten using various identities. For example, and equivalent regular expression corresponding to FSA (d) is
	(01***U**10*)10*.

Which, if any, of the following Regular Expressions are equivalent?
  A. (a**U**b)(ab*)(b***U**a)
  B. (aab***U**bab*)a
  C. aab***U**bab***U**aaba**U**bab*a
  D. aab***U**bab***U**aab*a**U**bab*a
  E. a***U**b*
	On first inspection, **B** can be discarded because it is the only RE whose strings must end in an *a*. **D** can also be discarded since it is the only RE that can accept a null string. **C** and **D** are not equal by inspection. After expanding **A**, we must compare it to **C** and **D**. It is equal to **D**, but not to **C**.