8 Equivalence of Regular Expressions and Finite-State Machines

BIG IDEA: Sometimes, two very different definitions define the same thing.

We have defined the class of regular language using DFAs. We have also defined regular expressions. Finite-state machines and regular expressions could not be more different from one another. Amazingly, the class of languages that can be described by regular expressions is exactly the class of regular languages (the ones solvable by DFAs)! We will demonstrate the following two lemmas. (A lemma is a theorem that is proved as a step in proving a more important theorem.)

Lemma 8.1. If e is a regular expression then $L(e)$ is a regular language.

Lemma 8.2. If A is a regular language then there exists a regular expression e so that $L(e) = A$.

The following theorem follows immediately from the two lemmas.

Theorem 8.3. Language A is regular (solvable by a DFA) if and only if there exists a regular expression that describes language A.

Proofs of Lemmas 8.1 and 8.2 are long, and they involve defining new types of finite-state machines that are of an intermediate nature between regular expressions and DFAs.

8.1 Nondeterministic Finite-State Machines

Like a DFA, a *nondeterministic finite automaton*, or NFA, is a 5-tuple (Σ, Q, q_0, F, δ_N). An NFA differs from a DFA in the following two ways.

1. For each state q and each symbol $c \in \Sigma$, instead of giving a single state, the transition function $\delta_N(q, c)$ of gives a *set of states* that can

be reached from q upon reading symbol c. For example, there can be several transitions from state 2 to other states, or there can be no transitions from state 2 to other states.

2. An NFA accepts string s just when there exists a path from the start state q_0 to an accepting state (a member of F), where the symbols along the path are labeled, in sequence, by the symbols in string s.

Figure 8.1 shows the transition diagram of an NFA that accepts all strings over alphabet $\{a, b\}$ that end on ab.

8.1.1 The Subset Construction

Theorem 8.4. For every NFA M there is an equivalent DFA M'. The two are equivalent in the sense that they accept the same language.

Proof Sketch. The proof is an algorithm, called the *subset construction*, that converts an NFA into an equivalent DFA. The idea is simple: make the DFA keep track of all states that the NFA could possibly be in. So each state of the DFA is a set of states of the NFA. Here are the main ideas of the subset construction.

1. If q_0 is the start state of the NFA, then the start state of the DFA is $\{q_0\}$. That is, the DFA starts out in a state where it can only be in state q_0.

2. A state of the DFA is an accepting state provided it contains at least one accepting state of the NFA.

3. Suppose that δ_N is the transition function of the NFA. The DFA has a transition from set s to set t labeled by symbol c provided

$$t = \bigcup_{q \in s} \delta_N(q, c).$$

Typically, not all of the sets of states are accessible from the start state. Figure 8.2 shows the DFA that is obtained from the NFA in Figure 8.1 by the subset construction. Inaccessible states are omitted.

$\Diamond$

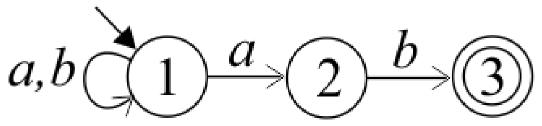

Figure 8.1. Transition diagram of an NFA that recognizes the set of all strings over alphabet $\{a, b\}$ that end on ab. For example, it accepts "ab", "$bbaab$" and "$aaaaaab$".
Notice that there are two transitions out of state 1 labeled by symbol a.
There are no transitions leaving state 3.
To run an NFA, start in state q_0 and follow transitions, writing the symbol of each transition as you follow it. If it is possible to write down string s and stop at an accepting state, then s is in the language of the NFA.

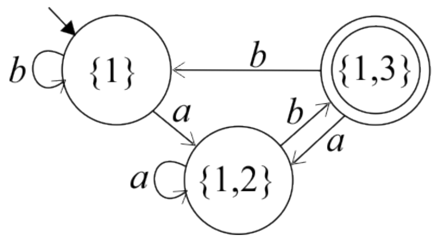

Figure 8.2. The DFA that is obtained from the NFA in Figure 8.1 using the subset construction. Each state of the DFA is a set of states of the NFA.

To build the DFA, start by creating start state $\{q_0\}$. Then create new states as they are needed.

For example, this DFA has a transition from state {1,2} to state {1,3} labeled b because there is a transition in the DFA from 1 to 1 labeled b and there is also a transition from 2 to 3 labeled b.

NFAs with ε-Transitions

An NFA with epsilon-transitions (an *NFA$_\varepsilon$*) is like an NFA, but it allows transitions labeled ε, called ε-transitions. You follow an ε-transition without reading a symbol.

It is easy to show that ε-transitions are not essential; you can get rid of them. That is, the following theorem is true. Its proof is left as an exercise. (Just add new non-epsilon-transitions that allow an NFA to reach all the same states as a particular NFA$_\varepsilon$.)

Theorem 8.5. For every NFA$_\varepsilon$ M, there is an equivalent NFA M' (without ε-transitions).

$\diamondsuit$

There is an important property of NFAs with ε-transitions. You can convert any NFA$_\varepsilon$ to an equivalent one with only one accepting state. That is easy! Just add a new state (to be the sole accepting state), add an epsilon transition from each accepting state to the new accepting state, and finally make all of the states except the new accepting state rejecting states.

Converting a Regular Expression to an NFA with ε-Transitions

Now we are ready to show how to convert a regular expression to a DFA. The idea is to convert the regular expression to an NFA$_\varepsilon$, then to convert that to an ordinary NFA, then to convert the NFA to a DFA using the subset construction.

Theorem 8.6. For every regular expression, there is an equivalent NFA$_\varepsilon$.

Proof. The proof is by induction on the length of a regular expression. Refer to the definition of a regular language in Section 7. A regular expression has one of five different forms: $\emptyset$, c (where c is a symbol), $A \cup B$, AB and A^*.

It is obvious that there is an NFA for an empty set and for every singleton set that contains a string of length 1. Also, we already know that the set of regular language is closed under union; having shown that the languages of regular expressions A and B are regular, we can conclude that the language of $A \cup B$ is also regular by that closure result.

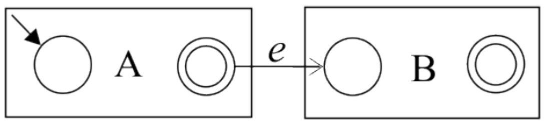

Figure 8.3. The regular languages are closed under concatenation. Suppose A and B are regular languages. Get an NFA$_\varepsilon$ for each of A and B, and ensure that the NFA$_\varepsilon$ has exactly one accepting state. Build an NFA$_\varepsilon$ as shown in the above diagram, connecting the NFA$_\varepsilon$ for A with that for B. (The ε-transition is labeled e.) Make the accepting state of A nonaccepting, and make only the start state of A be the start state of the combined machines.

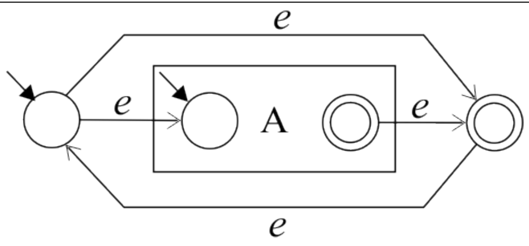

Figure 8.4. The regular languages are closed under Kleene closure. Suppose that A is a regular language. Get an NFA$_\varepsilon$ for A, and assume that it has exactly one accepting state. Build an NFA$_\varepsilon$ for A^* as shown in the diagram above.

All we need to do is to show that the regular languages are closed under concatenation and Kleene closure.

1. Suppose that A and B are regular languages. Then the concatenation AB is also regular. See Figure 8.3.

2. Suppose that A is regular language. Then the Kleene closure A^* of A is also regular. See Figure 8.4.

$\diamondsuit$

Now we are ready to prove Lemma 8.1.

Lemma 8.1. If e is a regular expression then $L(e)$ is a regular language.

Proof. Convert regular expression e to an NFA$_\varepsilon$, then to an NFA using Theorem 8.5, an finally to a DFA using the subset construction.

$\diamondsuit$

8.2 Converting a DFA to a Regular Expression

Now that we have shown how to convert a regular expression to an equivalent DFA (in a few steps), we need to show how to convert a DFA into an equivalent regular expression. For that, we need yet another type of finite-state machine.

A *generalized finite-state machine* (a GFA) has each transition labeled by a regular expression. The idea is that you can follow a transition while reading any member of the regular expression that labels the transition.

Theorem 8.7. For every DFA M there is an equivalent regular expression e.

Proof. Start with a DFA. It is a special case of an NFA$_\varepsilon$ that happens not to have any ε-transitions. Add a new accepting state and ε-transitions from all former accepting states so that there is exactly one accepting state. An NFA$_\varepsilon$ is a special case of a GFA, and we start with that GFA. (Replace ε by regular expression $\emptyset^*$.)

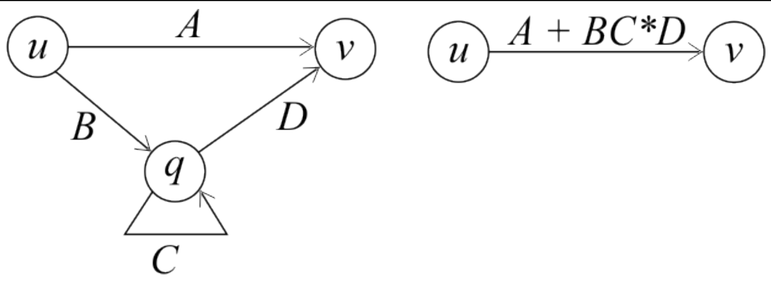

Figure 8.5. Choose a state q to remove that is not the start state and not the accepting state. First modify the transitions between states u and v for all pairs of states (u, v) where $u \neq q$ and $v \neq q$. The left-hand diagram shows transitions from u to q, q to itself and q to v. The right-hand diagram shows the new transition from u to v. Notice that the new regular expression labeling the transition from u to v makes it unnecessary to go through q.

Transitions must be modified for every pair of states (u, v) where $u \neq q$ and $v \neq q$, including the case where $u = v$.

If there is no transition between any of the shown states, add a transition labeled $\emptyset$ between them. Such a transition can never be taken. If you prefer, you can think of special modifications to make when some of the transitions are missing.

Now the idea is to remove states from the GFA one at a time, but not to remove the start or accepting state. Figure 8.5 shows how to remove state q. It is not the most efficient way to do that, but it gets the job done.

Eventually, you reach a GFA that has only two states, a start state and an accepting state, as shown in Figure 8.6. It is easy to convert that GFA into a regular expression.

$\diamondsuit$

Lemma 8.2. If A is a regular language then there exists a regular expression e so that $L(e) = A$.

Proof. That is immediate from Theorem 8.7.

$\diamondsuit$

prev **next**

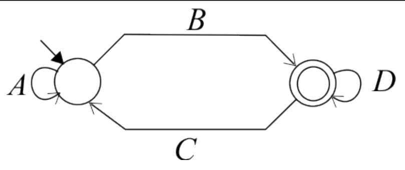

Figure 8.6. The final step in converting a GFA into a regular expression. The diagram shows the case where there are only two states left. Convert the GFA shown in the diagram into regular expression $A^*B(D \cup CA^*B)^*$. That captures all ways of getting from the start state to the accepting state.