Lecture 4

Regular operations and regular expressions

This lecture focuses on three fundamentally important operations on languages—*union*, *concatenation*, and *Kleene star*—which are collectively known as the *regular operations*. We will prove that the regular languages are *closed* under the regular operations, as well as some other basic operations defined on languages. We will then formally define *regular expressions*, and prove that they offer an alternative characterization of the regular languages.

4.1 Regular operations

Let us begin with a formal definition of the regular operations.

Definition 4.1. The *regular operations* are the operations *union*, *concatenation*, and *Kleene star* (or just *star*, for short), which are defined as follows for any choice of an alphabet Σ and languages $A, B \subseteq \Sigma^*$:

1. *Union.* The language $A \cup B \subseteq \Sigma^*$ is defined as

$$A \cup B = \{w : w \in A \text{ or } w \in B\}. \tag{4.1}$$

   In words, this is just the ordinary union of two sets that happen to be languages.

2. *Concatenation.* The language $AB \subseteq \Sigma^*$ is defined as

$$AB = \{wx : w \in A \text{ and } x \in B\}. \tag{4.2}$$

   In words, this is the language of all strings obtained by concatenating together a string from A and a string from B, with the string from A on the left and the string from B on the right.

Note that there is nothing about a string of the form wx that indicates where w stops and x starts; it is just the sequence of symbols you get by putting w and x together.

3. *Kleene star.* The language A^* is defined as

$$A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup \cdots \tag{4.3}$$

In words, A^* is the language obtained by selecting any finite number of strings from A and concatenating them together. (This includes the possibility to select no strings at all from A, where we follow the convention that concatenating together no strings at all gives the empty string.)

Note that the name *regular operations* is just a name that has been chosen for these three operations. They are special operations and they do indeed have a close connection to the regular languages, but naming them *the regular operations* is a choice we have made and not something mandated in a mathematical sense.

4.2 Closure of regular languages under regular operations

Now we will prove that when regular operations are performed on regular languages, the result must always be a regular language. That is to say, the regular languages are *closed* with respect to the regular operations.

Theorem 4.2. *The regular languages are closed with respect to the regular operations: if $A, B \subseteq \Sigma^*$ are regular languages, then the languages $A \cup B$, AB, and A^* are also regular.*

Proof. Let us assume A and B are fixed regular languages for the remainder of the proof. Because these languages are regular, there must exist DFAs

$$M_A = (P, \Sigma, \delta, p_0, F) \quad \text{and} \quad M_B = (Q, \Sigma, \mu, q_0, G) \tag{4.4}$$

such that $\mathrm{L}(M_A) = A$ and $\mathrm{L}(M_B) = B$. We will make use of these DFAs as we prove that the languages $A \cup B$, AB, and A^* are regular. Because we are free to give whatever names we like to the states of a DFA without influencing the language it recognizes, there is no generality lost in assuming that P and Q are disjoint sets (meaning that $P \cap Q = \varnothing$).

The first regular operation is union. From the previous lecture, we know that if there exists an NFA N such that $\mathrm{L}(N) = A \cup B$, then $A \cup B$ is regular. With that fact in mind, our goal will be to define such an NFA. We will define this NFA N so that its states include all elements of both P and Q, as well as an additional

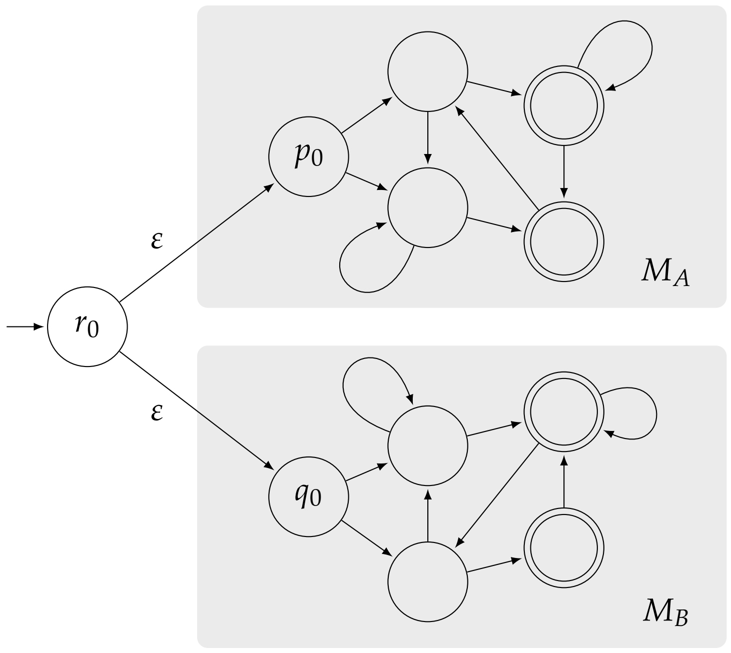


Figure 4.1: DFAs M_A and M_B are combined to form an NFA for the language $L(M_A) \cup L(M_B)$.

state r_0 that is in neither P nor Q. This new state r_0 will be the start state of N. The transition function of N is to be defined so that all of the transitions among the states P defined by δ and all of the transitions among the states Q defined by μ are present, as well as two ε-transitions, one from r_0 to p_0 and one from r_0 to q_0.

Figure 4.1 illustrates what the NFA N looks like in terms of a state diagram. You should imagine that the shaded rectangles labeled M_A and M_B are the state diagrams of M_A and M_B. (The illustrations in the figure are only meant to suggest hypothetical state diagrams for these two DFAs. The actual state diagrams for M_A and M_B can be arbitrary.)

We can specify N more formally as follows:

$$N = (R, \Sigma, \eta, r_0, F \cup G) \tag{4.5}$$

where

$$R = P \cup Q \cup \{r_0\} \tag{4.6}$$

(and we assume P, Q, and $\{r_0\}$ are disjoint sets as suggested above) and the transition function

$$\eta : R \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(R) \tag{4.7}$$

is defined as follows:

$$\begin{aligned}
\eta(p,a) &= \{\delta(p,a)\} && \text{(for all } p \in P \text{ and } a \in \Sigma\text{)},\\
\eta(p,\varepsilon) &= \varnothing && \text{(for all } p \in P\text{)},\\
\eta(q,a) &= \{\mu(q,a)\} && \text{(for all } q \in Q \text{ and } a \in \Sigma\text{)},\\
\eta(q,\varepsilon) &= \varnothing && \text{(for all } q \in Q\text{)},\\
\eta(r_0,a) &= \varnothing && \text{(for all } a \in \Sigma\text{)},\\
\eta(r_0,\varepsilon) &= \{p_0,q_0\}.
\end{aligned}$$

The set of accept states of N in $F \cup G$.

Every string that is accepted by M_A is also accepted by N. This is because N may first follow the ε-transition from r_0 to p_0 and then follow the same transitions that would be followed by M_A. Because the accept states of N include all of the accept states of M_A, this allows N to accept.

By similar reasoning, every string accepted by M_B is also accepted by N.

Finally, every string that is accepted by N must be accepted by either M_A or M_B (or both), because every accepting computation of N begins with one of the two ε-transitions and then necessarily mimics an accepting computation of either M_A or M_B depending on which ε-transition was taken. It therefore follows that

$$\mathrm{L}(N) = \mathrm{L}(M_A) \cup \mathrm{L}(M_B) = A \cup B, \tag{4.8}$$

and so we conclude that $A \cup B$ is regular.

The second regular operation is concatenation. The idea is similar to the proof that $A \cup B$ is regular: we will use the DFAs M_A and M_B to define an NFA N for the language AB. This time we will take the state set of N to be the union $P \cup Q$, and the start state p_0 of M_A will be the start state of N. All of the transitions defined by M_A and M_B will be included in N, and in addition we will add an ε-transition from each accept state of M_A to the start state of M_B. Finally, the accept states of N will be just the accept states G of M_B (and not the accept states of M_A). Figure 4.2 illustrates the construction of N based on M_A and M_B.

In formal terms, N is the NFA defined as

$$N = (P \cup Q, \Sigma, \eta, p_0, G) \tag{4.9}$$

where the transition function

$$\eta : (P \cup Q) \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(P \cup Q) \tag{4.10}$$

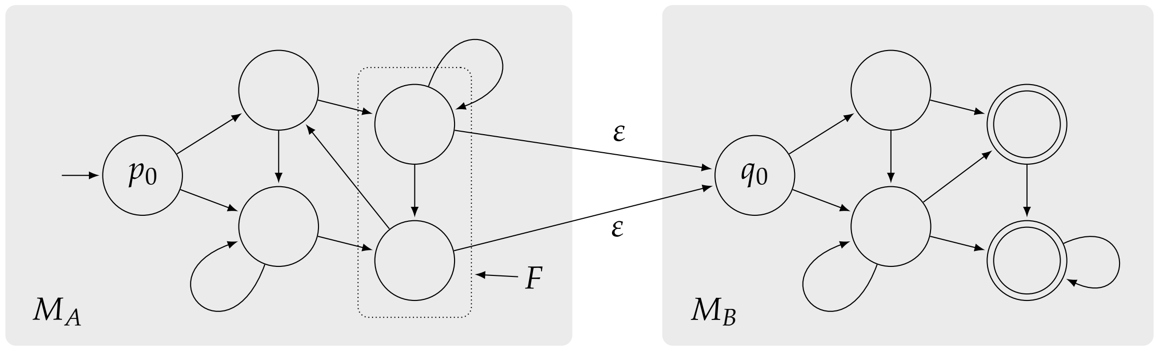


Figure 4.2: DFAs M_A and M_B are combined to form an NFA for the language $\mathrm{L}(M_A)\,\mathrm{L}(M_B)$.

is given by

$$\begin{aligned}
\eta(p,a) &= \{\delta(p,a)\} && \text{(for all } p \in P \text{ and } a \in \Sigma\text{)},\\
\eta(q,a) &= \{\mu(q,a)\} && \text{(for all } q \in Q \text{ and } a \in \Sigma\text{)},\\
\eta(p,\varepsilon) &= \{q_0\} && \text{(for all } p \in F\text{)},\\
\eta(p,\varepsilon) &= \varnothing && \text{(for all } p \in P\backslash F\text{)},\\
\eta(q,\varepsilon) &= \varnothing && \text{(for all } q \in Q\text{)}.
\end{aligned}$$

Along similar lines to what was done in the proof that $A \cup B$ is regular, one can argue that N recognizes the language AB, from which it follows that AB is regular.

The third regular operation is star. We will prove that A^* is regular, and once again the proof proceeds along similar lines. This time we will just consider M_A and not M_B because the language B is not involved. Let us start with the formal specification of N this time; define

$$N = (R, \Sigma, \eta, r_0, \{r_0\}) \tag{4.11}$$

where $R = P \cup \{r_0\}$ and the transition function

$$\eta : R \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(R) \tag{4.12}$$

is defined as

$$\begin{aligned}
\eta(r_0,a) &= \varnothing && \text{(for all } a \in \Sigma\text{)},\\
\eta(r_0,\varepsilon) &= p_0, && \\
\eta(p,a) &= \{\delta(p,a)\} && \text{(for every } p \in P \text{ and } a \in \Sigma\text{)},\\
\eta(p,\varepsilon) &= \{r_0\} && \text{(for every } p \in F\text{)},\\
\eta(p,\varepsilon) &= \varnothing && \text{(for every } p \in P\backslash F\text{)}.
\end{aligned}$$

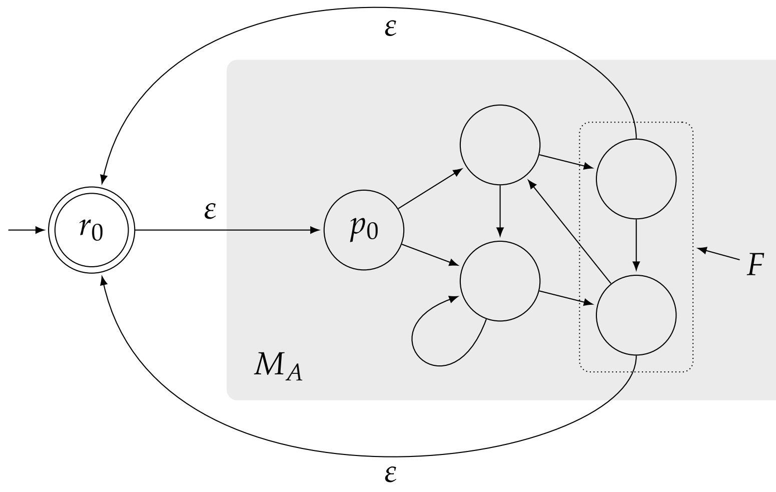


Figure 4.3: The DFA M_A is modified to form an NFA for the language $\mathrm{L}(M_A)^*$.

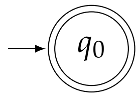
In words, we take N to be the NFA whose states are the states of M_A along with an additional state r_0, which is both the start state of N and its only accept state. The transitions of N include all of the transitions of M_A, along with an ε-transition from r_0 to the start state p_0 of M_A, and ε-transitions from all of the accept states of M_A back to r_0. Figure 4.3 provides an illustration of how N relates to M_A.

It is evident that N recognizes the language A^*. This is because the strings it accepts are precisely those strings that cause N to start at r_0 and loop back to r_0 zero or more times, with each loop corresponding to some string that is accepted by M_A. As $\mathrm{L}(N) = A^*$, it follows that A^* is regular, and so the proof is complete. □

It is natural to ask why we could not easily conclude, for a regular language A, that A^* is regular using the fact that the regular languages are closed under both union and concatenation. In more detail, we have that

$$A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup \cdots \tag{4.13}$$

It is easy to see that the language $\{\varepsilon\}$ is regular—here is the state diagram for an NFA that recognizes the language $\{\varepsilon\}$ (for any choice of an alphabet):



The language $\{\varepsilon\} \cup A$ is therefore regular because the union of two regular languages is also regular. We also have that AA is regular because the concatenation of two regular languages is regular, and therefore $\{\varepsilon\} \cup A \cup AA$ is regular because it is the union of the two regular languages $\{\varepsilon\} \cup A$ and AA. Continuing on like this we find that the language

$$\{\varepsilon\} \cup A \cup AA \cup AAA \tag{4.14}$$

is regular, the language

$$\{\varepsilon\} \cup A \cup AA \cup AAA \cup AAAA \tag{4.15}$$

is regular, and so on. Does this imply that A^* is regular?

The answer is no. Although it is true that A^* is regular whenever A is regular, as we proved earlier, the argument just suggested based on combining unions and concatenations alone does not establish it. This is because we can never conclude from this argument that the *infinite* union (4.13) is regular, but only that *finite* unions such as (4.15) are regular.

If you are still skeptical or uncertain, consider this statement:

> If A is a finite language, then A^* is also a finite language.

This statement is false in general. For example, $A = \{0\}$ is finite, but

$$A^* = \{\varepsilon, 0, 00, 000, \ldots\} \tag{4.16}$$

is infinite. On the other hand, it is true that the union of two finite languages is finite, and the concatenation of two finite languages is finite, so something must go wrong when you try to combine these facts in order to conclude that A^* is finite. The situation is similar when the property of being finite is replaced by the property of being regular.

4.3 Other closure properties of regular languages

There are many other operations on languages aside from the regular operations under which the regular languages are closed. For example, the *complement* of a regular language is also regular. Just to be sure the terminology is clear, here is the definition of the complement of a language.

Definition 4.3. Let $A \subseteq \Sigma^*$ be a language over the alphabet Σ. The *complement* of A, which is denoted $\overline{A}$, is the language consisting of all strings over Σ that are not contained in A:

$$\overline{A} = \Sigma^* \backslash A. \tag{4.17}$$

(For the sake of clarity, we will use a backslash to denote set differences: $S\backslash T$ is the set of all elements in S that are not in T.)

Proposition 4.4. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a regular language over the alphabet Σ. The language $\overline{A}$ is also regular.*

This proposition is very easy to prove: because A is regular, there must exist a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $\mathrm{L}(M) = A$. We obtain a DFA for the language $\overline{A}$ simply by swapping the accept and reject states of M. That is, the DFA $K = (Q, \Sigma, \delta, q_0, Q\backslash F)$ recognizes $\overline{A}$.

While it is easy to obtain a DFA for the complement of a language if you have a DFA for the original language simply by swapping the accept and reject states, this does not work for NFAs. You might, for instance, swap the accept and reject states of an NFA and end up with an NFA that recognizes something very different from the complement of the language you started with. This is due to the asymmetric nature of accepting and rejecting for nondeterministic models.

Within the next few lectures we will see more examples of operations under which the regular languages are closed. Here is one more for this lecture.

Proposition 4.5. *Let Σ be an alphabet and let A and B be regular languages over the alphabet Σ. The intersection $A \cap B$ is also regular.*

This time we can just combine closure properties we already know to obtain this one. This is because De Morgan's laws imply that

$$A \cap B = \overline{\overline{A} \cup \overline{B}}. \tag{4.18}$$

If A and B are regular, then it follows that $\overline{A}$ and $\overline{B}$ are regular, and therefore $\overline{A} \cup \overline{B}$ is regular, and because the complement of this regular language is $A \cap B$ we have that $A \cap B$ is regular.

There is another way to conclude that $A \cap B$ is regular, which is arguably more direct. Because the languages A and B are regular, there must exist DFAs

$$M_A = (P, \Sigma, \delta, p_0, F) \quad \text{and} \quad M_B = (Q, \Sigma, \mu, q_0, G) \tag{4.19}$$

such that $\mathrm{L}(M_A) = A$ and $\mathrm{L}(M_B) = B$. We can obtain a DFA M recognizing $A \cap B$ using a *Cartesian product* construction:

$$M = (P \times Q, \Sigma, \eta, (p_0, q_0), F \times G) \tag{4.20}$$

where

$$\eta((p,q), a) = (\delta(p,a), \mu(q,a)) \tag{4.21}$$

for every $p \in P$, $q \in Q$, and $a \in \Sigma$. In essence, the DFA M is what you get if you build a DFA that runs M_A and M_B in parallel, and accepts if and only if both M_A and M_B accept. You could also get a DFA for $A \cup B$ using a similar idea (but accepting if and only if M_A accepts or M_B accepts).

4.4 Regular expressions

Regular expressions are commonly used in programming languages and other applications to specify patterns for searching and string matching. When regular expressions are used in practice, they are typically endowed with a rich set of convenient operations, but in this course we shall take a minimal definition of regular expressions allowing only the three regular operations (and no other operations like negation or special symbols marking the first or last characters of an input).

Here is the formal definition of regular expressions. The definition is an example of an *inductive definition*, and some comments on inductive definitions will follow.

Definition 4.6. Let Σ be an alphabet. It is said that R is a *regular expression* over the alphabet Σ if any of these properties holds:

1. $R = \varnothing$.
2. $R = \varepsilon$.
3. $R = a$ for some choice of $a \in \Sigma$.
4. $R = (R_1 \cup R_2)$ for regular expressions R_1 and R_2.
5. $R = (R_1 R_2)$ for regular expressions R_1 and R_2.
6. $R = (R_1^*)$ for a regular expression R_1.

When you see an inductive definition such as this one, you should interpret it in the most sensible way, as opposed to thinking of it as something circular or paradoxical. For instance, when it is said that $R = (R_1^*)$ for a regular expression R_1, it is to be understood that R_1 is *already* well-defined as a regular expression. We cannot, for instance, take R_1 to be the regular expression R that we are defining—for then we would have $R = (R)^*$, which might be interpreted as a strange, fractal-like expression that looks like this:

$$R = (((\cdots(\cdots)^*\cdots)^*)^*)^*. \tag{4.22}$$

Such a thing makes no sense as a regular expression, and is not valid according to a sensible interpretation of the definition.

Here are some valid examples of regular expressions over the binary alphabet $\Sigma = \{0, 1\}$:

$$\begin{gathered} \varnothing \\ \varepsilon \\ 0 \\ 1 \\ (0 \cup 1) \\ ((0 \cup 1)^*) \\ (((0 \cup \varepsilon)^*)1) \end{gathered}$$

When we are talking about regular expressions over an alphabet Σ, you should think of them as being strings over the alphabet

$$\Sigma \cup \{\, (\, ,\,)\, ,\, *\, ,\, \cup,\, \varepsilon,\, \varnothing\} \tag{4.23}$$

(assuming of course that Σ and $\{\, (\, ,\,)\, ,\, *\, ,\, \cup,\, \varepsilon,\, \varnothing\}$ are disjoint). Some authors will use a different font for regular expressions so that this is more obvious, but this will not be done in these notes.

Next we will define the *language recognized* (or *matched*) by a given regular expression. Again it is an inductive definition, and it directly parallels the regular expression definition itself. If it looks to you like it is stating something obvious, then your impression is correct—we require a formal definition, but it essentially says that we should define the language matched by a regular expression in the most straightforward and natural way.

Definition 4.7. Let R be a regular expression over the alphabet Σ. The *language recognized* by R, which is denoted $\mathrm{L}(R)$, is defined as follows:

1. If $R = \varnothing$, then $\mathrm{L}(R) = \varnothing$.
2. If $R = \varepsilon$, then $\mathrm{L}(R) = \{\varepsilon\}$.
3. If $R = a$ for $a \in \Sigma$, then $\mathrm{L}(R) = \{a\}$.
4. If $R = (R_1 \cup R_2)$ for regular expressions R_1 and R_2, then $\mathrm{L}(R) = \mathrm{L}(R_1) \cup \mathrm{L}(R_2)$.
5. If $R = (R_1 R_2)$ for regular expressions R_1 and R_2, then $\mathrm{L}(R) = \mathrm{L}(R_1)\,\mathrm{L}(R_2)$.
6. If $R = (R_1^*)$ for a regular expression R_1, then $\mathrm{L}(R) = \mathrm{L}(R_1)^*$.

Order of precedence for regular operations

It might appear that regular expressions arising from Definition 4.6 have a lot of parentheses. For instance, the regular expression $(((0 \cup \varepsilon)^*)1)$ has more parentheses than it has non-parenthesis symbols. The parentheses ensure that every regular expression has an unambiguous meaning.

We can, however, reduce the need for so many parentheses by introducing an *order of precedence* for the regular operations. The order is as follows:

1. star (highest precedence)
2. concatenation
3. union (lowest precedence).

To be more precise, we are not changing the formal definition of regular expressions, we are just introducing a convention that allows some parentheses to be implicit, which makes for simpler-looking regular expressions. For example, we write

$$10^* \cup 1 \tag{4.24}$$

rather than

$$((1(0^*)) \cup 1). \tag{4.25}$$

Having agreed upon the order of precedence above, the simpler-looking expression is understood to mean the second expression.

A simple way to remember the order of precedence is to view the regular operations as being analogous to algebraic operations that you are already familiar with: star looks like exponentiation, concatenation looks like multiplication, and unions are similar to additions. So, just as the expression $xy^2 + z$ has the same meaning as $((x(y^2)) + z)$, the expression $10^* \cup 1$ has the same meaning as $((1(0^*)) \cup 1)$.

Regular expressions characterize the regular languages

At this point it is natural to ask which languages have regular expressions. The answer is that the class of languages having regular expressions is precisely the class of regular languages. If it were otherwise, you would have to wonder why the names were chosen as they were.

There are two implications needed to establish that the regular languages coincide with the class of languages having regular expressions. Let us start with the first implication, which is the content of the following proposition.

Proposition 4.8. *Let Σ be an alphabet and let R be a regular expression over the alphabet Σ. The language $\mathrm{L}(R)$ is regular.*

The idea behind the proof of this proposition is simple enough: we can easily build DFAs for the languages $\varnothing$, $\{\varepsilon\}$, and $\{a\}$ (for each symbol $a \in \Sigma$), and by repeatedly using the constructions described in the proof of Theorem 4.2, one can combine together such DFAs to build an NFA recognizing the same language as any given regular expression.

The other implication is the content of the following theorem, which is more difficult to prove than the proposition above.

Theorem 4.9. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a regular language. There exists a regular expression over the alphabet Σ such that $\mathrm{L}(R) = A$.*

Proof. Because A is regular, there must exist a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $\mathrm{L}(M) = A$. We are free to use whatever names we like for the states of a DFA, so no generality is lost in assuming $Q = \{1, \ldots, n\}$ for some positive integer n.

We are now going to define the language

$$B_{p,q}^{k} \subseteq \Sigma^*, \tag{4.26}$$

for every choice of states $p, q \in \{1, \ldots, n\}$ and an integer $k \in \{0, \ldots, n\}$, to be the set of all strings w that cause M to operate in the following way:

> If we start M in the state p, then by reading w the DFA M moves to the state q. Moreover, aside from the beginning state p and the ending state q, the DFA M only touches states contained in the set $\{1, \ldots, k\}$ when reading w in this way.

For example, the language $B_{p,q}^{n}$ is simply the set of all strings causing M to move from p to q because restricting the intermediate states that M touches to those contained in the set $\{1, \ldots, n\}$ is no restriction whatsoever. At the other extreme, the set $B_{p,q}^{0}$ must be a finite set; it could be the empty set if there are no direct transitions from p to q, it includes the empty string in the case $p = q$, and in general it includes a length-one string corresponding to each symbol that causes M to transition from p to q.

Now, we will prove by induction on k that there exists a regular expression $R_{p,q}^{k}$ satisfying

$$\mathrm{L}(R_{p,q}^{k}) = B_{p,q}^{k}, \tag{4.27}$$

for every choice of $p, q \in \{1, \ldots, n\}$ and $k \in \{0, \ldots, n\}$. The base case is $k = 0$. The language $B_{p,q}^{0}$ is finite for every $p, q \in \{1, \ldots, n\}$, consisting entirely of strings of length 0 or 1, so it is straightforward to define a corresponding regular expression $R_{p,q}^{0}$ that matches $B_{p,q}^{0}$.

For the induction step, we assume $k \geq 1$, and that there exists a regular expression $R_{p,q}^{k-1}$ satisfying

$$\mathrm{L}(R_{p,q}^{k-1}) = B_{p,q}^{k-1} \tag{4.28}$$

for every $p, q \in \{1, \ldots, n\}$. It is the case that

$$B_{p,q}^{k} = B_{p,q}^{k-1} \cup B_{p,k}^{k-1}\big(B_{k,k}^{k-1}\big)^{*} B_{k,q}^{k-1}. \tag{4.29}$$

This equality reflects the fact that the strings that cause M to move from p to q through the intermediate states $\{1, \ldots, k\}$ are precisely those strings that either (i) cause M to move from p to q through the intermediate states $\{1, \ldots, k-1\}$, so that state k is not visited as an intermediate state, or (ii) cause M to move from p to q through the intermediate states $\{1, \ldots, k\}$, visiting the state k as an intermediate state one or more times. We may therefore define a regular expression $R_{p,q}^{k}$ satisfying (4.27) for every $p, q \in \{1, \ldots, n\}$ as

$$R_{p,q}^{k} = R_{p,q}^{k-1} \cup R_{p,k}^{k-1}\big(R_{k,k}^{k-1}\big)^{*} R_{k,q}^{k-1}. \tag{4.30}$$

Finally, we obtain a regular expression R satisfying $\mathrm{L}(R) = A$ by defining

$$R = \bigcup_{q \in F} R_{q_0,q}^{n}. \tag{4.31}$$

In words, R is the regular expression we obtain by forming the union over all regular expressions $R_{q_0,q}^{n}$ where q is an accept state. This completes the proof. □

There is a procedure that can be used to convert a given DFA into an equivalent regular expression. The idea behind this conversion process has some similarities to the proof above. It tends to get messy, producing rather large and complicated-looking regular expressions from relatively simple DFAs, but it works—and just like the conversion of an NFA to an equivalent DFA, it can be implemented by a computer.