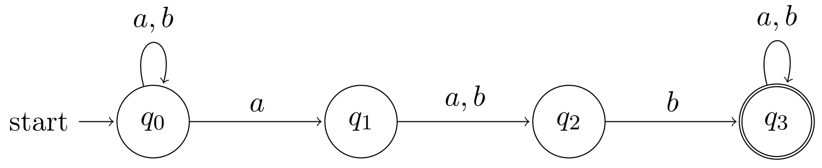
1 Let $L = \{w \in \{a, b\}^* \mid a$ appears in some position i of w, and a b appears in position $i + 2\}$.

1 A. Create an NFA N for L with at most four states.

Solution:

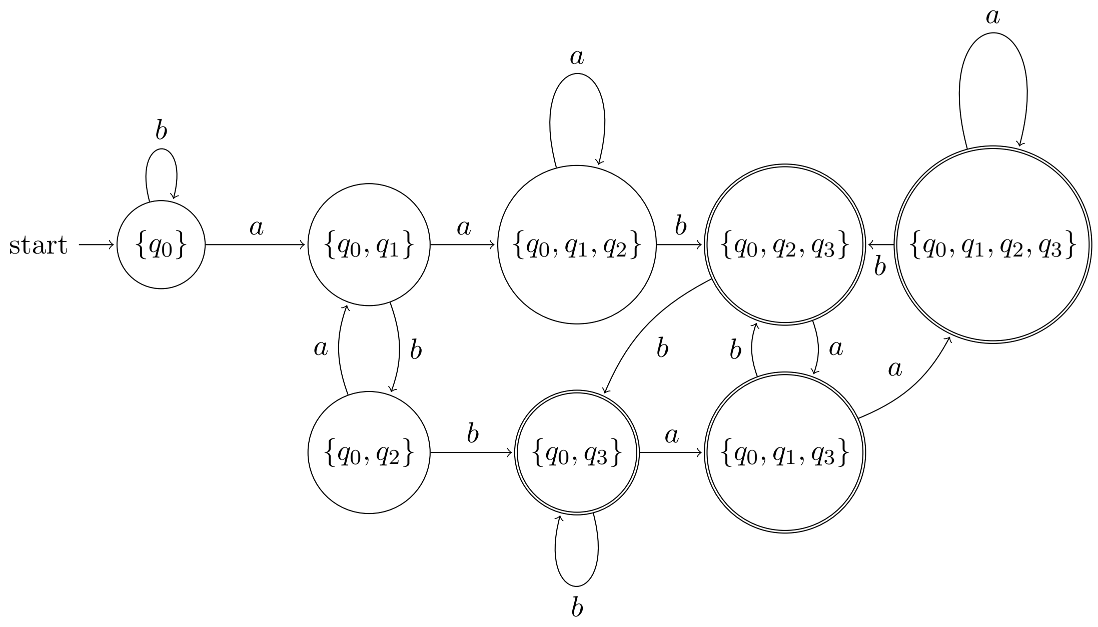
The following NFA N accepts the language. The machine starts at state q_0. On seeing the symbol a, the NFA has the choice of either staying at q_0 or to check if it is followed, 2 positions later, with a b.



1 B. Using the "power-set" construction, create a DFA M from N. Rather than writing down the sixteen states and trying to fill in the transitions, build the states as needed using "incremental subset" construction, because you won't end up with unreachable or otherwise superfluous states.

Solution:

Using the "power-set" construction, we obtain the following DFA M.



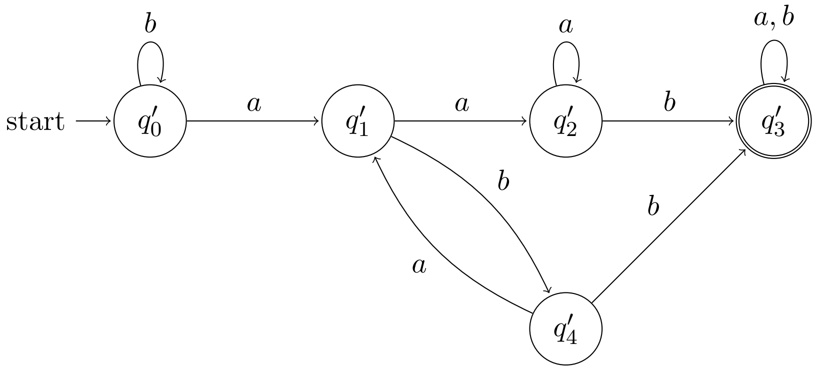
1 C. Now directly design a DFA M' for L with only five states, and explain the relationship between M and M'.

Solution:

The DFA M' is as follows. M' remembers the last two symbols seen so far.

- q'_0 is the start state. M'

- q'_1 corresponds to having seen ba as the last two symbols (or just a if this is the first symbol).
- q'_2 corresponds to having seen aa as the last two symbols.
- q'_3 is the accepting state.
- q'_4 corresponds to having seen ab as the last two symbols.



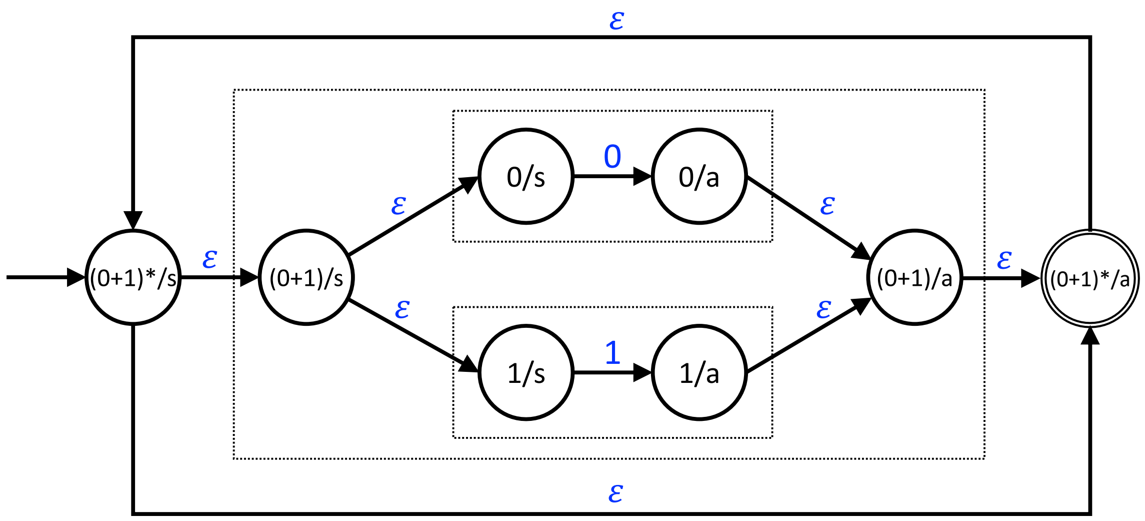
Note that if we contract all the accepting to states in M (from part (b)) to one state, then we obtain M'.

2

Use Thompson's algorithm to create an NFA for the following regular expressions:

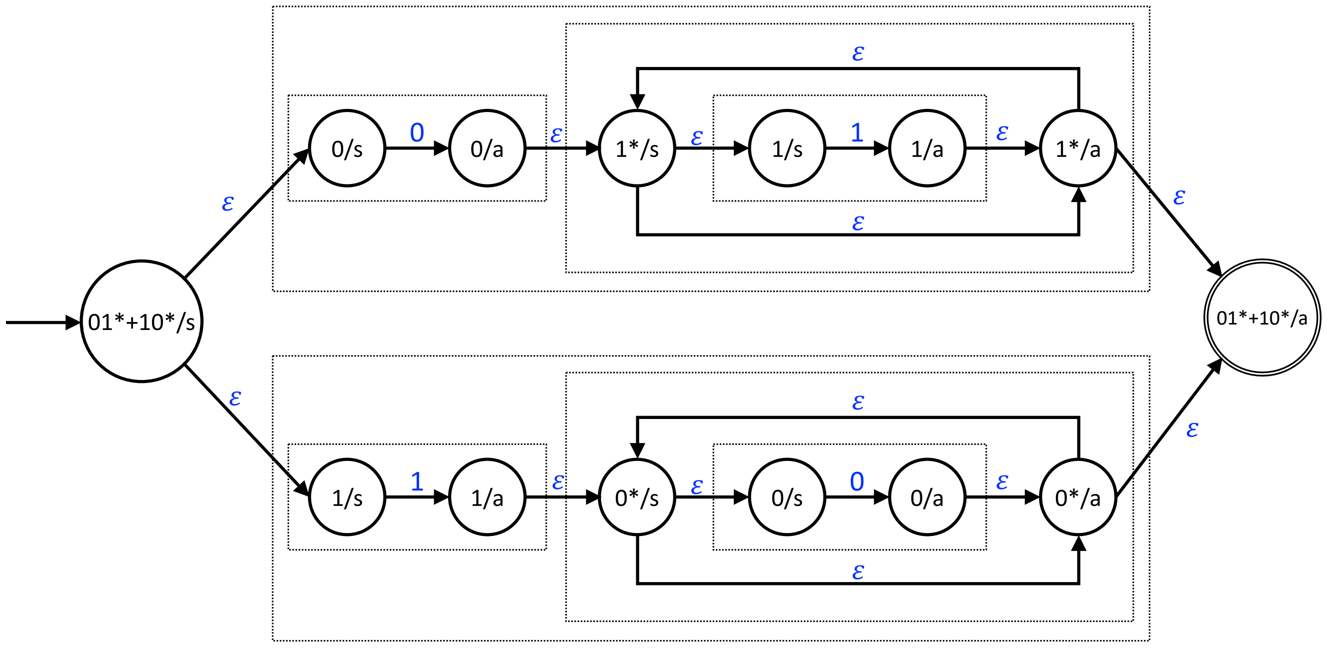
2.A. $(0+1)^*$

Solution:



2.B. $01^* + 10^*$

Solution:



3 Consider the following recursively defined function on strings:

$$stutter(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ aa \bullet stutter(x) & \text{if } w = ax \text{ for some symbol } a \text{ and some string } x \end{cases}$$

Intuitively, $stutter(w)$ doubles every symbol in w. For example:

- $stutter(PRESTO) = PPRREESSTTOO$
- $stutter(HOCUS␣POCUS) = HHOOCCUUSS␣␣PPOOCCUUSS$

Let L be an arbitrary regular language.

3.A. Prove that the language $stutter^{-1}(L) := \{w\}\, stutter(w) \in L$ is regular.

Solution:

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts L.
We construct an DFA $M' = (\Sigma, Q', s', A', \delta')$ that accepts $stutter^{-1}(L)$ as follows:

$$Q' = Q$$
$$s' = s$$
$$A' = A$$
$$\delta'(q, a) = \delta(\delta(q, a), a)$$

M' reads its input string w and simulates M running on $stutter(w)$. Each time M' reads a symbol, the simulation of M reads two copies of that symbol.

3.B. Prove that the language $stutter(L) := \{stutter(w)\}\, w \in L$ is regular.

Solution:

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts L.

We construct an DFA $M' = (\Sigma, Q', s', A', \delta')$ that accepts $\mathit{stutter}(L)$ as follows:

$$\begin{aligned}
Q' &= Q \times (\{\bullet\} \cup \Sigma) \cup \{\mathit{fail}\} \quad \text{for some } \bullet \notin \Sigma \\
s' &= (s, \bullet) \\
A' &= \{(q, \bullet)\}\ q \in A
\end{aligned}$$

$$\begin{aligned}
\delta'((q,\bullet), a) &= (q, a) \\
\delta'((q,a), b) &= \begin{cases} (\delta(q,a), \bullet) & \text{if } a = b \\ \mathit{fail} & \text{if } a \neq b \end{cases} \\
\delta'(\mathit{fail}, a) &= \mathit{fail}
\end{aligned}$$

M' reads the input string $\mathit{stutter}(w)$ and simulates M running on input w.

- State $(q, \bullet)$ means M' has just read an even symbol in $\mathit{stutter}(w)$, so M should ignore the next symbol (if any).
- For any symbol $a \in \Sigma$, state (q, a) means M' has just read an odd symbol in $\mathit{stutter}(w)$, and that symbol was a. If the next symbol is an a, then M should transition normally; otherwise, the simulation should fail.
- The state *fail* means M' has read two successive symbols that should have been equal but were not; the input string is not $\mathit{stutter}(w)$ for any string w.

Solution:

Let R be an arbitrary regular *expression*. We recursively construct a regular expression $\mathit{stutter}(R)$ as follows:

$$\mathit{stutter}(R) := \begin{cases} \varnothing & \text{if } R = \varnothing \\ \mathit{stutter}(w) & \text{if } R = w \text{ for some string } w \in \Sigma^* \\ \mathit{stutter}(A) + \mathit{stutter}(B) & \text{if } R = A + B \text{ for some regular expressions } A \text{ and } B \\ \mathit{stutter}(A)\ \mathit{stutter}(B) & \text{if } R = A\,B \text{ for some regular expressions } A \text{ and } B \\ (\mathit{stutter}(A))^* & \text{if } R = A^* \text{ for some regular expression } A \end{cases}$$

To prove that $L(\mathit{stutter}(R)) = \mathit{stutter}(L(R))$, we need the following identities for arbitrary *languages* A and B:

- $\mathit{stutter}(A \cup B) = \mathit{stutter}(A) \cup \mathit{stutter}(B)$
- $\mathit{stutter}(A \bullet B) = \mathit{stutter}(A) \bullet \mathit{stutter}(B)$
- $\mathit{stutter}(A^*) = \mathit{stutter}(A)^*$

These identities can all be proved by inductive definition-chasing, after which the claim $L(\mathit{stutter}(R)) = \mathit{stutter}(L(R))$ follows by induction. We leave the details of the induction proofs as an exercise for ~~a future semester~~ ~~an exam~~ the reader.

Equivalently, we can directly transform R into $\mathit{stutter}(R)$ by replacing every explicit string $w \in \Sigma^*$ inside R with $\mathit{stutter}(w)$ (with additional parentheses if necessary). For example:

$$\mathit{stutter}\big((1+\varepsilon)(01)^*(0+\varepsilon) + 0^*\big) \;=\; (11+\varepsilon)(0011)^*(00+\varepsilon) + (00)^*$$

Although this may look simpler, actually *proving* that it works requires the same induction arguments.

4 Consider the following recursively defined function on strings:

$$evens(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \varepsilon & \text{if } w = a \text{ for some symbol } a \\ b \cdot evens(x) & \text{if } w = abx \text{ for some symbols } a \text{ and } b \text{ and some string } x \end{cases}$$

Intuitively, $evens(w)$ skips over every other symbol in w. For example:

- $evens(EXPELLIARMUS) = XELAMS$
- $evens(AVADA␣KEDAVRA) = VD␣EAR$.

Once again, let L be an arbitrary regular language.

4.A. Prove that the language $evens^{-1}(L) := \{w\}\ evens(w) \in L$ is regular.

Solution:

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts L. We construct an DFA $M' = (\Sigma, Q', s', A', \delta')$ that accepts $evens^{-1}(L)$ as follows:

$$\begin{aligned} Q' &= Q \times \{0,1\} \\ s' &= (s, 0) \\ A' &= A \times \{0,1\} \end{aligned}$$

$$\begin{aligned} \delta'((q,0),a) &= (q,1) \\ \delta'((q,1),a) &= (\delta(q,a),0) \end{aligned}$$

M' reads its input string w and simulates M running on $evens(w)$.

- State $(q, 0)$ means M' has just read an even symbol in w, so M should ignore the next symbol (if any).
- State $(q, 1)$ means M' has just read an odd symbol in w, so M should read the next symbol (if any).

4.B. Prove that the language $evens(L) := \{evens(w)\}\ w \in L$ is regular.

Solution:

Let $M = (\Sigma, Q, s, A, \delta)$ be a DFA that accepts L. We construct an NFA $M' = (\Sigma, Q', s', A', \delta')$ that accepts $evens(L)$ as follows:

$$\begin{aligned} Q' &= Q \\ s' &= s \\ A' &= A \cup \{q \in Q\}\ \delta(q,a) \cap A \neq \varnothing \text{ for some } a \in \Sigma \end{aligned}$$

$$\delta'(q,a) = \bigcup_{b \in \Sigma} \{\delta(\delta(q,b),a)\}$$

M' reads the input string $evens(w)$ and simulates M running on string w, while nondeterministically guessing the missing symbols in w.

- When M' reads the symbol a from $evens(w)$, it guesses a symbol $b \in \Sigma$ and simulates M reading ba from w.

- When M' finishes *evens*(w), it guesses whether w has even or odd length, and in the odd case, it guesses the last character of w.