14.1 The Subset Construction: NFA’s into DFA’s

14.1.1 The Construction and an Example

Now that we’ve defined ordinary NFA’s and $\epsilon$-NFA’s, we can start on the process of converting them into DFA’s. Remember that this is part of the process of getting a DFA that decides the language denoted by a given regular expression, because we’ll later show how to change regular expressions into $\epsilon$-NFA’s. Note also that it is trivial to convert a DFA into an NFA or an NFA into an $\epsilon$-NFA, because an NFA is allowed to have exactly one transition per letter per state, and an $\epsilon$-NFA is allowed to have no $\epsilon$-moves.

We’ll convert ordinary NFA’s into DFA’s by a process known as the Subset Construction. Given an NFA $M$, we’ll make a DFA $M'$ whose states will represent sets of states of $M$. For any input word $w$, $\delta^*(\iota, w)$ will be the set of states that are reachable in $M$ by a path labeled $w$, that is, the set $\{s : \Delta^*(\iota, w, s)\}$. This will be enough information to tell whether $w \in L(M)$, because $w \in L(M)$ is defined to be true if and only if $\exists f : (f \in F) \land \Delta^*(\iota, w, f)$ and we can easily determine this if we know exactly which states are reachable.

We illustrate the process first with an example, the Fibonacci language $(0 + 01)^*$, which we can represent by an NFA $M$ (Figure 14-17), with state set $\{\iota, p\}$, initial state $\iota$, only final state $\iota$, input alphabet $\{0, 1\}$, and transitions $(\iota, 0, \iota), (\iota, 0, p)$, and $(p, 1, \iota)$. Just as in our example of the language $(a + ab + bc)^*$ from Section 14.5, there is a path labelled $w$ through this NFA from $\iota$ to $\iota$ if and only if $w$ is some concatenation of 0’s and 01’s.

$M'$’s states are to be the four possible subsets of the states of $M$: $\emptyset, \{\iota\}, \{p\}, \{\iota, p\}$. (In general if $M$ has $k$ states, there will be $2^k$ states in $M'$. But we often don’t have to consider all of them, as we’ll see below.) The start state of $M'$ will be the set of states that $M$ might be in after reading $\epsilon$, namely the set containing just $\iota$. (Note that this would be harder to figure out if $M$ were an $\epsilon$-NFA — we’re describing the process only for ordinary NFA’s now.) The final states of $M'$ will be those sets containing a final state of $M$, $\{\iota\}$ and $\{\iota, p\}$. Now we need only define the $\delta$ function for $M'$.

If we know where $M$ is, we know exactly the set of places it could go on one move while reading a given letter: $\delta(x, a) = \{y : \Delta(x, a, y)\}$. In our example, this lets us compute $\delta(\{\iota\}, 0) = \{\iota, p\}$, $\delta(\{\iota\}, 1) = \emptyset$, $\delta(\{p\}, 0) = \emptyset$, and $\delta(\{p\}, 1) = \{\iota\}$. But what about the other two states of $M'$? $\emptyset$ is easy — if you are nowhere at all, you aren’t going to get anywhere by reading a letter, so $\delta(\emptyset, a) = \emptyset$ for any $a$. (Viewed another way, if no paths at all are labeled by the string seen so far, no paths can be labeled by any extension of that string.) What about $\{\iota, p\}$? If there are paths labeled $w$
ending at either \( \iota \) or at \( p \), then the paths labeled \( wa \) could end in any state \( s \) such that \( \Delta(\iota, a, s) \) or \( \Delta(p, a, s) \). In general,

\[
\delta(S, a) = \bigcup_{x \in S} \{ y : \Delta(x, a, y) \}.
\]

You can compute this as

\[
\delta(S, a) = \bigcup_{x \in S} \delta(\{x\}, a)
\]

if you’ve already computed \( \delta \) on the singleton sets. In our example, \( \delta(\{\iota, p\}, 0) = \{\iota, p\} \) and \( \delta(\{\iota, p\}, 1) = \{\iota\} \).

The example has given us a four-state DFA that correctly decides \( L(M) \) (Figure 14-18).

We’ll prove below that the languages of the DFA and NFA are always the same when we carry out this construction. But first note that the state \( \{p\} \) in our constructed DFA is inaccessible — no possible path from the start state (or anywhere else, for that matter) leads to it, so our DFA would behave exactly the same if we left it out and had a three-state DFA. In general, we can save ourselves a lot of work by building our DFA in a lazy way. Build the start state, then repeatedly find \( \delta(S, a) \) for every set \( S \) you’ve found already, and every letter \( a \), until the process closes (that is, until every arrow out of a state you’ve found terminates at a state you’ve found). This must happen eventually, as there are only finitely many states, but it often happens long before all the states have been reached. By not considering states until we’re sure that they’re reachable from the start state, we avoid wasting our time on the others.

14.1.2 Another Example: The No-aba Language

In Section 14.2 we defined a language called No-aba, consisting of all strings over \( \Sigma = \{a, b\} \) that do not have \( aba \) as a substring. There is perhaps no obvious way to design an NFA for this language, other than the DFA’s we’ve already constructed, but if we didn’t have the DFA we might well have hit upon the following NFA for its complement Yes-aba (Figure 14-19), based on the regular

\[\text{We could now check that the three-state DFA is the minimal automaton for this language – see Exercise 14.6.1.}\]

14-2
expression $\Sigma^*aba\Sigma^*$. Define our NFA to have states \{1, 2, 3, 4\}, start state 1, final state set \{4\}, and transitions (1, a, 1), (1, b, 1), (1, a, 2), (2, b, 3), (3, a, 4), (4, a, 4), and (4, b, 4). This NFA can stay at state 1 as long as it wants, go to state 4 by reading an $aba$ substring, and then stay in the final state 4 for the rest of the string. It’s pretty obvious that a string can be accepted by this NFA if and only if it has an $aba$ substring.

Note right away that we cannot convert this NFA for the language “Yes-aba” to an NFA for No-aba by swapping the final and the non-final states. Lots of strings have some accepting paths and some non-accepting paths, and these strings would be accepted by both our NFA and this alleged “complement” NFA\(^2\). We can use the subset construction, however, to get a DFA for the language Yes-aba and then swap that machine’s final and non-final states to get a DFA for the language No-aba.

So our first state is the start state, \{1\}. On an $a$, we could go to state 1 or 2, so $\delta(\{1\}, a) = \{1, 2\}$. On a $b$ we can only go to 1 so $\delta(\{1\}, b) = \{1\}$. We’ve created one new state, which we must now process. $\delta(\{1, 2\}, a) = \{1, 2\}$ because we could go to 1 or 2 from 1 or nowhere from 2. $\delta(\{1, 2\}, b) = \{1, 3\}$ because we can go to 1 from 1 or to 3 from 2. Now we must process the new state \{1, 3\}. $\delta(\{1, 3\}, a) = \{1, 2, 4\}$ because we can go from 1 to 1 or 2, and from 3 to 4. $\delta(\{1, 3\}, b) = \{1\}$ because there is nowhere to go from 3. There is still a new state, \{1, 2, 4\}. $\delta(\{1, 2, 4\}, a) = \{1, 2, 4\}$, but $\delta(\{1, 2, 4\}, b) = \{1, 3, 4\}$ so we still have more states. $\delta(\{1, 3, 4\}, a) = \{1, 2, 4\}$, so we can hope to finish, but $\delta(\{1, 3, 4\}, b) = \{1, 4\}$ so we’re not done yet. Fortunately $\delta(\{1, 4\}, a) = \{1, 2, 4\}$ and $\delta(\{1, 4\}, b) = \{1, 4\}$, so we’re done after examining only six of the sixteen possible states (Figure 14-20). The three final states are those containing 4: \{1, 4\}, \{1, 2, 4\}, and \{1, 3, 4\}.

We could have said from inspection of the NFA that state 1 would be part of any set of states that was reachable in the DFA, because it is clear that for any input string $w$, 1 is reachable from 1 by reading $w$ (as there are loops at state 1 for each letter). The six-state result of our process is not minimal, of course. The three final states have arrows only to each other, so they could be collapsed into a single final state with loops for each letter, leaving a four-state DFA that is just the DFA we had before\(^3\).

14.1.3 Validity of the Subset Construction

It remains only to give a formal proof that the Subset Construction is correct. We have formal recursive definitions for the path relation $\Delta^*$ of the NFA $M$, and the string transition function

\(^2\)In this example, that alleged NFA for the complement would actually accept all strings in $\Sigma^*$.

\(^3\)Of course, the formal state minimization algorithm would also produce this collapse.
\[ \delta^* \text{ of the constructed DFA } M'. \text{ We have defined } w \in L(M) \text{ to be true if and only if } \exists f : (f \in F) \land \Delta^*(\iota, w, f), \text{ and } w \in L(M') \text{ to be true if and only if } \delta^*(\{\iota\}, w) \in F'. \text{ So we will show the two languages to be equal by proving that these two conditions are equivalent. This in turn will follow from the correctness of our construction, specifically that } \]

\[ \delta^*(\{\iota\}, w) = \{ s : \Delta^*(\iota, w, s) \}, \]

a proposition which we will now prove by induction for all strings \( w \).

For the base case, we know that \( \delta^*(\{\iota\}, \epsilon) = \{\iota\} \) by definition of \( \delta^* \), and that \( \Delta^*(\iota, \epsilon, s) \) if and only if \( \iota = s \) by definition of \( \Delta^* \). Therefore our desired equation is true in the case \( w = \epsilon \) because both sides of it are equal to \( \{\iota\} \).

For the inductive case, let our string \( w \) be written \( va \) for some string \( v \) and some letter \( a \). We must show \( \delta^*(\{\iota\}, va) = \{ s : \Delta^*(\iota, va, s) \} \), given the inductive hypothesis that \( \delta^*(\{\iota\}, v) = \{ s : \Delta^*(\iota, v, s) \} \).

This is an exercise in unrolling the definitions. Start with

\[ \delta^*(\{\iota\}, va) \]

and begin by applying the definition of \( \delta^* \) to get

\[ \delta(\delta^*(\{\iota\}, v), a). \]

Apply the inductive hypothesis to get

\[ \delta(\{ s : \Delta^*(\iota, v, s) \}, a) \]

and then apply the definition of \( \delta \) in terms of \( \Delta \) to get

\[ \{ t : \exists u : (u \in \{ s : \Delta^*(\iota, v, s) \}) \land \Delta(u, a, t) \}. \]

Simplifying the set theory, this is

\[ \{ t : \exists u : \Delta^*(\iota, v, u) \land \Delta(u, a, t) \} \]
which by definition of $\Delta^*$ is just

$$\{ t : \Delta^*(\iota, va, t) \}$$

as desired.

In the next section we’ll see how to get from an $\epsilon$-NFA to an ordinary NFA, which combined with this Subset Construction will tell us that DFA’s, NFA’s, and $\epsilon$-NFA’s all accept the same class of languages.

### 14.1.4 Exercises

E14.6.1 Let $D$ be the DFA obtained from the DFA shown in Figure 14-18 by deleting the state $\{p\}$. Prove that $D$ is the minimal automaton for its language.

E14.6.2 Let $N$ be any NFA with $n$ states, input alphabet $\{a, b\}$ and start state $\iota$, whose transition relation contains both $\langle \iota, a, \iota \rangle$ and $\langle \iota, b, \iota \rangle$. Let $D$ be the DFA equivalent to $N$ made by the Subset Construction (avoiding inaccessible states). Prove that $D$ has at most $2^{n-1}$ states.

E14.6.3 Design an NFA whose language is $bb + ba^*a$. Apply the Subset Construction to your NFA to get a DFA. Is your DFA minimal?

E14.6.4 Apply the Subset Construction to the NFA $N$ from Exercise 14.5.5, and describe the resulting DFA. Be sure not to construct unreachable states. Make it minimal if it is not so already.

E14.6.5 Repeat the tasks of Exercise 14.6.4 for the NFA $K$ from Problem 14.5.3, shown in Figure 14-16.

### 14.1.5 Problems

P14.6.1 Define a subDFA to be an NFA in which there is never more than one transition out of a given state for a given letter (there might be zero or one). Prove that the Subset Construction (avoiding inaccessible states) applied to an $n$-state subDFA yields a DFA with at most $n + 1$ states. Explain exactly when it is necessary to add the $n + 1$st state.

P14.6.2 Prove that in any NFA $M$, if $\Delta^*(s, v, t)$ and $\Delta^*(t, w, u)$ are true, then so is $\Delta^*(s, vw, u)$. Here, of course, $s, t,$ and $u$ are arbitrary states and $v$ and $w$ are arbitrary strings over the correct alphabet. Give an example where the converse of this implication is false.

P14.6.3 Let $KJB$ be the language over alphabet $\{a, \ldots, z\}$ consisting of those words occurring in the King James Bible$^4$ so:

$$KJB = \{ \text{in, the, beginning, god, created, ...} \}.

Describe (informally) an NFA whose language is $KJB$. Describe, again informally, what happens when you apply the subset construction to this NFA to get a DFA, ignoring unreachable states. Roughly, how does the number of states of the DFA compare to the number in the

---

$^4$Considered to be written all in lower case. We need to fix a particular edition of a text to pose this problem completely, as there must be complete agreement about which words occur in which order, and with which spellings.
NFA? How does each number compare to the number of distinct words in the King James Bible?

P14.6.4 In Problem 14.3.4 we defined the language \( S = \Sigma^* a \Sigma^k \) of strings whose \( k + 1 \)’st to last letter exists and is \( a \). (Here \( k \) is an arbitrary natural.) Design an NFA for this language with \( k + 2 \) states and apply the Subset Construction to it. Minimizing your DFA if necessary, state a conclusion as to how many states the equivalent DFA of an \( n \)-state NFA might require.

P14.6.5 Does any 2-state (ordinary) NFA have a language whose minimal DFA has four states?