14.1 Nondeterministic Finite Automata

14.1.1 Motivation for NFA’s

We're now ready to start on establishing the relationship between languages decided by finite-state machines (recognizable languages) and languages denoted by regular expressions (regular languages). Our central result is:

**Kleene’s Theorem:** A language is recognizable if and only if it is regular. In particular, we can take a DFA deciding a language and construct a regular expression denoting it, or vice versa.

Why would we want to prove something like this? The mathematician’s answer is that if we find the same class of mathematical objects coming up in two places, we must be on to something — this class of languages is probably important and useful in a variety of contexts. More simply, proving that the two classes are the same also proves that everything we know about one class is true of the other:

- The complement of a recognizable language is clearly recognizable, because we can turn its DFA into a DFA for the complement by switching the final and non-final states. It’s not obvious how to turn a regular expression into a regular expression for its complement, but the theorem says that this must be possible. (The same is true for the intersection of two recognizable languages — why?)

- The regular languages, on the other hand, are by definition closed under union, concatenation, and Kleene star. We proved in Chapter 5 that they are closed under reversal, one’s-complement, and prefix. The recognizable languages must by the theorem also be closed under all these operations, although for some of them this is difficult to prove directly.

- Given a regular expression and a string, it is not always easy to tell whether the string is in the language denoted by the expression. But if we have a string and a DFA, it is easy to run the DFA on the string and find out. Turning the regular expression into a DFA may be the best way to decide whether a string is in the language.

- If we are given a definition of a language in some form other than a DFA or a regular expression, it may be easier to turn it into one rather than the other. For example, the set of strings with an odd number of a’s and an odd number of b’s has an easy DFA and a horrible regular expression. But the Fibonacci language can be given by a recursive definition (“λ is in, 1 is in, if w is in so are w0 and w01”) from which the regular expression \((\lambda + 1)(0 + 01)^*\) just jumps out.

The problem with turning a regular expression into a DFA is that the regular expression is not **deterministic** and the DFA must be. “Deterministic” means that the same input must always produce the same output. But look at the problem of determining whether the strings \(aba\) and \(abc\)
are in the language \((a + ab + bc)^*\). Both strings are in the language, but for different reasons — the \(b\) in the middle is part of an \(ab\) in the first example and part of a \(bc\) in the second. When we’ve read \(ab\) and are waiting for more input, we have to be ready to interpret the \(b\) in either way based on what happens later. The regular expression doesn’t give us a decision algorithm directly, just a template into which we might or might not be able to fit a string.

### 14.1.2 NFA Definitions

This leads us to define the **non-deterministic finite automaton** or **NFA**, as a machine that has states, an initial state, final states, and input alphabet just like a DFA but has a **transition relation** rather than a transition function. Given a state \(s\) and a letter \(a\), there might be zero, one, or more than one state such that it is possible for the NFA to go from \(s\) to \(t\) in one move while reading \(a\). We define \(\Delta\) to be the set of triples \(\langle s, a, t \rangle\) such that this is true, and write “there is an \(a\)-transition from \(s\) to \(t\)” as either “\(\Delta(s, a, t)\)” or “\(\langle s, a, t \rangle \in \Delta\)”.

Like DFA’s, NFA’s can be represented by labeled directed multigraphs. We again have a node for each state, and we still represent \(a\)-transitions by arcs labeled with \(a\). But now any directed multigraph with arcs labeled by letters is a legal NFA, whereas we needed to have exactly one arc per letter out of each state in a DFA. (Actually we want to follow the usual convention that we don’t have parallel arcs (with the same beginning, end, and label), but they don’t really hurt much if they’re there.)

Here’s an example of an NFA for the regular language \((a + ab + bc)^*\) (Figure 14-14). We have a start state \(\iota\) that is also the only final state. Put an \(a\)-loop at \(\iota\), and make two other loops for the strings \(ab\) and \(bc\), using one new state for each. So our state set is \(\{\iota, p, q\}\) and our transition relation \(\Delta\) is

\[
\{\langle \iota, a, \iota \rangle, \langle \iota, a, p \rangle, \langle p, b, \iota \rangle, \langle \iota, b, q \rangle, \langle q, c, \iota \rangle\}\}
\]

This isn’t a DFA because of the two possibilities for reading an \(a\) from state \(\iota\), and the several situations in which there is no move available. But pretty clearly, you can have a path from \(\iota\) to itself if and only if that path takes some sequence of \(a\)-loops, \(ab\)-loops, and \(bc\)-loops, which is possible if and only if the labels on the path spell out a string in \((a + ab + bc)^*\).
We can define a relation $\Delta^*$ analogous to $\delta^*$ in a DFA, where $\langle s, w, t \rangle \in \Delta^*$ means that there is a path from $s$ to $t$, where the labels on the arrows in the path, read in order, spell out the word $w$. Thus $\Delta^*$ is a subset of $S \times \Sigma^* \times S$, while $\Delta$ is a subset of $S \times \Sigma \times S$. If $\iota$ is the initial state and $F$ the set of final states, we define the language of the machine to be exactly those strings for which $\Delta^*(\iota, w, f)$ holds for some final state $f \in F$.

What does this mean? A string is in the language if and only if it is possible to start at $\iota$, read the string, and wind up in a final state. There may be many paths, or no paths, labeled by $w$, and thus many states, or no states, $t$ such that $\Delta^*(\iota, w, t)$. What matters is whether at least one of these states is final.

It’s not necessarily easy to tell, given an NFA and a string, whether the string is in the language of the NFA. If it is, you can show that it is by showing a path from $\iota$ to a final state labeled by the string, but if it isn’t you have to argue somehow that no such path exists. Searching all such paths would in general be an exponentially long process, because multiple choices might exist at every step. (So changing an NFA into an equivalent DFA would a useful thing to be able to do.)

Just as we defined the function $\delta^*$ recursively from the function $\delta$, we can define $\Delta^*$ recursively from $\Delta$. We let $\Delta^*(s, \lambda, t) \leftrightarrow (s = t)$, and define $\Delta^*(s, wa, u)$ to be true if and only if for some state $t$, $\Delta^*(s, w, t)$ and $\Delta(t, a, u)$. This can be expressed as either a top-down or a bottom-up recursive definition, but note that the top-down definition gives a recursive algorithm that will run very badly in general. Given a seven-letter word $w$, it would test whether $\Delta^*(s, w, t)$ by recursively calling itself on a six-letter word for each state of the NFA, so that the seven-letter test would take far longer than a six-letter test (Could it help to use dynamic programming, as in Section 8.5?). We’ll see in Section 14.6 how to revise the algorithm to be more efficient, by turning the NFA into a DFA.

### 14.1.3 The Model of $\lambda$-NFAs

We’ve defined an NFA to have single-letter labels on each of its arrows. When we develop general methods to turn regular expressions into NFA’s, however, it will be convenient to also allow arrows labeled by $\lambda$, called “$\lambda$-moves”. We’ll call such an enhanced machine a $\lambda$-NFA\(^2\). So along with the choice of which arrow to follow when more than one has the same label, a $\lambda$-NFA also has the choice of whether to follow a $\lambda$-arrow, without reading any input at all.

How do we formally interpret $\lambda$-moves? A path in the NFA’s directed graph now has a sequence of labels that are either letters or $\lambda$’s, but we can interpret the sequence as a string by leaving out the $\lambda$’s and just reading the letters. Then we still say that $w \in L(M)$ for a $\lambda$-NFA $M$ if and only if there is some path, with labels concatenating together to $w$, from the start state to a final state. In machine terms, the automaton is allowed to “jump” along an $\lambda$-move without reading any input. We can still define $\Delta^*(s, w, t)$ as “there is a path labeled $w$ from $s$ to $t$” and define $w \in L(M)$ to be true if and only if the statement $\exists f \in F : \Delta^*(\iota, w, f)$ holds.

\(^2\)Different presentations of this material vary on the terminology here. Many define an “NFA” to be the same as our “$\lambda$-NFA”. 

14-3
Here is an example of a $\lambda$-NFA (Figure 14-15). We’ll have states \{1, 2, 3, 4\}, start state 1, single final state 4, input alphabet \{a, b\}, and transitions \(\langle 1, a, 4 \rangle\), \(\langle 1, b, 3 \rangle\), \(\langle 1, \lambda, 2 \rangle\), \(\langle 1, a, 2 \rangle\), \(\langle 2, \lambda, 4 \rangle\), \(\langle 2, b, 3 \rangle\), \(\langle 3, a, 4 \rangle\), and \(\langle 4, b, 4 \rangle\). What strings are in the language of this machine? Well, $bb$ is because we can take the $\lambda$-move from 1 to 2, another $\lambda$-move from 2 to 4, and then the $b$-loop at 4 twice. But $aa$ is not in the language because no path reads two $a$’s without a $b$ in between them. By checking out all the ways to get from 1 to 4, we get a language of $(\lambda + a + ba + aba)b^*$.

Unfortunately, defining $\Delta^*$ formally is now even harder for $\lambda$-NFA’s than it was for ordinary NFA’s. We still want to proceed by induction on all strings $w$ to define whether $\langle s, w, t \rangle \in \Delta^*$. But now even the base case $w = \lambda$ is a bit tricky. We could have $s = t$, $\Delta(s, \lambda, t)$, or even a path of $\lambda$ moves. (For example, in the $\lambda$-NFA above, $\Delta^*(1, \lambda, 4)$ is true.) We need to define $\Delta^*(s, \lambda, t)$ just as for the path relation in any graph:

- For any $s$, $\Delta^*(s, \lambda, s)$.
- If $\Delta^*(s, \lambda, t)$ and $\Delta(t, \lambda, u)$, then $\Delta^*(s, \lambda, u)$.
- If “$\Delta^*(s, \lambda, t)$” cannot be derived by these two rules, then it is not true.

Now we can define the $\Delta^*$ relation for general strings. How can $\Delta^*(s, wa, t)$ be true? We must have $\Delta^*(s, w, u)$ for some $u$, and then be able to get from $u$ to $t$ reading just an $a$. But this is possible if and only if for any states $x$ and $y$, $\Delta^*(u, \lambda, x)$, $\Delta(x, a, y)$, and $\Delta^*(y, \lambda, t)$. Again, we could write recursive code to check this from this definition, but it would be hideously slow.

Our remaining two tasks, then, will be to convert $\lambda$-NFA’s to ordinary NFA’s and then ordinary NFA’s to DFA’s. We’ll start with the latter, because it’s far simpler.

14.1.4 Exercises

E14.5.1 Prove that given any $\lambda$-NFA $N$, there is another $\lambda$-NFA that is equivalent to $N$ and has exactly one final state.

---

3If we had defined an “NFA” to be what we are calling a “$\lambda$-NFA”, we would be forced to do both these tasks at once – we prefer to isolate the simpler construction that turns our NFA’s into DFA’s.
E14.5.2 Define an **NFA with multiple start states** to be like an NFA except that it has a start state set rather than a single start state. A string \( w \) is in its language if and only if \( \Delta^*(s, w, t) \) for some start state \( s \) and some final state \( t \). Prove that if \( M \) is an NFA with multiple start states, there is a \( \lambda \)-NFA (with only one start state) equivalent to \( M \).

E14.5.3 How many different NFA’s are there with exactly two states? (Here “different” means “not identical”, not “inequivalent”.)

E14.5.4 Let \( M \) be the following NFA. The alphabet is \( \{a, b\} \), states are \( \iota \) and \( p \), \( \iota \) is the start state, both states are final, and

\[
\Delta = \{\langle \iota, a, p \rangle, \langle p, b, \iota \rangle\}.
\]

(a) Draw the labeled directed graph for this NFA.

(b) Give three examples of strings in \( L(N) \), and three examples of strings not in \( L(N) \). Justify your answers.

E14.5.5 Let \( N \) be the following NFA. The alphabet is \( \{a, b\} \). States are \( \{\iota, p, q, r, s\} \). The start state is \( \iota \). The final state set is \( \{p, s\} \). The relation \( \Delta \) is given by:

\[
\Delta = \{\langle \iota, a, p \rangle, \langle \iota, b, q \rangle, \langle \iota, b, r \rangle, \langle r, a, s \rangle, \langle s, b, p \rangle, \langle s, a, q \rangle, \langle s, a, r \rangle, \langle p, b, s \rangle, \langle r, b, q \rangle\}.
\]

(a) Draw the labeled directed graph for this NFA.

(b) Give three examples of strings in \( L(N) \), and three examples of strings not in \( L(N) \). Justify your answers.

(c) Prove that \( \Delta^*(\iota, aba, q) \). Are there any other states \( x \) such that \( \Delta^*(\iota, aba, x) \)? Justify your answer by listing all possible paths out of \( \iota \) labelled by \( aba \).

(d) It’s possible to get an equivalent NFA by deleting one of the states. Which one? Describe the new NFA \( N' \) obtained by the deletion and explain carefully why \( L(N) = L(N') \).

### 14.1.5 Problems

P14.5.1 Explain how to build an NFA \( M \) with \( L(M) = S^* \), where \( S \) is any finite set of strings. (More precisely, \( S \) is the regular expression denoting the finite set of strings.) Demonstrate your construction by building an NFA for \( S^* \) where \( S = aa + baab + ba + bbb \).

P14.5.2 Define a **multiple-letter NFA** to be one where transitions may be labeled with strings of two or more letters as well as by single letters. Explain how to replace a multiple-letter NFA with an equivalent ordinary NFA. (Hint: If \( \langle s, abc, t \rangle \) is a transition, add two states and new letter-transitions so that the NFA can still go from \( s \) to \( t \) while reading \( abc \), but can’t do anything that it couldn’t do before. Then delete the transition \( \langle s, abc, t \rangle \).)

P14.5.3 Let the NFA \( K \) (Figure 14-16) have alphabet \( \{a, b\} \), state set \( \{p, q, r\} \), start state \( p \), only final state \( r \), and transition relation

\[
\Delta = \{\langle p, a, q \rangle, \langle p, b, q \rangle, \langle p, a, r \rangle, \langle p, b, r \rangle, \langle q, a, p \rangle, \langle q, b, p \rangle, \langle q, a, r \rangle, \langle q, b, r \rangle, \langle r, a, p \rangle, \langle r, b, p \rangle, \langle r, a, q \rangle, \langle r, b, q \rangle\}.
\]

What is the language \( L(K) \)? (Consider the string \( \lambda \) carefully.) How many paths are there in \( K \) from \( p \) to \( r \) that are labeled by the string \( aaa \)? By the string \( aaaaaaaaaa \)? By the string \( abbabbaab \)?
P14.5.4 Let $N$ be an NFA with $k$ states. For every letter $a \in \Sigma$, define a $k$ by $k$ boolean matrix $M_a$ such that $M_a(i,j)$ is true if and only if $(i,a,j) \in \Delta$. Define a function $f$ from $\Sigma^*$ to the set of $k$ by $k$ boolean matrices by the rules $f(\lambda) = I$, $f(wa) = f(w)M_a$. Prove that $w \in L(N)$ if and only if there is a final state $f$ such that the $(i,f)$ entry of $f(w)$ is true.

P14.5.5 Consider the set of languages of the two-state NFA’s counted in Exercise 14.5.3. For each such NFA $N$, we can let the regular expressions $\alpha$, $\beta$, $\gamma$, and $\delta$ denote the set of letters on which $N$ may go in one step from $i$ to $i$, $i$ to $p$, $p$ to $i$, and $p$ to $p$ respectively. (For example, if $(p,a,p)$ and $(p,b,p)$ are both in $\Delta$, then $\delta = a + b$.) For each of the four possible final state sets of $N$, give a regular expression for $L(N)$ in terms of $\alpha$, $\beta$, $\gamma$, and $\delta$. Use this expression to find $L(N)$ for as many as possible of the two-state NFA’s. (For example, all the NFA’s with empty final set have an empty language, and this is a quarter of all the NFA’s.)