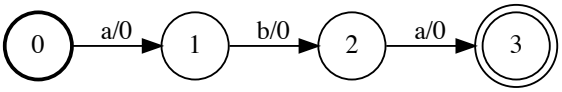


Figure 3.1: An example of a simple acceptor with language $\{aba\}$, in other words $\mathcal{L}(\mathcal{A}) = \{aba\}$.

3 Basic Operations

An operation on a transducer (or acceptor) takes one or more transducers as input and outputs a transducer. You can think of these operations as functions on graphs. As a reminder, I'll use uppercase script letters to represent graphs, so $\mathcal{A}$ for example can represent a graph. Functions will be denoted by lower case variables. So $f(\mathcal{A})$ is a function which takes as input a single graph and outputs a graph.

3.1 Closure

The closure, sometimes called the Kleene star, is a unary function (takes a single input) which can operate on either an acceptor or transducer. If the sequence $\mathbf{x}$ is accepted by $\mathcal{A}$, then zero or more copies of $\mathbf{x}$ are accepted by the closure of $\mathcal{A}$. More formally, if the language of an acceptor is $\mathcal{L}(\mathcal{A})$, then the language of the closure of $\mathcal{A}$ is $\{\mathbf{x}^n \mid \mathbf{x} \in \mathcal{L}(\mathcal{A}), \quad n = 0, 1, \ldots, \}$. The notation $\mathbf{x}^n$ means $\mathbf{x}$ concatenated n times. So $\mathbf{x}^2$ is $\mathbf{xx}$ and $\mathbf{x}^0$ is the empty string. Usually the closure of an acceptor is denoted by *, as in $\mathcal{A}^*$. This is the same notation used in regular expressions.

The closure of a graph is easy to construct with the use of ϵ transitions. The language of the graph in figure 3.1 is the string *aba*.

The closure of the graph needs to accept an arbitrary number of copies of *aba* including the empty string. To accept the empty string we make the start state an accept state as well. To accept one or more copies of *aba* we simply wire up the old accept states to the new start state with ϵ transitions.

The closure of the graph in figure 3.1 is shown in figure 3.2.

Example 3.1. You might notice that state 4 in the graph in figure 3.2 is not necessary. Consider an alternate construction for computing the closure of a graph. We could have made the state 0 into an accept state and connected state 3 to state 0 with an ϵ transition, as in the graph in figure 3.3.

For the graph in figure 3.1, this alternate construction works and requires fewer

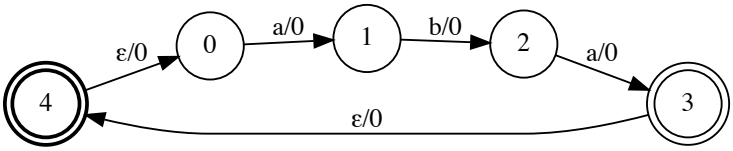


Figure 3.2: The closure $\mathcal{A}^*$ of the graph in figure 3.1. The language of the graph is $\{\epsilon, aba, abaaba, \ldots\}$.

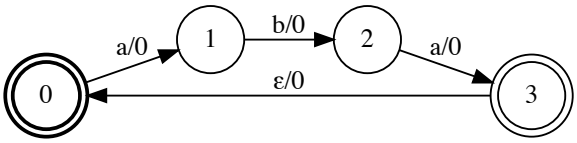


Figure 3.3: The closure $\mathcal{A}^*$ of the graph in figure 3.1 using an alternate, simpler construction which connects the accept state to the original start state with an ϵ transition. This construction does not work for every case.

states and arcs. In the general case, this construction turns every start state into an accept state instead of adding a new start state. Give an example where this doesn't work? In other words, give an example where the graph from this modified construction is not the correct closure of the original graph.

An example for which the alternate construction does not work is shown in the graph in figure 3.4. The language of the graph is $a^n b$ (any number of a's followed by a b) and the closure is $(a^n b)^*$, or any sequence that ends with b.

If we follow the modified construction for the closure, as in the graph in figure 3.5, then the language would incorrectly include sequences that do not end with b such as a^*. The graph following the correct construction of the closure is in figure 3.6.

■

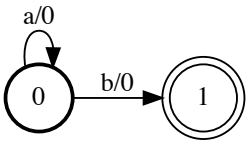


Figure 3.4: An acceptor for which the alternate construction for the closure does not yield the correct result.

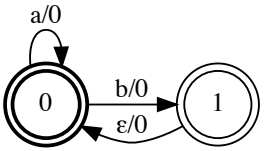


Figure 3.5: The alternate construction which incorrectly computes the closure of the graph in figure 3.4.

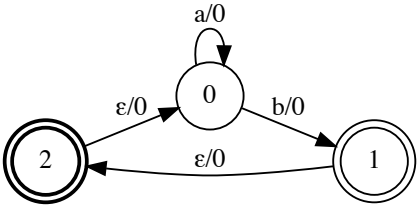


Figure 3.6: The correct closure of the graph in figure 3.4 which has the language $(a^n b)^*$, which is any sequence which ends with b.

3.2 Union

The union takes as input two or more graphs and produces a new graph. The language of the resultant graph is the union of the languages of the input graphs. More formally let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be n graphs. The language of the union graph is given by $\{x \mid x \in \mathcal{A}_i \text{ for some } i = 1, \ldots, n\}$. I'll occasionally use the + sign to denote the union, as in $\mathcal{U} = \mathcal{A}_1 + \mathcal{A}_2$.

Since we let a graph have multiple start states and multiple accept states, the union is easy to construct. A state in the union graph is a start state if it was a start state in one of the original graphs. A state in the union graph is an accept state if it was an accept state in one of the original graphs.

Consider the three graphs in figure 3.7 with languages $\{ab, aba, abaa, \ldots\}$, $\{ba\}$, and $\{ac\}$ respectively.

Notice in the union graph in figure 3.8 the only visual distinction from the individual graphs is that the states are numbered consecutively from 0 to 8 indicating a single graph with nine states instead of three individual graphs. The language of the union graph is $\{ab, aba, abaa, \ldots\} \cup \{ba\} \cup \{ac\}$.

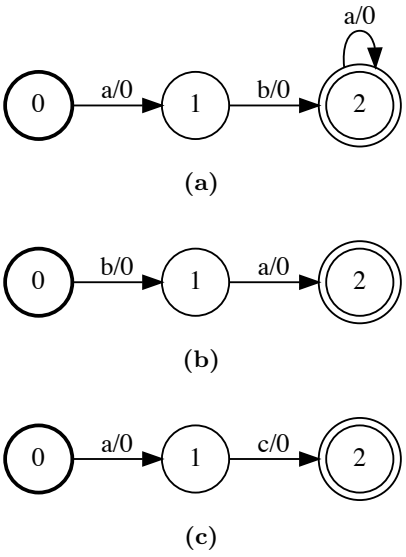


Figure 3.7: Three acceptors with languages $\{ab, aba, abaa, \ldots\}$, $\{ba\}$, and $\{ac\}$ from top to bottom, respectively.

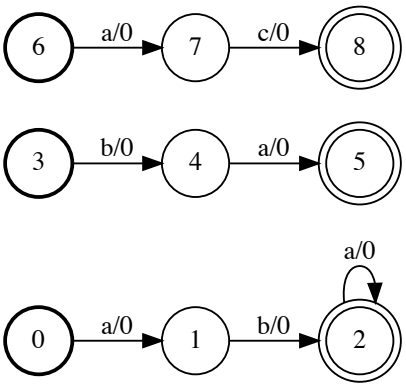


Figure 3.8: The union of the three acceptors in figure 3.7 with language $\{ab, aba, abaa, \ldots\} \cup \{ba\} \cup \{ac\}$.

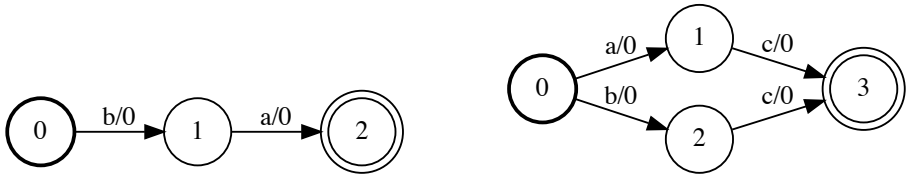


Figure 3.9: The acceptor on left has language $\{ba\}$ and the acceptor on the right has language $\{ac, bc\}$.

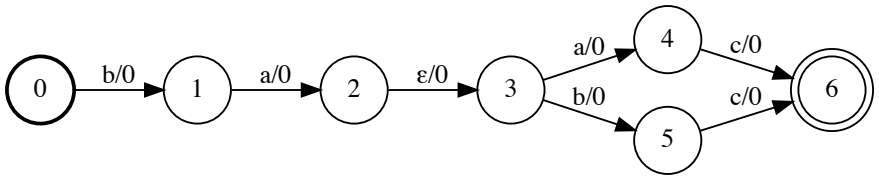


Figure 3.10: The graph is the concatenation of the two graphs in figure 3.9 and has the language $\{baac, babc\}$.

3.3 Concatenate

Like union, concatenate produces a new graph given two or more graphs as input. The language of the concatenated graph is the set of strings which can be formed by any concatenation of strings from the individual graph. Concatenate is not commutative, the order of the input graphs matters. More formally the language of the concatenated graph is given by $\{\mathbf{x}_1 \dots \mathbf{x}_n \mid \mathbf{x}_1 \in \mathcal{L}(\mathcal{A}_1), \dots, \mathbf{x}_n \in \mathcal{L}(\mathcal{A}_n)\}$. Occasionally, I will denote concatenation by placing the graphs side-by-side. So $\mathcal{A}_1\mathcal{A}_2$ represents the concatenation of $\mathcal{A}_1$ and $\mathcal{A}_2$.

The concatenated graph can be constructed from the original input graphs by connecting the accept states of one graph to the start states of the next. Assume we are concatenating $\mathcal{A}_1, \dots, \mathcal{A}_n$. The start states of the concatenated graph are the start states of the first graph, $\mathcal{A}_1$. The accept states of the concatenated graph are the accept states of the last graph, $\mathcal{A}_n$. For any two graph $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$, we connect each accept state of $\mathcal{A}_i$ to each start state of $\mathcal{A}_{i+1}$ with an ϵ transition.

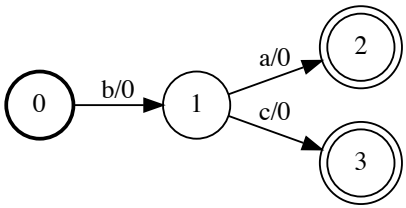
As an example, consider the two graphs in figure 3.9. The concatenated graph is in figure 3.10 and has the language $\{baac, babc\}$.

Example 3.2. What is the identity graph for the concatenation function? The identity in a binary operation is the value which when used in the operation leaves the second input unchanged. In multiplication this would be 1 since $c * 1 = c$ for any real value c.

(a) Identity

(b) Annihilator

Figure 3.11: The identity and the annihilator for the concatenate operation. The language of the identity graph is the empty string $\{\epsilon\}$. The language of the annihilator graph is the empty set $\{\}$.



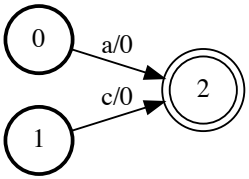


Figure 3.12: The acceptor on the right has the language $\{ba, bc\}$, the acceptor on the left has the language $\{a, c\}$.

What is the equivalent of the annihilator graph in the concatenation function? The annihilator in a binary operation is the value such that the operation with the annihilator always returns the annihilator. For multiplication 0 is the annihilator since $c * 0 = 0$ for any real value c.

The graph which accepts the empty string is the identity. The graph which does not accept any strings is the annihilator. See the figure 3.11 for an example of these two graphs.

The identity graph is a single node which is both a start and accept state. The language of the identity graph is the empty string. The annihilator graph is a single non accepting state. The language of the annihilator graph is the empty set. Note the subtle distinction between the language that contains the empty string and the language that is the empty set. The former can be written as $\{\epsilon\}$ whereas the latter is $\{\}$ (also commonly denoted by $\varnothing$). ■

Example 3.3. Construct the concatenation of the two graphs in figure 3.12.

The concatenated graph is in figure 3.13.

■

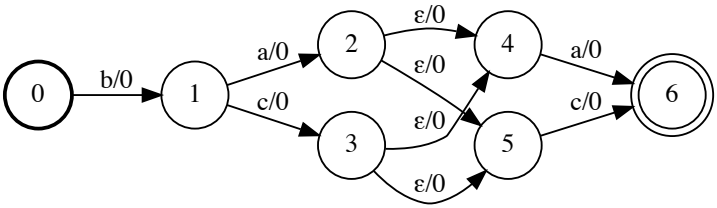


Figure 3.13: The concatenation of the two graphs in figure 3.12 has the language $\{baa, bac, bca, bcc\}$.

Example 3.4. Suppose we have a list of graphs to concatenate $\mathcal{A}_1, \ldots, \mathcal{A}_n$ where the i-th graph has s_i start states and a_i accept states. How many new arcs will the concatenated graph require?

For each consecutive pair of graphs $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$, we need to add $a_i * s_{i+1}$ connecting arcs in the concatenated graph. So the total number of additional arcs is:

$$\sum_{i=1}^{n-1} a_i * s_{i+1}.$$

■

3.4 Summary

We've seen three basic operations so far:

- **Closure:** The closed graph accepts any string in the input graph repeated zero or more times. The closure of a graph $\mathcal{A}$ is denoted $\mathcal{A}^*$.
- **Union:** The union graph accepts any string from any of the input graphs. The union of two graphs $\mathcal{A}_1$ and $\mathcal{A}_2$ is denoted $\mathcal{A}_1 + \mathcal{A}_2$.
- **Concatenate:** The concatenated graph accepts any string which can be formed by concatenating strings (respecting order) from the input graphs. The concatenation of two graphs $\mathcal{A}_1$ and $\mathcal{A}_2$ is denoted $\mathcal{A}_1\mathcal{A}_2$.

Example 3.5. Assume you are given the following individual graphs $\mathcal{A}_a$, $\mathcal{A}_b$, and $\mathcal{A}_c$, which recognize a, b, and c respectively, as in figure 3.14. Using only closure, union, and concatenate, construct the graph which recognizes any number of repeats of the strings aa, bb, and cc. For example $aabb$ and $bbaacc$ are in the language but b and $ccaab$ are not.

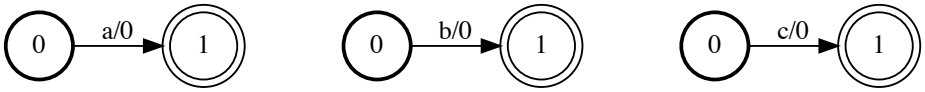


Figure 3.14: The three individual automata with languages $\{a\}$, $\{b\}$, and $\{c\}$ from left to right, respectively.

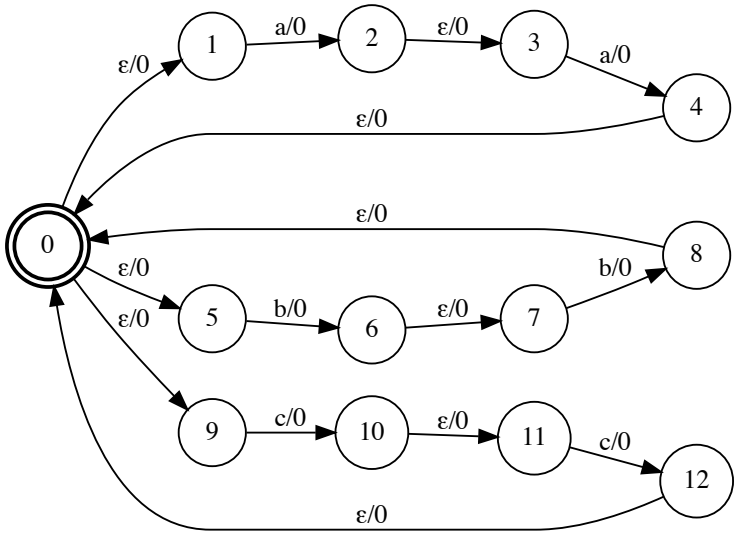


Figure 3.15: The even numbered repeats graph constructed from the individual token graphs using the operations $\mathcal{A} = (\mathcal{A}_a\mathcal{A}_a + \mathcal{A}_b\mathcal{A}_b + \mathcal{A}_c\mathcal{A}_c)^*$.

First concatenate the individual graphs with themselves to get graphs which recognize *aa*, *bb*, and *cc*. Then take the union of the three concatenated graphs followed by the closure. The resulting graph is shown in figure 3.15. The equation to compute the desired graph is $\mathcal{A} = (\mathcal{A}_a\mathcal{A}_a + \mathcal{A}_b\mathcal{A}_b + \mathcal{A}_c\mathcal{A}_c)^*$.

■