# Closure Properties

Removing power from Turing machines provided us with a new machine. We also found a new class of sets. Now it is time to examine this class. Our first questions concern operations on sets within the class.

Set *complement* is our first operation to examine. Since we are dealing with strings (rather than numbers), we must redefine this operation. Thus the definition of complement will slightly different. This is to be expected because even though 0100 and 100 are the same number, they are different strings. Here is the definition. *If a set contains strings over an alphabet, then its complement contains all of the strings (over the alphabet) not in the set.*

> **Theorem 1.** *If a set is accepted by a finite automaton then its complement can be accepted by a finite automaton.*

> **Proof**. Let the finite automaton $M = (S, I, \delta, s_0, F)$ accept the set T(M). We must now show that there is a finite automaton which accepts the set of strings over the alphabet I which M does not accept. Our strategy will be to look at the operation of M, accepting when it rejects and rejecting when M accepts. Since we know that strings which take M to a state of F are accepted by M and those which take M into S-F are rejected, then our course is fairly clear. Consider the machine:
>
> $$M' = (S, I, \delta, s_0, S\text{-}F).$$
>
> It is exactly the same as M except for its final or accepting states. Thus it should accept when M rejects. When we precisely examine this, we find that for all strings x over the alphabet I:
>
> $$\begin{aligned} x \in T(M) &\text{ if and only if } \delta^*(s_0,x) \in F \\ &\text{if and only if } \delta^*(s_0,x) \notin S\text{-}F \\ &\text{if and only if } x \notin T(M') \end{aligned}$$
>
> and so $T(M') = \overline{T(M)}$.

The last proof contained an example of our first method of dealing with finite automata: *rearranging an existing machine.* Now we shall employ another strategy: *combining two machines.* This is actually going to be parallel

processing. Suppose we take the two machines whose state graphs are in figure 1.

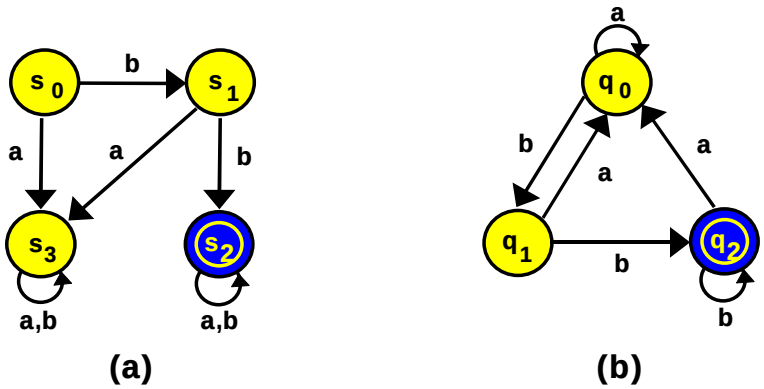
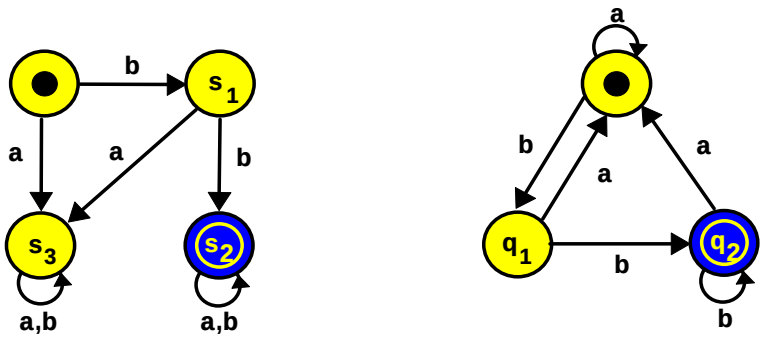


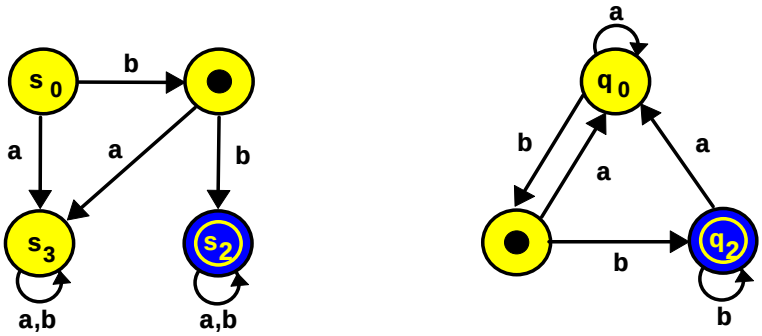

**Figure 1 - Finite Automaton Examples**

We have seen the machine ($M_1$) of figure 1a before, it accepts all strings (over {a, b}) which begin with two b's. The other machine ($M_2$) accepts strings which end with two b's. Let's try to combine them into one machine which accepts strings which either begin or end with two b's.

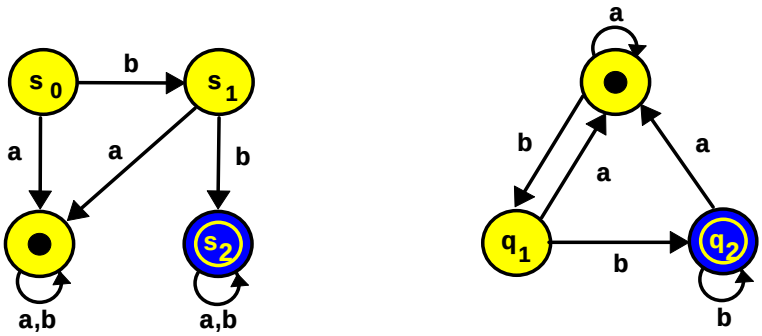
Why not run both machines at the same time on an input? We could keep track of what state each machine is in by placing pebbles upon the appropriate states and then advancing them according to the transition functions of each machine. Both machines begin in their starting states, as pictured in the state graphs below




with pebbles on $s_0$ and $q_0$. If both machines now read the symbol b on their input tapes, they move the pebbles to new states and assume configurations like these




with pebbles on $s_1$ and $q_1$. The pebbles have advanced according to the transition functions of the machines. Now let's have them both read an a. At this point, they both advance their pebbles to the next state and enter the configurations




with pebbles on $s_3$ and $q_0$.

With this picture in mind, let's trace the computations of both machines as they process several input strings. Pay particular attention to the *pairs* of states the machines go through. Our first string is bbabb, which will be accepted by both machines.

| ***Input:*** | **b** | **b** | **a** | **b** | **b** | |
|---|---|---|---|---|---|---|
| ***$M_1$'s states*** | **$s_0$** | **$s_1$** | **$s_2$** | **$s_2$** | **$s_2$** | **$s_2$** |
| ***$M_2$'s states*** | **$q_0$** | **$q_1$** | **$q_2$** | **$q_0$** | **$q_1$** | **$q_2$** |

Now let us look at an input string neither will accept: babab.

| ***Input:*** | **b** | **a** | **b** | **a** | **b** | |
|---|---|---|---|---|---|---|
| ***$M_1$'s states*** | **$s_0$** | **$s_1$** | **$s_3$** | **$s_3$** | **$s_3$** | **$s_3$** |
| ***$M_2$'s states*** | **$q_0$** | **$q_1$** | **$q_0$** | **$q_1$** | **$q_0$** | **$q_1$** |

And finally, the string baabb which will be accepted by $M_2$ but not $M_1$.

| *Input:* | b | a | a | b | b | |
|---|---|---|---|---|---|---|
| *$M_1$'s states* | $s_0$ | $s_1$ | $s_3$ | $s_3$ | $s_3$ | $s_3$ |
| *$M_2$'s states* | $q_0$ | $q_1$ | $q_0$ | $q_0$ | $q_1$ | $q_2$ |

If we imagine a *multiprocessing* finite automaton with two processors (one for $M_1$ and one for $M_2$), it would probably look just like the pictures above. Its *state* could be a state pair (one from each machine) corresponding to the pebble positions. Then, if a pebble ended up on an accepting state for either machine (that is, either $s_2$ or $q_2$), our multiprocessing finite automaton would accept.

This is not difficult at all! All we need to do is to define a new class of machines and we can accept several things at once. (Note that the new machines accept unions of sets accepted by finite automata.) Or, if we think for a bit, we find that we do not need to define a new class for this. The next result shows that we already have this facility with finite automata. The proof of the theorem demonstrates this by careful manipulation of the symbolic definition of finite automata.

**Theorem 2.** *The class of sets accepted by finite automata is closed under union.*

**Proof Sketch.** Let $M_1 = (S, I, \delta, s_0, F)$ and $M_2 = (Q, I, \gamma, q_0, G)$ be two arbitrary finite automata. To prove the theorem we must show that there is another machine ($M_3$) which accepts every string accepted by $M_1$ or $M_2$.

We shall try the multiprocessing pebble machine concept and work it into the definition of finite automata. Thus the states of $M_3$ are pairs of states (one from $M_1$ and one from $M_2$). This works out nicely since the set of pairs of states from S and Q is known as the *cross product* (written $S \times Q$) between S and Q. The starting state is obviously $<s_0 , q_0>$. Thus:

$$M_3 = (S \times Q, I, \xi, <s_0, q_0>, H)$$

where $\xi$ and H will be described presently.

The transition function $\xi$ is just a combination of $\delta$ and $\gamma$, since it simulates the advance of pebbles on the state graphs. It uses $\delta$ to change the states in S and $\gamma$, to change the states in Q. In other words, if a is a symbol of I:

$$\xi(<s_i , q_i >, a) = <\delta(s_i, a), \gamma(q_i, a)>.$$

Our final states are all of the pairs which contain a state from F or a state from G. In cross product notation this is:

$$H = (F \times Q) \cup (S \times G).$$

We have now defined $M_3$ and know that it is indeed a finite automaton because it satisfies the definition of finite automata. We claim it does accept $T(M_1) \cup T(M_2)$ since it mimics the operation of our intuitive multiprocessing pebble machine. The remainder of the formal proof (which we shall leave as an exercise) is merely an induction on the length of input strings to show that for all strings x over the alphabet I:

$$x \in T(M_1) \cup T(M_2) \text{ iff } \delta^*(s_0,x) \in F \text{ or } \gamma^*(q_0,x) \in G$$
$$\text{iff } \xi^*(<s_0 , q_0 >,x) \in H.$$

Thus by construction we have shown that the class of sets accepted by finite automata is closed under union.

By manipulating the notation we have shown that two finite automata can be combined. Let's take a look at the result of applying this construction to the machines of figure 1. (This is pictured in figure 2.)

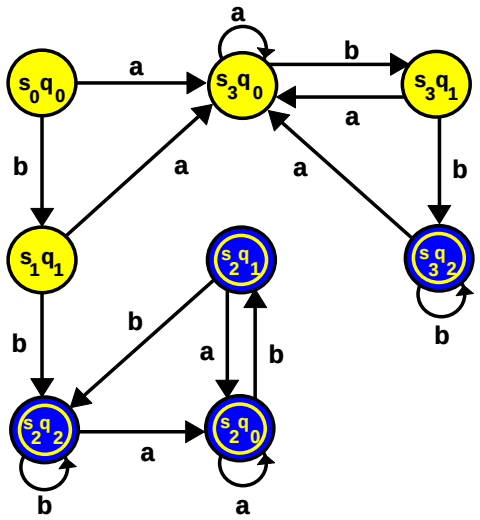



**Figure 2 - Union of $M_1$ and $M_2$**

Note that not all pairs of states are included in the state graph. (For example, $<s_0, q_1>$ and $<s_1, q_2>$ are missing.) This is because it is impossible to get to these states from $<s_0 , q_0>$.

This is indeed a complicated machine! But, if we are a bit clever, we might notice that if the machine enters a state pair containing $s_2$, then it remains in pairs containing $s_2$. Thus we can combine all of these pairs and get the smaller but equivalent machine of figure 3.

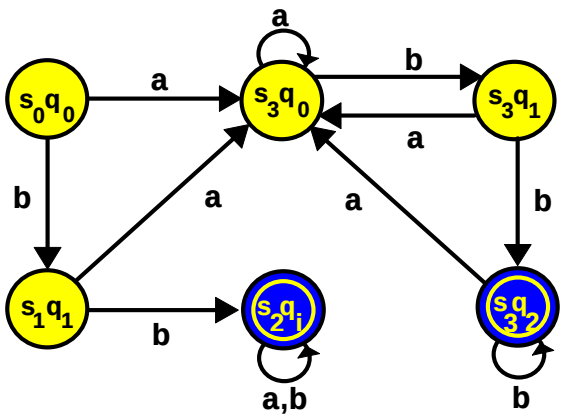



**Figure 3 - Reduced Union Machine**

This is very likely what anyone would design if they had to build a machine from scratch rather than use the construction detailed in the theorem. Lest anyone think that finding smaller, equivalent machines is always this simple, we must admit that this was an elementary example and we did some prior planning to provide such clean results.

The final Boolean operation is set intersection. We state this here and leave the proof as an exercise in Boolean algebra identities (De Morgan's Laws) or machine construction - as you wish!

> **Theorem 3.** *The class of sets accepted by finite automata is closed under intersection.*