Converting Regular Expressions to Discrete Finite Automata: A Tutorial

David Christiansen

2013-01-03

This is a tutorial on how to convert regular expressions to nondeterministic finite automata (NFA) and how to convert these to deterministic finite automata. It's meant to be straightforward and easy to follow, rather than worrying about every technical detail. Please see Torben Mogensen's book for the nitty-gritty. In this tutorial, we write concrete regular expressions in `typewriter font` and variables ranging over regular expressions with various forms of the letter r. The regular expression matching the empty string is written ε, pronounced "epsilon". We use α and β, pronounced "alpha" and "beta", to represent arbitrary symbols.

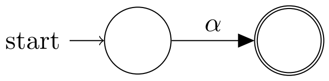
Converting Regular Expressions to NFAs

A regular expression can consist of the following constructions:

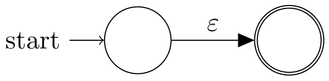
- It can match the empty string, represented by ε.
- It can match a symbol α from the alphabet of the language to be matched.
- It can match either of two regular expressions r_1 and r_2, written $r_1 | r_2$.
- It can match a sequence of two regular expressions r_1 and r_2, written $r_1 r_2$.
- It can match zero or more instances of a regular expression r, written $r*$.

We convert a regular expression to a nondeterministic finite automaton (NFA) by considering each case in the above definition. By definition, every automaton (whether NFA or DFA) has a *single* start state. An automaton may have more than one accepting state, but because of the way the rules below are constructed, the resulting NFAs will have exactly one accepting state. Even if this were not the case, it would be easy to create a unique accepting state by simply creating a new unique accepting state and inserting ε-transitions from the previous accepting states to the new one.

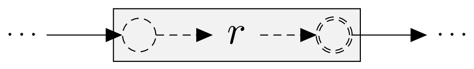
We begin with the base cases, that is, regular expressions α and ε. The symbol α is matched by an automaton that has a start state, an accepting state, and a transition on α from the start state to the accepting state:



Because we are creating an NFA, we can use the same construction to match the empty expression ε, just with an ε-transition to the accepting state rather than a symbol from the alphabet. In other words, the regular expression ε gives rise to the following automaton:

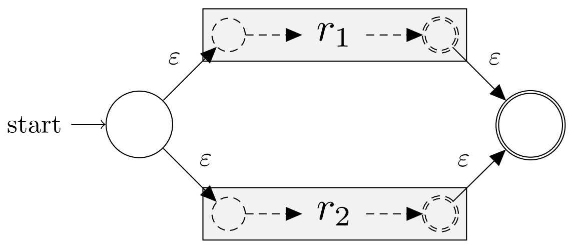


In the remaining cases, that is, the composite regular expressions, we need to represent the result of a recursive use of the conversion procedure. The result of converting some regular expression r to an NFA will be shown as follows:

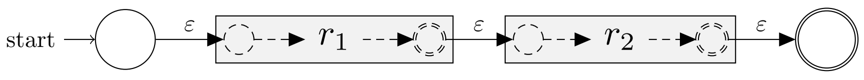


The leftmost state in the box represents the start state of r's automaton, and the accepting state to the right represent the accepting states of r's automaton.

A choice between r_1 and r_2, written $r_1 \,|\, r_2$, is constructed by creating a new start state with ε-transitions to the start states of r_1's and r_2's automata, and an accepting state with ε-transitions from their accepting states, as follows:

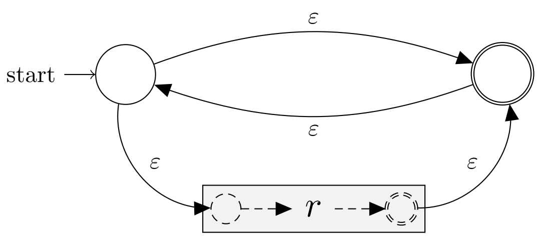


A sequence of two regular expressions r_1 and r_2, written r_1r_2, is converted to an NFA by simply attaching the accepting states of r_1's NFA to the initial state of r_2's NFA:



The NFA of the repeating expression $r*$ has an ε-transition from its start state to r's start state, and an ε-transition from r's accepting state to its accepting state. Additionally, there are ε-transitions from the start state to the accepting state and back again. This enables r to be skipped or to be repeated as many times as necessary.

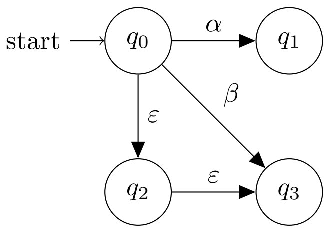
In picture form, the NFA corresponding to $r*$ is as follows:



Converting NFAs to DFAs

To convert an NFA to a DFA, we must find a way to remove all ε-transitions and to ensure that there is one transition per symbol in each state. We do this by constructing a DFA in which *each state corresponds to a set of some states* from the NFA. In the DFA, transitions from a state S_1 by some symbol α go to the state S_2 that consists of all the possible NFA-states that could be reached by α from some NFA state q contained in the present DFA state S_1. The resulting DFA "simulates" the given NFA in the sense that a single DFA-transition represents many simultaneous NFA-transitions.

The first concept we need is the ε-closure, pronounced "epsilon closure". The ε-closure of an NFA state q is the set containing q along with all states in the automaton that are reachable by any number of ε-transitions from q. In the following automaton, the ε-closures are given in the table to the right:



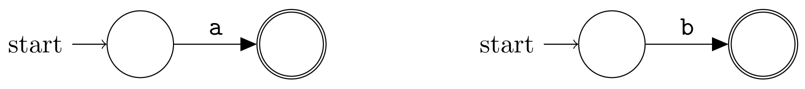
State	ε-closure
q_0	$\{q_0, q_2, q_3\}$
q_1	$\{q_1\}$
q_2	$\{q_2, q_3\}$
q_3	$\{q_3\}$

Likewise, we can define the ε-closure of a *set of states* to be the states reachable by ε-transitions from its members. In other words, this is the union of the ε-closures of its elements.

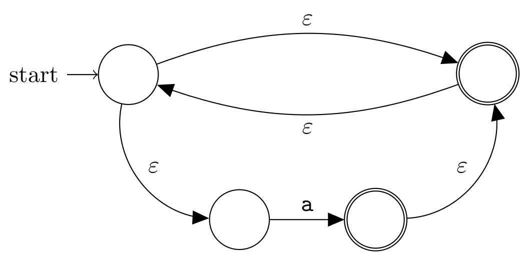
To convert our NFA to its DFA counterpart, we begin by taking the ε-closure of the start state q_0 of our NFA and constructing a new start state S_0 in our DFA corresponding to that ε-closure. Next, for each symbol α in our alphabet, we record the set of NFA states that we can reach from S_0 on that symbol. For each such set, we make a DFA state corresponding to its ε-closure, taking care to do this only once for each set. In the case two sets are equal, we simply reuse the existing DFA state that we already constructed. This process is then repeated for each of the new DFA states (that is, set of NFA states) until we run out of DFA states to process. Finally, every DFA state whose corresponding set of NFA states contains an accepting state is itself marked as an accepting state.

Example

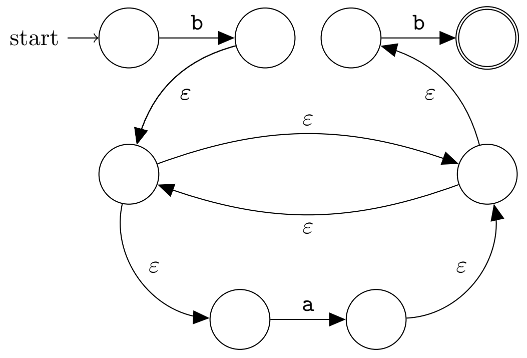
For this example, we'll convert the regular expression `ba*b` into a DFA. We being with simple automata matching `a` and `b`:



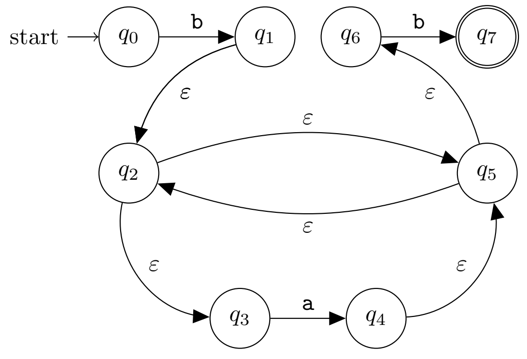
Next, we construct the automaton matching `a*` using the automaton for `a`:



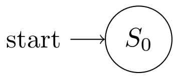
Finally, we attach our `b`-automaton to each end using the rule for sequencing:



To convert this NFA to a DFA, we begin by labeling the states so we can refer to them during the process:

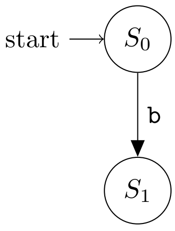


We begin by taking the ε-closure of the start state, q_0. As there are no epsilon transitions, we simply have the singleton set $\{q_0\}$. Our DFA is as follows:



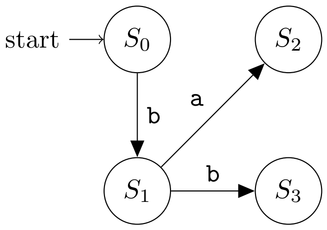
State	a	b	NFA States
S_0	?	?	$\{q_0\}$

The NFA state q_0 has no transitions on a. Therefore, we mark this table entry as an error. On a b, it has a transition to q_1. The ε-closure of q_1 is $\{q_1, q_2, q_3, q_5, q_6\}$. We assign this set to a new DFA state S_1, yielding the following automaton:



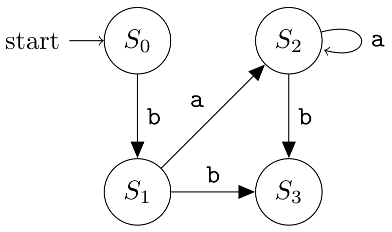
State	a	b	NFA States
S_0	Err	S_1	$\{q_0\}$
S_1	?	?	$\{q_1, q_2, q_3, q_5, q_6\}$

The NFA state q_3 has an a-transition to q_4, which is the only a-transition in S_1's NFA states. The ε-closure of q_4 is $\{q_2, q_3, q_4, q_5, q_6\}$. The NFA state q_6 has the only b-transition in S_1's NFA states, and it leads to q_7, whose ε-closure is simply $\{q_7\}$.



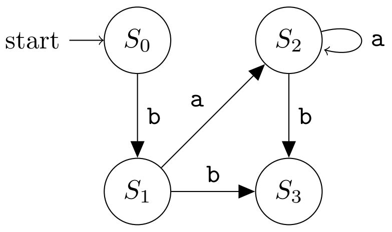
State	a	b	NFA States
S_0	Err	S_1	$\{q_0\}$
S_1	S_2	S_3	$\{q_1, q_2, q_3, q_5, q_6\}$
S_2	?	?	$\{q_2, q_3, q_4, q_5, q_6\}$
S_3	?	?	$\{q_7\}$

Analyzing S_2, we find an a-transition from the NFA state q_3 to the NFA state q_4. However, we already have a DFA state corresponding to q_4's ε-closure; namely, S_2 itself. Likewise, we have a b-transition to q_7, whose ε-closure is represented by the DFA state S_3.



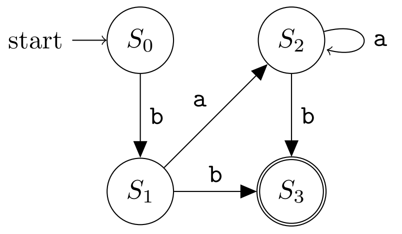
State	a	b	NFA States
S_0	Err	S_1	$\{q_0\}$
S_1	S_2	S_3	$\{q_1, q_2, q_3, q_5, q_6\}$
S_2	S_2	S_3	$\{q_2, q_3, q_4, q_5, q_6\}$
S_3	?	?	$\{q_7\}$

Finally, we examine S_3, corresponding to $\{q_7\}$. There are no outgoing transitions from q_7, so we mark these as errors:



State	a	b	NFA States
S_0	Err	S_1	$\{q_0\}$
S_1	S_2	S_3	$\{q_1, q_2, q_3, q_5, q_6\}$
S_2	S_2	S_3	$\{q_2, q_3, q_4, q_5, q_6\}$
S_3	Err	Err	$\{q_7\}$

Finally, we mark each DFA state (set of NFA states) as accepting if at least one of its member NFA states are accepting. In this case, only q_7 is accepting, which means that only S_3 is accepting.



State	`a`	`b`	Accepting
S_0	Err	S_1	No
S_1	S_2	S_3	No
S_2	S_2	S_3	No
S_3	Err	Err	Yes (from q_7)

The DFA is now fully constructed.

Acknowledgments

I would like to thank Janus Varmarken for correcting an error in the first version of this tutorial.