We can express regular languages using regular expressions, which consist of just three operations (proof to follow): union, concatenation, and Kleene star.

- 01^* = strings starting with 0 followed by as many 1's as desired
- $a(a \cup b)^* c$ = strings starting with a, ending in c, and with as many a's or b's in between as desired.

Note that union is often written as the analagous | (or) from programming, i.e. $a(a \cup B)^* c = a(a|b)^* c$.
Similarly, we usually never write $\circ$ and leave it implicit.
Also, when we write 0 or a, these are shorthand for $\{0\}$ and $\{a\}$ since these are operations on sets.

Formal Definition of Regular Expressions

We say R is a regular expression if R is (on an alphabet Σ)

- a for some $a \in \Sigma$
- ε
- ϕ
- $R_1 \cup R_2$, where R_1, R_2 are regular expressions
- $R_1 \circ R_2$, " "
- R_1^*, where R_1 is a regular expression

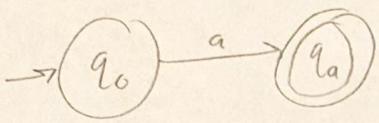
We now prove regular expressions characterizes the regular languages.

Thm) If a language is described by a regular expression, then it is regular.

Pf) Given a regular expression R, we construct an NFA N whose language $L(N) = R$ (recall R is a set of strings too). R falls into one of six cases.

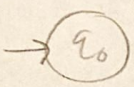
Technically, we would use structural induction, but we'll not bother

1) $R = a$ for some $a \in \Sigma$.



2) $R = \varepsilon$

3) $R = \emptyset$



4) $R = R_1 \cup R_2$

5) $R = R_1 \circ R_2$

6) $R = R_1^*$

For cases 4-6, we already constructed an NFA that accepts those languages (we assume R_1, R_2 have an equivalent NFA as they are smaller problems).

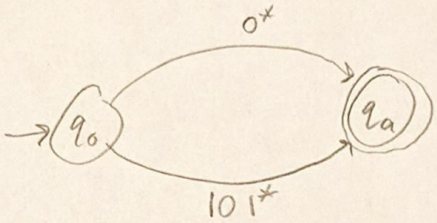
□

Thm) If a language is regular, it is described by a regular expression.

Pf (sketch) We sketch a proof here that is otherwise very technical despite its clear intuition.

A generalized nondeterministic finite automaton (GNFA) is an NFA except we allow REs on its transitions. These transitions consume a matching input, and like an NFA, the GNFA accepts an input if a path through it ends in an accepting state.

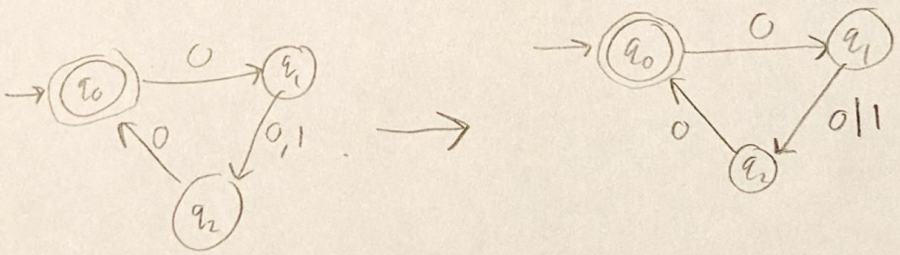
Ex)



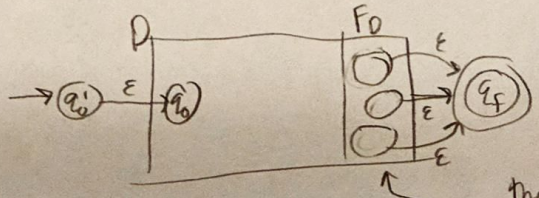
Accepts the language given by $0^* | 101^*$.

First note that a DFA is a GNFA. If a transition has more than one character, we union them together.

Ex)



Given a DFA D, we add 2 states to D, a new start state q_0' with an ε transition to the old state state q_0 and a new unique accepting state q_f with an ε transition to it from every old accepting state in F_D.



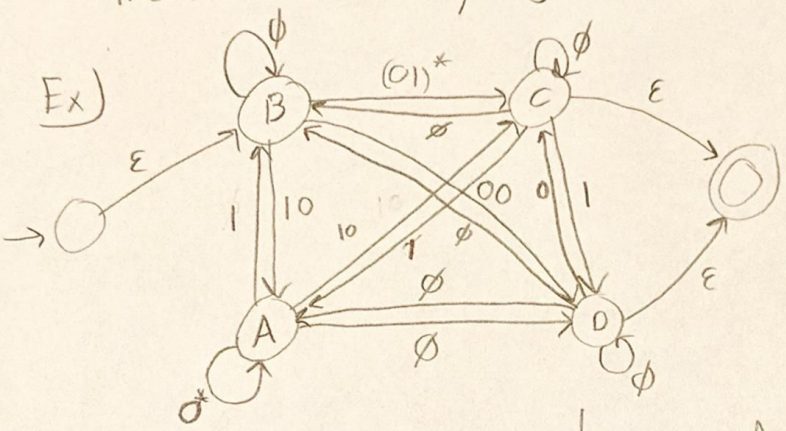
We add to D wherever a transition between 2 states is absent a transition with the RE $\emptyset$.

(not necessarily unique)

We now iteratively remove states from Q_D (the original set of states) until only the 2 new states are left. The transition between them will be the RE we want.

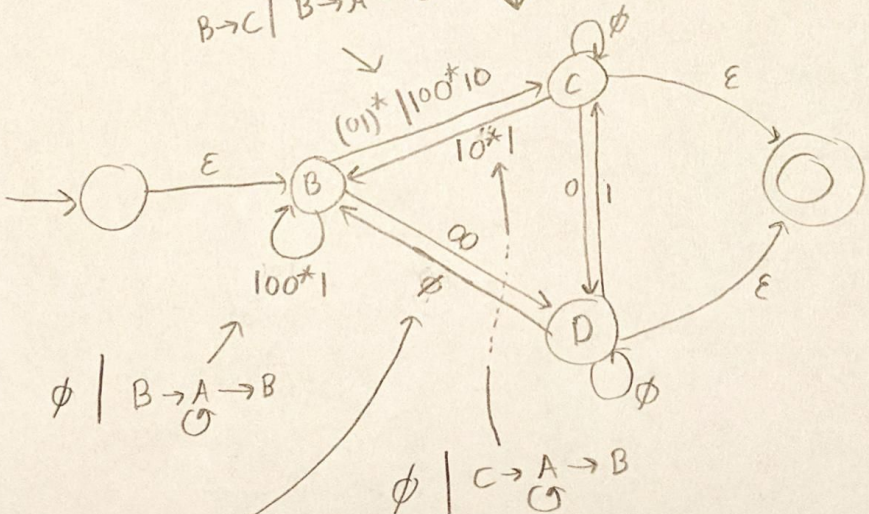
To do this, pick a state. We remove it from the graph. We then patch the dangling transitions by concatenating them along the appropriate path and unioning each path ending in the same state passing through the old vertex.

Ex)



Remove A

$B \to C \mid B \to A \to C$



$\emptyset \mid B \to A \to B$

$\emptyset \mid C \to A \to B$

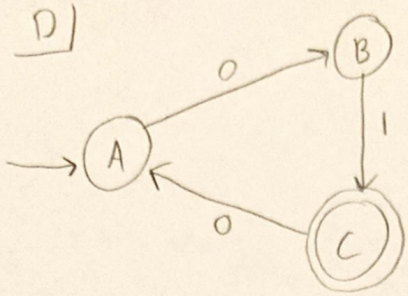
$D \to A \to B \mid D \to B$

Both have $\emptyset$ on the path, so the paths both produce $\emptyset$

When we reduce the GNFA to 2 state, as previously stated, we have the RE we want and we're done.

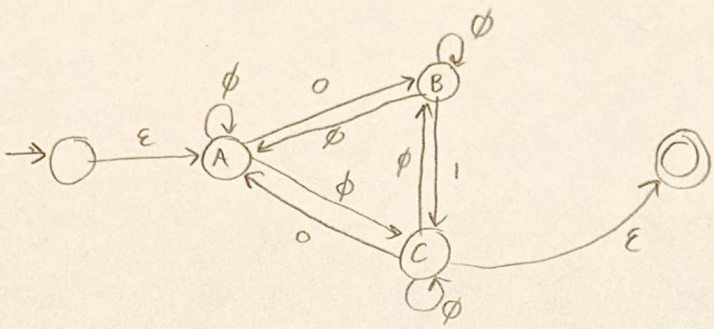
Ex)

D)

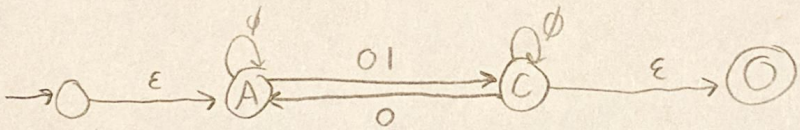


$L(D) = (010)^* 01$

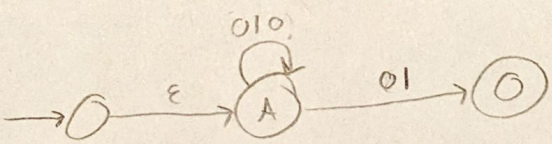
↓ Make GNFA in prescribed form



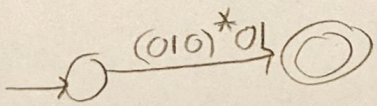
↓ Remove B



↓ Remove C



↓ Remove A



Lo' and behold, $L(D)$ = the RE $(010)^*01$.

□