1 Finite Automata and Regular Expressions

Motivation: Given a pattern (regular expression) for string searching, we might want to convert it into a deterministic finite automaton or nondeterministic finite automaton to make string searching more efficient; a deterministic automaton only has to scan each input symbol once. Can this always be done?

Theorem 1.1 *If $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for languages $L_i \subseteq \Sigma^*$ then*

1. *there is an automaton M recognizing $L_1 \cup L_2$*
2. *there is an automaton M recognizing $L_1 \circ L_2$*
3. *there is an automaton recognizing L_1^**
4. *there is an automaton recognizing $\Sigma^* - L_1$*
5. *there is an automaton recognizing $L_1 \cap L_2$*
6. *if $a \in \Sigma$ then there is an automaton recognizing $\{a\}$*
7. *there is an automaton recognizing $\emptyset$*

From all of these things it follows that if A is a regular language then there is a finite automaton recognizing A.

For example, justify why there would be a finite automaton recognizing the language represented by $a \cup (ab)^*$.

Proof: We will do the proof for nondeterministic automata since deterministic and nondeterministic automata are of equivalent power.

1.1 Union

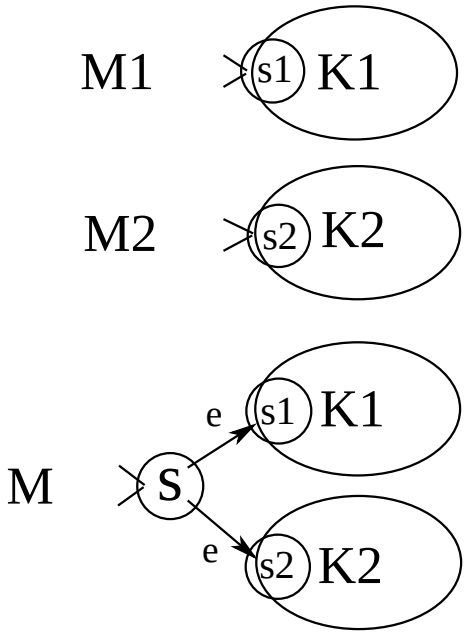
For union, suppose M_1 is $(K_1, \Sigma, \Delta_1, s_1, F_1)$ and M_2 is $(K_2, \Sigma, \Delta_2, s_2, F_2)$. Then let M be $(K, \Sigma, \Delta, s, F)$ where

$$K = K_1 \cup K_2 \cup \{s\}$$

$$F = F_1 \cup F_2$$

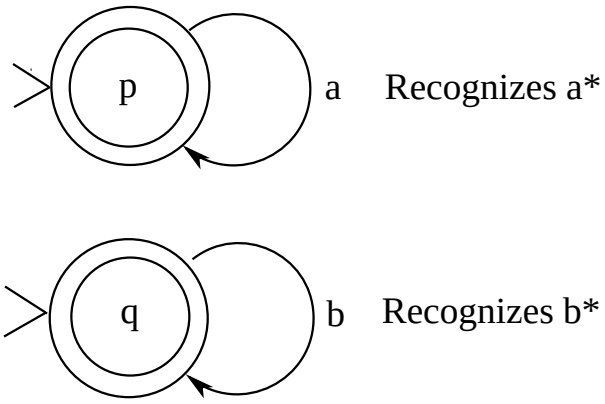
$$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, e, s_1), (s, e, s_2)\}$$

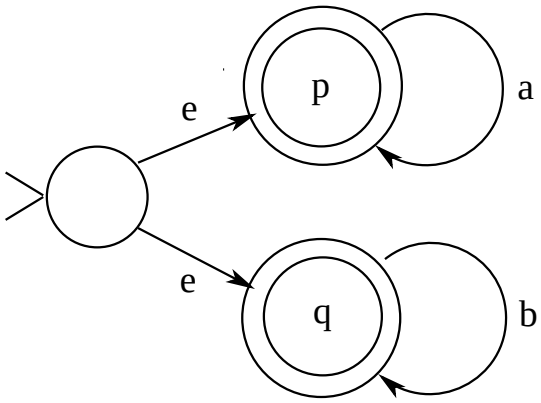
and s is a new state. Then $L(M) = L(M_1) \cup L(M_2)$. Diagram:



Note that ϵ arrows are convenient for this construction.

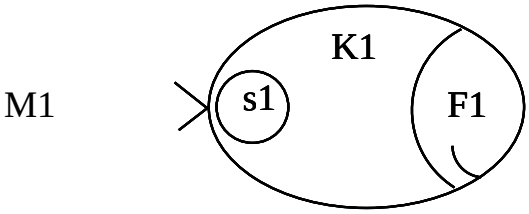
1.1.1 Example

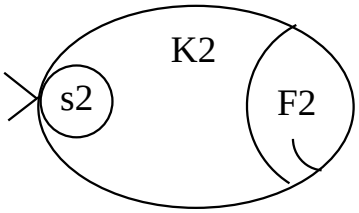


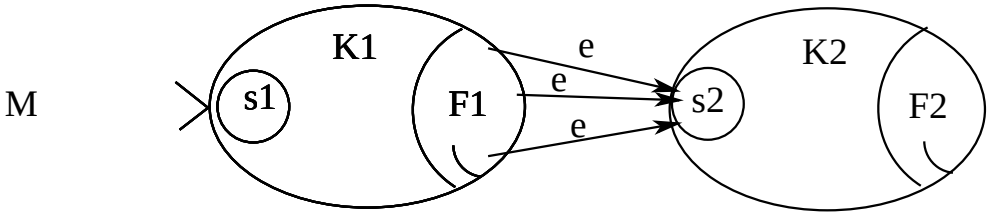


Recognizes a* U b*

1.2 Concatenation

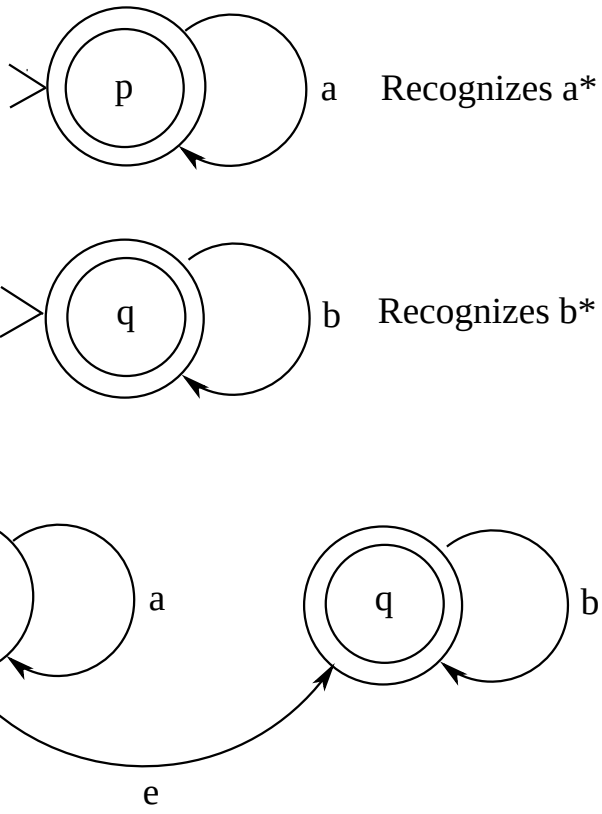




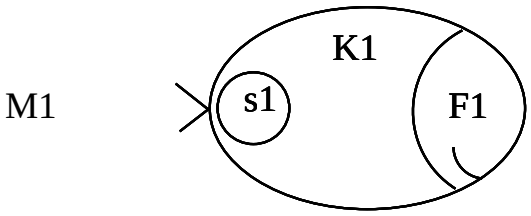


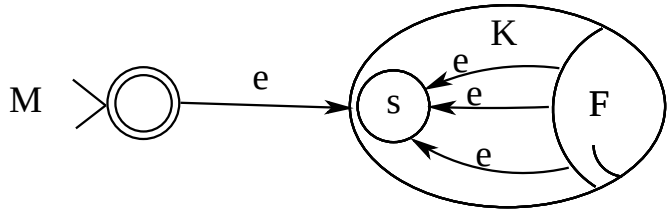
The states in F_1 are no longer accepting states. Then $L(M) = L(M_1) \circ L(M_2)$.

1.2.1 Example



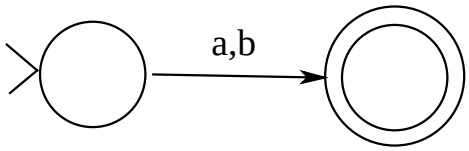
1.3 Kleene star



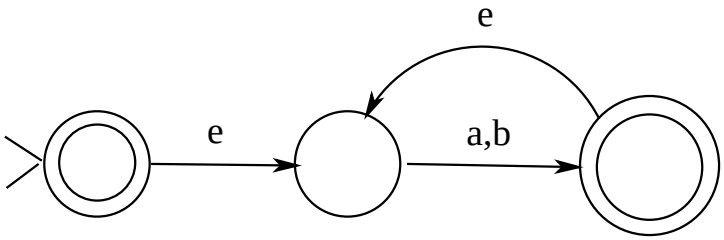


Then $L(M) = L(M_1)^*$.

1.3.1 Example

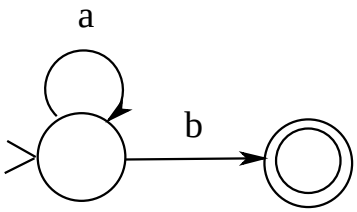


Recognizes {a,b}



Recognizes {a,b}*

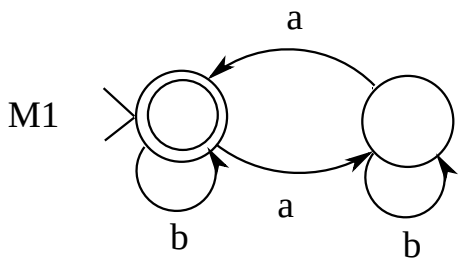
How would you modify this automaton to recognize $\{a, b\}^+$?
Another simple construction for Kleene star fails for this automaton:



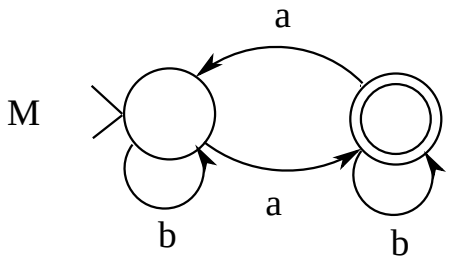
1.4 Complementation

Let $M_1 = (K, \Sigma, \delta, s, F)$ be a *deterministic* finite automaton. Let M be $(K, \Sigma, \delta, s, K - F)$. Then $L(M) = \Sigma^* - L(M_1)$.

1.4.1 Example



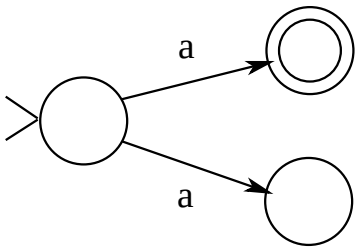
Recognizes strings with even number of a's



Recognizes strings with odd number of a's

Why does the automaton have to be deterministic for this to work?

An example showing that M_1 has to be deterministic for this construction to work:



1.5 Intersection

For this, note that $L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$.

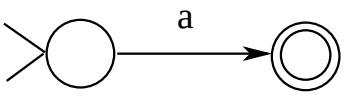
1.6 Other operations

Parts 6 and 7 of the theorem are trivial. Ask students to do them.

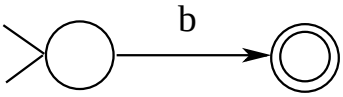
As a consequence of this theorem, if a language L is regular, then there is a finite automaton M recongizing L.

2 Example

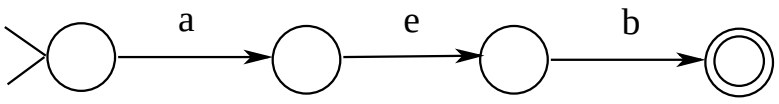
We construct a nondeterministic finite automaton recognizing $\mathcal{L}((ab)^* \cup a)$.



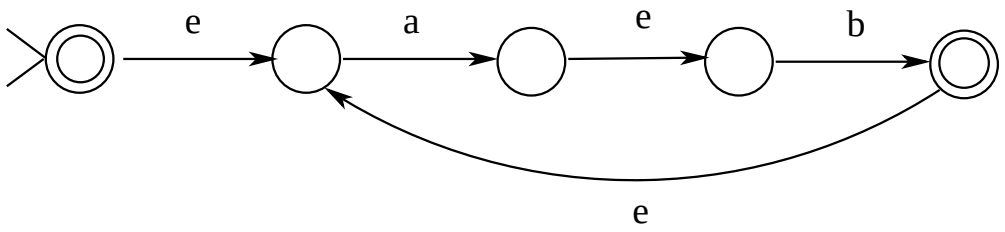
Recognizes {a}



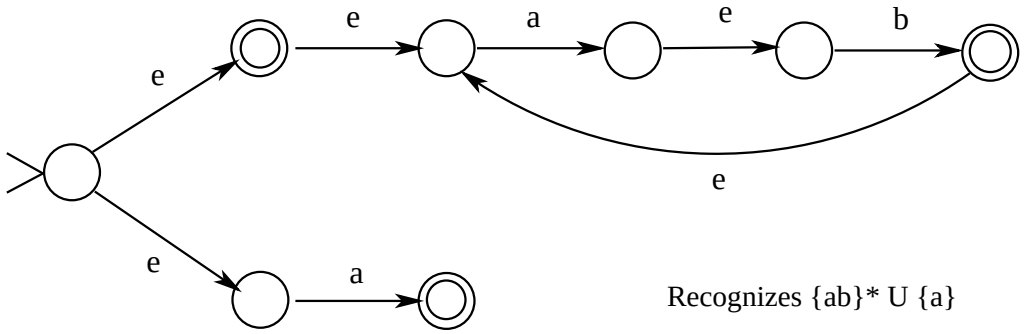
Recognizes {b}



Recognizes {ab}



Recognizes {ab}*



Recognizes {ab}* U {a}

Of course, this automaton is not the simplest possible one! But some such construction can be used for string searching, with $\{a, b\}^*$ put on the front, and can then be simulated using the set idea.

How would you optimize the above automaton to reduce the number of states? What are the simplest nondeterministic and deterministic automata for this language?

We now show that if a language L is recognized by a finite automaton M, then L is regular.

3 Converting automata to regular expressions

Can any finite automaton be converted to an equivalent regular expression? Would allowing finite automata in regular expressions increase the power of string searching? The answers to these questions are yes and no. For any finite automaton M there is a regular expression E such that $L(M) = \mathcal{L}(E)$.

Given a finite automaton, it can be converted to a regular expression. To do this, we generalize nondeterministic finite automata and allow regular

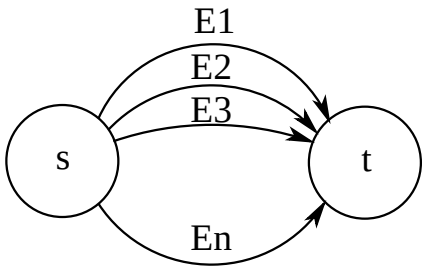
expressions on their arrows. If $s \stackrel{E}{\to} t$ where E is a regular expession, then this means that if the automaton is in state s, it can read a string in $\mathcal{L}(E)$ and transition to state t. Note that ordinary nondeterministic automata do not allow such regular expressions on arrows.

The automaton M can be converted to a regular expression by applying the following rules.

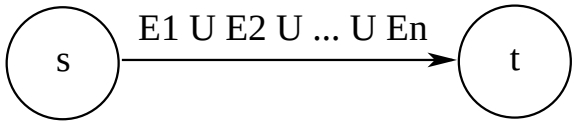
First, whenever possible, the following transformation should be applied to M and to all other automata M', M'', et cetera, obtained during this process:

> If for any states s and t in M,
>
> $$s \stackrel{E_1}{\to} t, \ldots, s \stackrel{E_n}{\to} t$$
>
> for $n > 1$ then all these arrows are removed from M and are replaced by the arrow
>
> $$s \stackrel{E_1 \cup E_2 \cup \ldots \cup E_n}{\to} t.$$

Diagram:



After processing:



Next, all states t of M other than the start state should be processed, one by one, to eliminate arrows leaving t, and possibly to eliminate t. A state

t of M can be processed to obtain another automaton M'. The automaton M' is initially set to be equal to M. Then arrows and states are added to M' and removed from M' as follows:

If in M we have

$$s \xrightarrow{E} t \xrightarrow{F} t \xrightarrow{G} u$$

and t is not the start state, then in M' we add the arrow

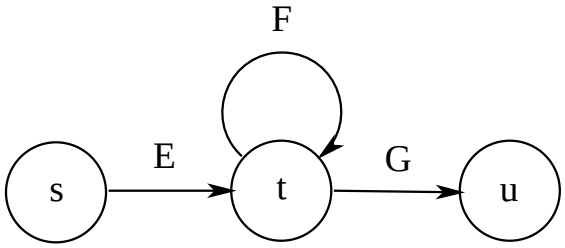
$$s \overset{E(F^*)G}{\longrightarrow} u.$$

Arrows like this are added for all states s and u that are not identical to t.

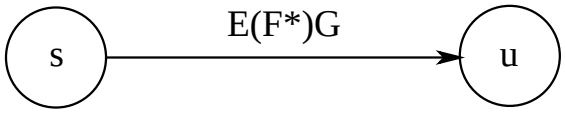
Note that s and u may be identical.

If there are no arrows from t to t, then the expression EG is used instead of $E(F^*)G$.

Diagram:



After processing:



Then, if t is not an accepting state, t and all arrows entering or leaving it are removed from M'. If t is an accepting state, then if in M we have

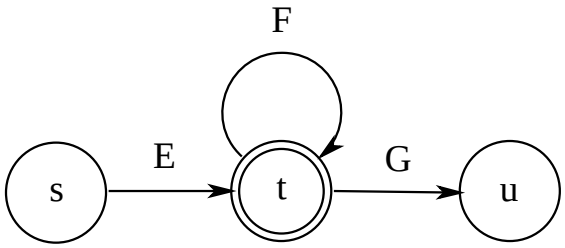
$$s \xrightarrow{E} t \xrightarrow{F} t$$

then in M' we have

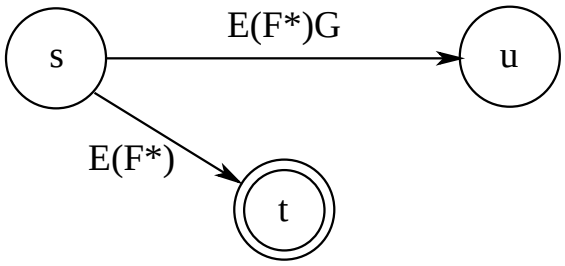
$$s \xrightarrow{E(F^*)} t.$$

This is done for all states s not identical to t. The state t remains in M', but all arrows leaving t are removed from M', and only such added arrows $s \xrightarrow{E(F^*)} t$ enter t in M'. If there are no arrows from t to t in M, then the expression E is used instead of $E(F^*)$.

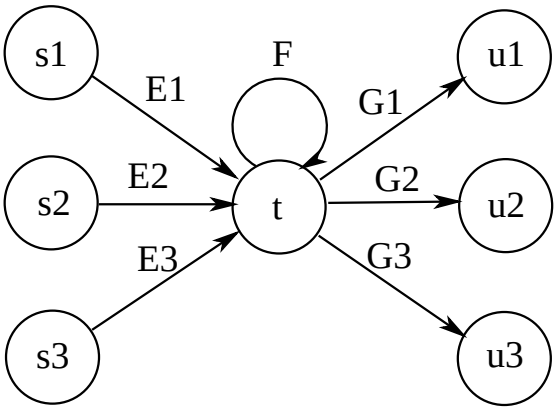
Simple example:



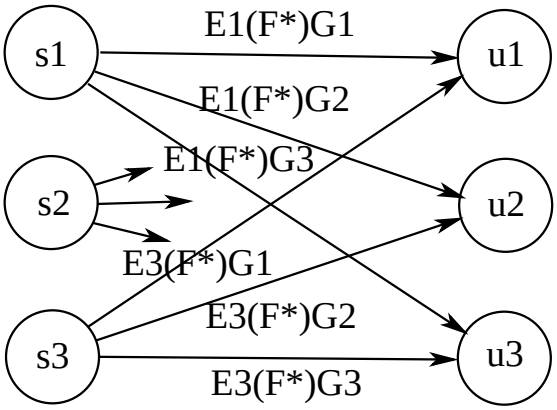
After processing:



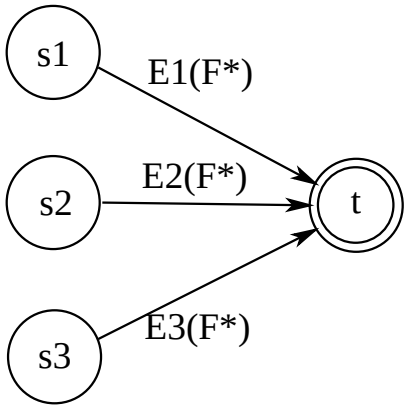
More complex example:



If t is not an accepting state then after processing we have this:



If t is an accepting state then in addition to all these arrows we have this:



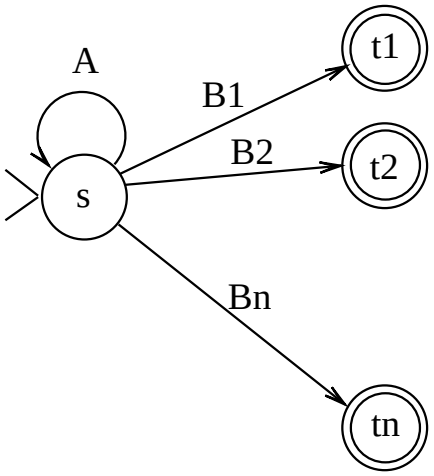
- Then if there is another state t' in M' other than the start state that has arrows leaving it, then some such t' is processed in M' to obtain M''.
- This processing of states t, t', et cetera, continues, repeatedly applying these rules, until an automaton N is obtained in which only the start state has arrows leaving it.
- That is, N only has a start state s and some accepting states $t_1, t_2, \ldots, t_n$ and only has arrows from the start state to itself and to the states $t_1, t_2, \ldots, t_n$.
- The start state s may or may not be an accepting state.
- There will be no arrows leaving the states of N other than the start state.

Thus we may only have the following kinds of arrows in N:

$$s \xrightarrow{A} s$$

$$s \xrightarrow{B_i} t_i, 1 \leq i \leq n$$

Thus N may look like this, if s is not an accepting state:



The final regular expression is obtained in the following way.

If the start state s is not an accepting state, then the final regular expression E is

$$A^*(B_1 \cup B_2 \cup \ldots \cup B_n).$$

If there is no arrow from s to s then E is

$$(B_1 \cup B_2 \cup \ldots \cup B_n).$$

If the start state s is an accepting state, then the final regular expression E is

$$A^*(\emptyset^* \cup B_1 \cup B_2 \cup \ldots \cup B_n).$$

This can also be written as

$$A^* \cup A^*(B_1 \cup B_2 \cup \ldots \cup B_n).$$

If there is no arrow from s to s then E is

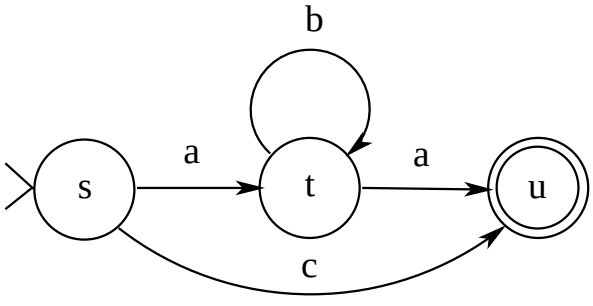
$$(\emptyset^* \cup B_1 \cup B_2 \cup \ldots \cup B_n).$$

Thus from M we obtain a regular expression E, and one can show that $L(M) = \mathcal{L}(E)$, that is, E represents the language recognized by M. The book gives another method to convert automata to regular expressions, but it is much harder to do on examples.

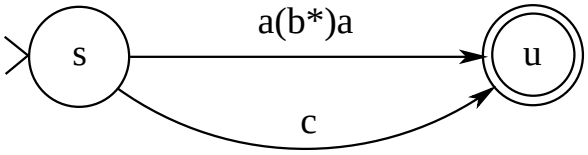
3.1 Examples

Here are some examples of the method.

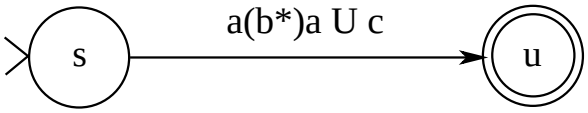
Starting automaton:



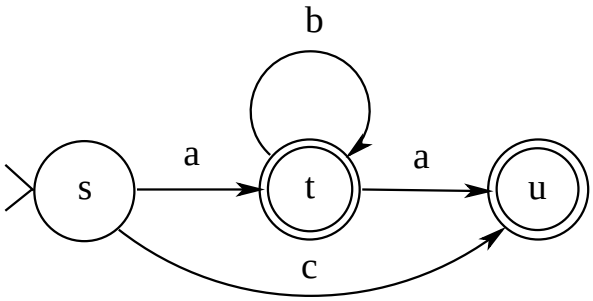
After eliminating t:



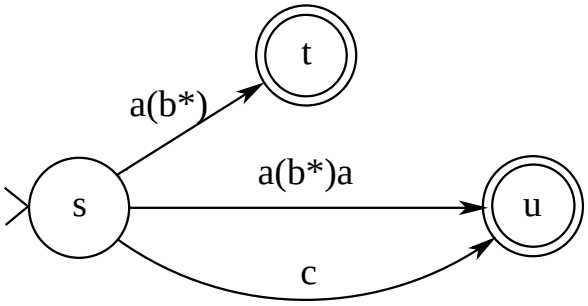
After collapsing arrows:



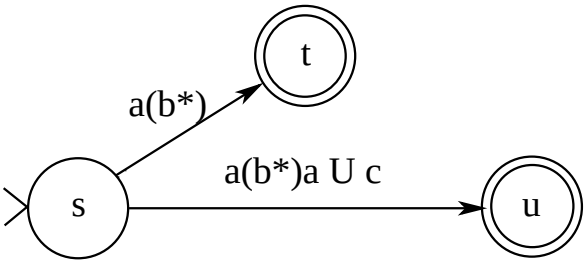
The final regular expression is $ab^*a \cup c$. Now suppose that t is an accepting state in this automaton:



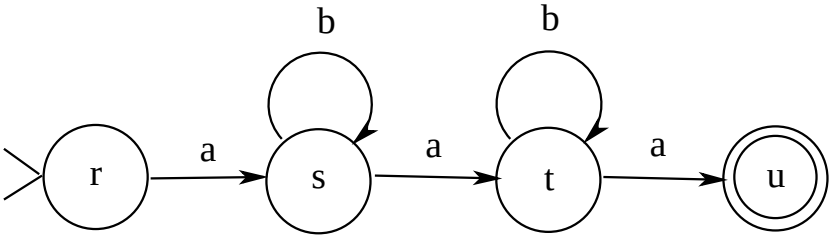
After processing t:



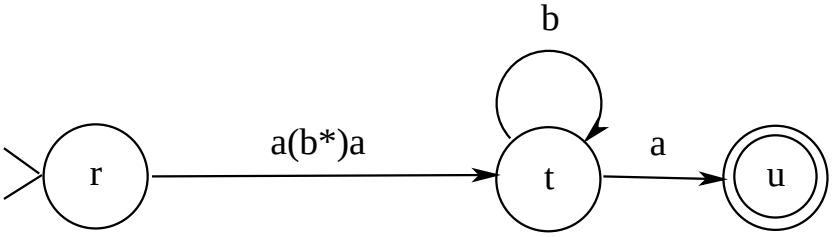
After collapsing arrows:



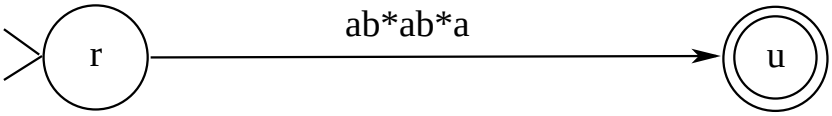
The final regular expression is $ab^* \cup ab^*a \cup c$. Now consider an example in which there are two states to eliminate.



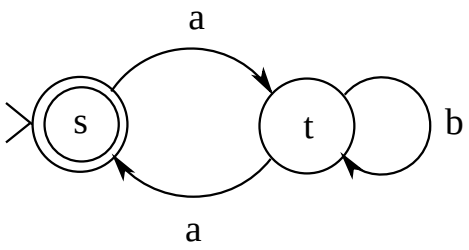
After eliminating state s:



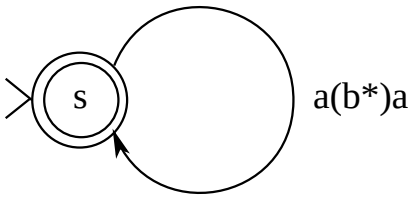
After eliminating state t:



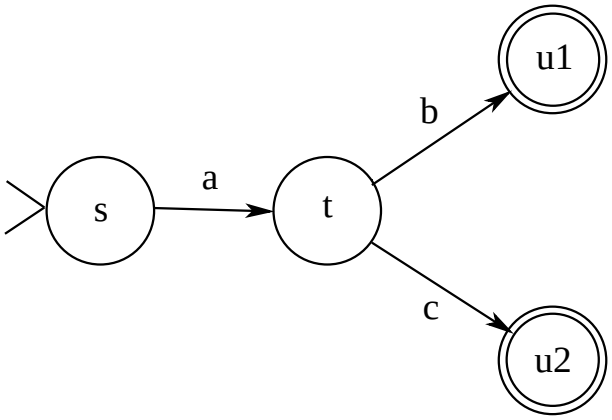
The final regular expression is ab^*ab^*a.
Now consider an example in which the states s and u are the same:



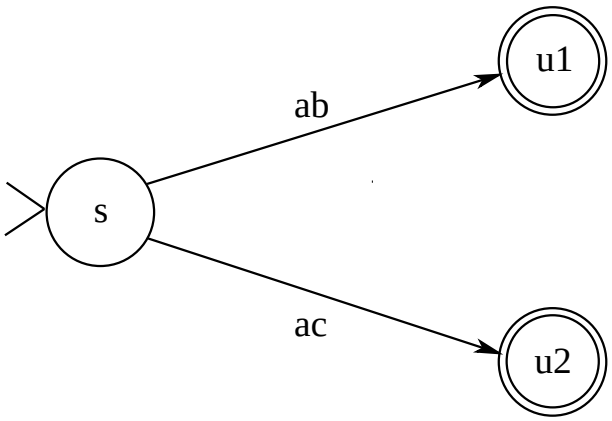
After processing state t:



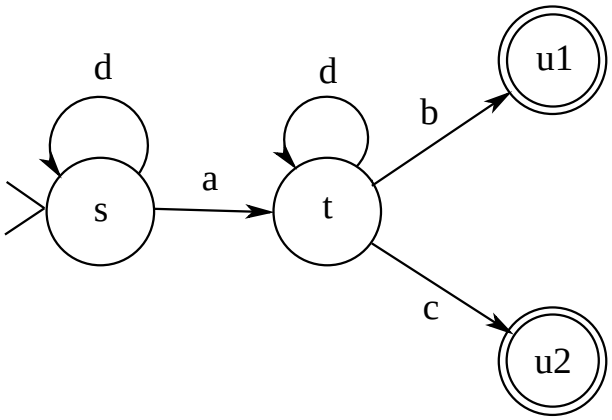
The final regular expression is $(ab^*a)^*$. Now consider an example with two states having arrows from t:



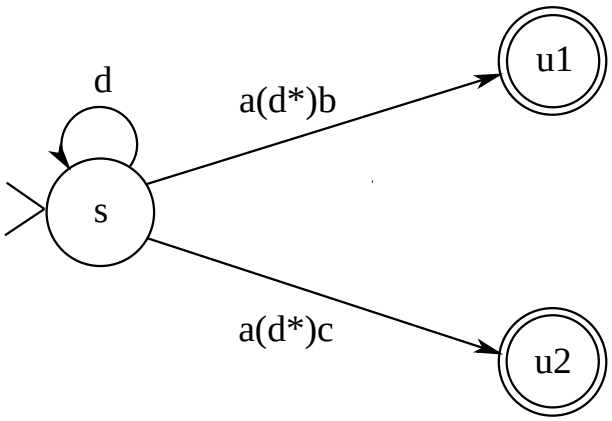
After processing state t, we have this automaton:



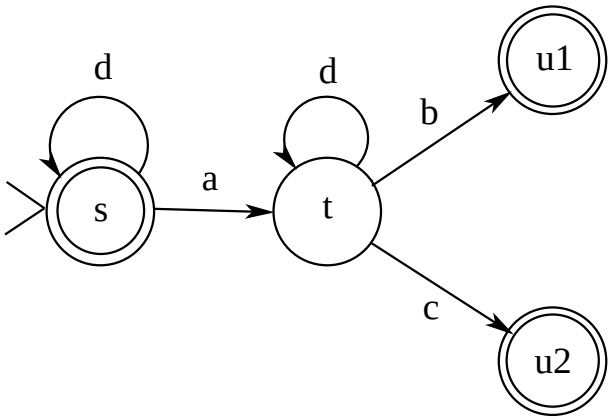
The final regular expression is $ab \cup ac$.
Now suppose there are more self-loops:



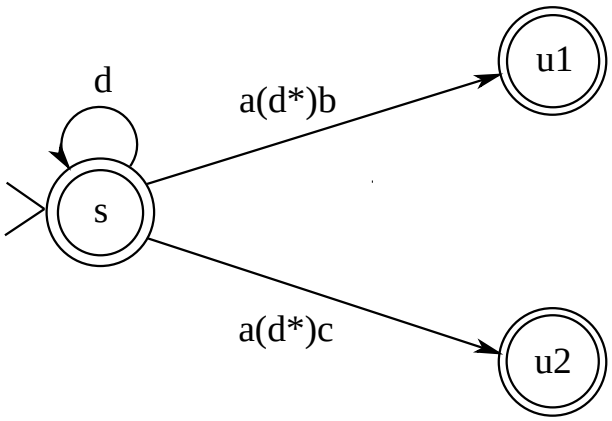
After processing state t, we have this automaton:



The final regular expression is $d^*(a(d^*)b \cup a(d^*)c)$.
Now suppose the start state is an accepting state:

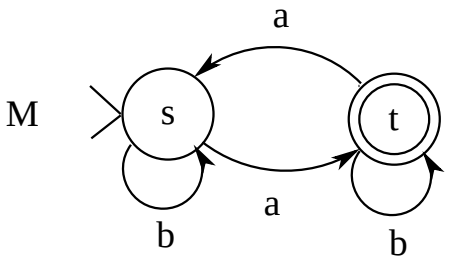


After processing state t, we have this automaton:



The final regular expression is $d^* \cup d^*(a(d^*)b \cup a(d^*)c)$.

Do this construction on the following automaton:



Putting all these results together, a language L is regular if and only if there is a finite automaton M such that $L = L(M)$.

Exercise: Find a regular expression for the set of strings having an odd number of a's and an even number of b's.

Exercise: Find a regular expression for the interview automaton.

Exercise: Suppose L_1 is regular and L_2 is non-regular. Is the concatenation $L_1 \circ L_2$ of L_1 and L_2 necessarily non-regular?

Exercise: Suppose $L \subseteq \Sigma^*$ is regular. Suppose $x \in \Sigma^*$. Let L' be $\{y : xy \in L\}$. Show that L' is regular.