[ext](http://lambda.uta.edu/cse5317/notes/node10.html) [p](http://lambda.uta.edu/cse5317/notes/node6.html) [revious](http://lambda.uta.edu/cse5317/notes/node8.html) [ontents](http://lambda.uta.edu/cse5317/notes/node1.html)   
**Next:** [2.4 Case Study: The](http://lambda.uta.edu/cse5317/notes/node10.html) **Up:** [2 Lexical Analysis](http://lambda.uta.edu/cse5317/notes/node6.html) **Previous:** [2.2 Deterministic Finite Automata](http://lambda.uta.edu/cse5317/notes/node8.html)   **[Contents](http://lambda.uta.edu/cse5317/notes/node1.html)** 

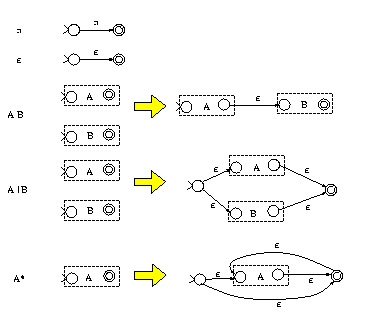
**2.3 Converting a Regular Expression into a Deterministic Finite Automaton**

The task of a scanner generator, such as JLex, is to generate the transition tables or to synthesize the scanner program given a scanner specification (in the form of a set of REs). So it needs to convert REs into a single DFA. This is accomplished in two steps: first it converts REs into a non-deterministic finite automaton (NFA) and then it converts the NFA into a DFA.

An NFA is similar to a DFA but it also permits multiple transitions over the same character and transitions over  \varepsilon$. In the case of multiple transitions from a state over the same character, when we are at this state and we read this character, we have more than one choice; the NFA succeeds if at least one of these choices succeeds. The  \varepsilon$ transition doesn't consume any input characters, so you may jump to another state for free.

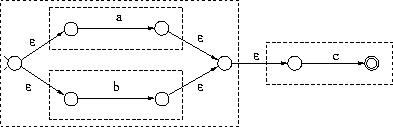
Clearly DFAs are a subset of NFAs. But it turns out that DFAs and NFAs have the same expressive power. The problem is that when converting a NFA to a DFA we may get an exponential blowup in the number of states.

We will first learn how to convert a RE into a NFA. This is the easy part. There are only 5 rules, one for each type of RE:



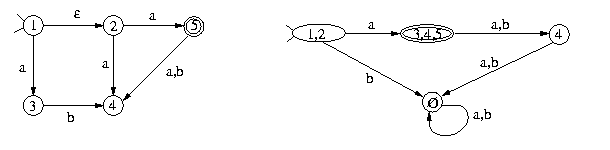
As it can been shown inductively, the above rules construct NFAs with only one final state. For example, the third rule indicates that, to construct the NFA for the RE *AB*, we construct the NFAs for *A* and *B*, which are represented as two boxes with one start state and one final state for each box. Then the NFA for *AB* is constructed by connecting the final state of *A* to the start state of *B* using an empty transition.

For example, the RE (*a*| *b*)*c* is mapped to the following NFA:



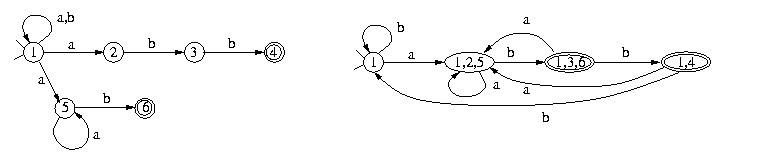
The next step is to convert a NFA to a DFA (called *subset construction*). Suppose that you assign a number to each NFA state. The DFA states generated by subset construction have sets of numbers, instead of just one number. For example, a DFA state may have been assigned the set {5, 6, 8}. This indicates that arriving to the state labeled {5, 6, 8} in the DFA is the same as arriving to the state 5, the state 6, or the state 8 in the NFA when parsing the same input. (Recall that a particular input sequence when parsed by a DFA, leads to a unique state, while when parsed by a NFA it may lead to multiple states.)

First we need to handle transitions that lead to other states for free (without consuming any input). These are the  \varepsilon$ transitions. We define the *closure* of a NFA node as the set of all the nodes reachable by this node using zero, one, or more  \varepsilon$ transitions. For example, The closure of node 1 in the left figure below



is the set {1, 2}. The start state of the constructed DFA is labeled by the closure of the NFA start state. For every DFA state labeled by some set {*s*1,..., *s*n} and for every character *c* in the language alphabet, you find all the states reachable by *s*1, *s*2, ..., or *s*n using *c* arrows and you union together the closures of these nodes. If this set is not the label of any other node in the DFA constructed so far, you create a new DFA node with this label. For example, node {1, 2} in the DFA above has an arrow to a {3, 4, 5} for the character *a* since the NFA node 3 can be reached by 1 on *a* and nodes 4 and 5 can be reached by 2. The *b* arrow for node {1, 2} goes to the error node which is associated with an empty set of NFA nodes.

The following NFA recognizes (*a*| *b*)\*(*abb* | *a*+*b*), even though it wasn't constructed with the above RE-to-NFA rules. It has the following DFA:



[ext](http://lambda.uta.edu/cse5317/notes/node10.html) [p](http://lambda.uta.edu/cse5317/notes/node6.html) [revious](http://lambda.uta.edu/cse5317/notes/node8.html) [ontents](http://lambda.uta.edu/cse5317/notes/node1.html)   
**Next:** [2.4 Case Study: The](http://lambda.uta.edu/cse5317/notes/node10.html) **Up:** [2 Lexical Analysis](http://lambda.uta.edu/cse5317/notes/node6.html) **Previous:** [2.2 Deterministic Finite Automata](http://lambda.uta.edu/cse5317/notes/node8.html)   **[Contents](http://lambda.uta.edu/cse5317/notes/node1.html)**

*fegaras 2012-01-10*