COMPUTATION: DAY 2

BURTON ROSENBERG
UNIVERSITY OF MIAMI
21 MARCH 2025

Contents

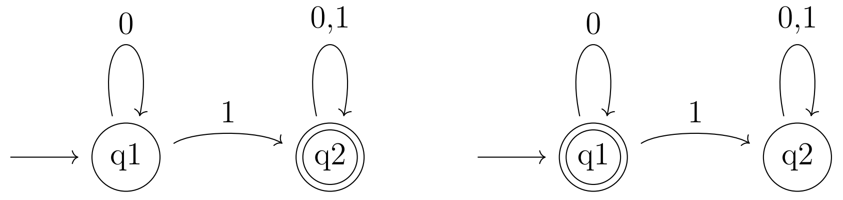
1. Finite Automata and Regular Languages

A finite automata is a 5-tuple,

$$M = \langle\, Q, \Sigma, \delta, q_0, F \,\rangle$$

where

- Q is a finite set of states,
- Σ is a finite alphabet set,
- $q_0 \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states and
- $\delta : Q \times \Sigma \to Q$ is the state transition function.



(A) At least one one. (B) Never any one's

FIGURE 1. Strings with at least one one and its complement language

The transition function δ directs a path through the states as allowed by a string $s \in \Sigma^*$,

$$s = \sigma_1 \, \sigma_2 \, \ldots \, \sigma_n$$

according to a sequence of states $q_j \in Q$,

$$q_i \xrightarrow{s} q_{i+n} = \langle \, q_i, q_{i+1}, \ldots, q_{i+n} \, \rangle$$

constrained by $\delta(q_{j-1}, \sigma_j) = q_j$. The finite accepts the stings s according to the rule,

$$M(s) = \begin{cases} T & q_0 \xrightarrow{s} q_f \text{ with } q_f \in F \\ F & \text{else} \end{cases}$$

where q_0 is the start state of the finite automata.

Definition 1.1. A set $S \subseteq \Sigma^*$ is a *regular language* if there exists a Finite Automata M such that $\big(s \in S\big) \equiv M(s)$.

Example: The language of strings over 0 and 1 with at least one 1 is regular. The finite automata that accepts exactly such string is show in figure 1a. Note that by exchanging the set of final and non-final states, the machine recognizes the *complement* language — the strings that contain no 1's, including the empty string.

Theorem 1.1. If $S \subseteq \Sigma^*$ is regular, so is the complement of S in Σ^*.

Exercise 1.1: Show that the set $\emptyset$ is a regular language. That is, give a machine M that accepts nothing, not even the empty string.
Exercise 1.2: Show that the set $\{ \, \epsilon \, \}$ is a regular language. That is, give a machine M that accepts only the empty string.
Exercise 1.3: Show that the set Σ^* is a regular language. That is, give a machine M that accepts everything.
Exercise 1.4: Show that the set $\{ \, s \in \Sigma^* \, | \, |s| > 0 \, \}$ is a regular language. That is, give a machine M that accepts any string but the empty string.

2. Regular Languages Closure Properties

A set is a mathematical object having a membership predicate. A set defined in any fashion that is sufficiently clear to determine membership. Given sets X, Y and U such that $X \subseteq U$, these new sets are obviously allowed, as the construction of a membership predicate follows easily,

$$\begin{aligned} X \cup Y &= \{ z \,|\, z \in X \text{ or } z \in Y \} \\ X \cap Y &= \{ z \,|\, z \in X \text{ and } z \in Y \} \\ X \setminus U &= \{ u \in U \,|\, u \notin X \} \\ X \times Y &= \{ (x,y) \,|\, x \in X \text{ and } y \in Y \} \end{aligned}$$

These are called, respectively, the *union*, *intersection*, *complement* and *cartesian product.*

While all of these should be familiar constructions, the *Kleene Star* is very important for us, and maybe be less familiar,

$$X^* = \emptyset \cup X \cup (X \times X) \cup (X \times X \times X) \cup \ldots = \bigcup_{i=0}^{\infty} X^i$$

We do not doubt the existence of such a set, because for an x of length i, we already have the predicate for $x \in X^i$.

The *free algebra on the set* Σ has as values sets of strings $\sigma \in \Sigma^*$, including the empty string ε where union of elements is addition and concatenation is multiplication. The empty string is the unit for multiplication, and the empty set is the null element for union. With these notions we have the concatenation of sets (elements of the free algebra),

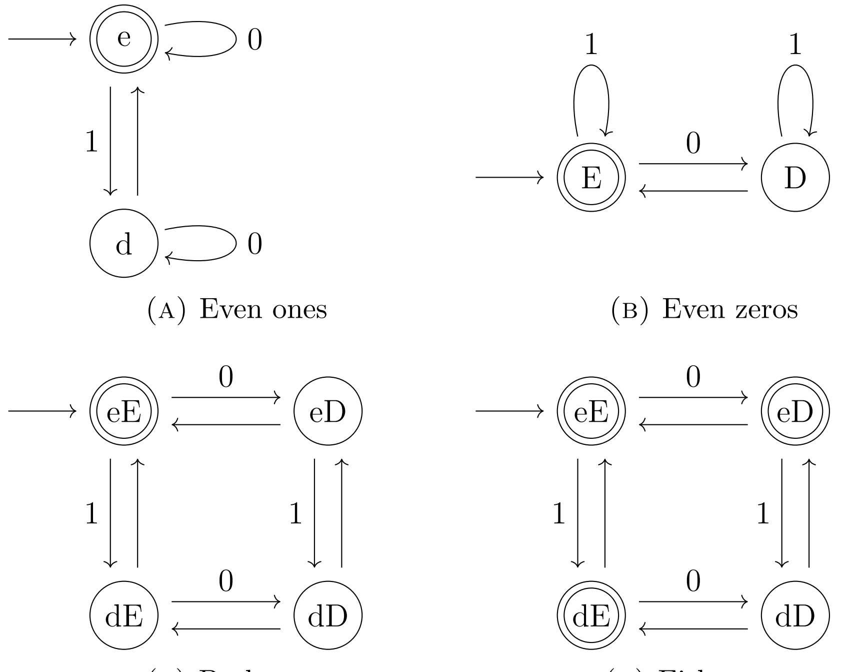
$$S \circ T = \{ s\,t \,|\, s \in S, t \in T \}$$

Regular languages are closed by these operations. We have already seen that if S is a regular language then $\Sigma^* \setminus S$, the complement of S, is a regular language. So far, to prove a set regular, one must present a finite automata accepting exactly the elements in the set. With closure properties a powerful new tool is available.

For instance, a singleton set is a regular language. Therefore by the closure by union, we have that any finite set is a regular language.

Our plan is to prove union and intersection closure laws by introducing the product machine. And to prove closure by concatenation and Kleene star by introducing a nondeterministic version of the finite automata.

Exercise 2.1: Show that the set $\{ s \}$ for an $s \in \Sigma^*$ is a regular language.



(A) Even ones (B) Even zeros

(C) Both even (D) Either even

FIGURE 2. Even and odd parity of 1's and 0's

3. The product machine construction

Theorem 3.1. If $S, T \subseteq \Sigma^*$ are regular languages, so are $S \cup T$ and $S \cap T$.

Proof: This is a proof by example, and I owe this nice example and the graphic depiction to Wayne Goddard of Clemson University. The languages,

$$\begin{aligned} L_1 &= \{\, \sigma \in \{\, 0, 1\,\}^* \mid \text{ there are an even number of 1's in } \sigma \,\} \\ L_2 &= \{\, \sigma \in \{\, 0, 1\,\}^* \mid \text{ there are an even number of 0's in } \sigma \,\} \end{aligned}$$

are regular, with the machine M_1 in figure 2a such that $\mathcal{L}(M_1) = L_1$ and machine M_2 in figure 2b such that $\mathcal{L}(M_2) = L_2$.

We describe the construction of M_3 shown in figure 2c such that,

$$\mathcal{L}(M_3) = L_1 \cap L_2,$$

which would prove that the language $L_1 \cap L_2$ is regular.

A string is in both languages when the two machines that recognize the languages are started together, and move together on a letter by letter basis, arriving both in a final state. The two machine conceptually run in parallel. We simulate the pair of machines with a single machine whose state set is the cartesian product of the state sets of the two machines,

$$Q_3 = Q_1 \times Q_2.$$

Then for any $q \in Q_3$, it is written as $q = (q_1, q_2)$, and means that in a parallel execution, M_1 would be in q_1 and M_2 would be in q_2.

The transition function folds into a single machine the transitions of the two component machines,

$$\delta_3(q) = \delta_3((q_1, q_2)) = (\delta_1(q_1), \delta_2(q_2)) = q'.$$

and the accepting states are only those where, when written in the two components, each the first and second component are accepting for their respective machines,

$$F_3 = F_1 \times F_2$$

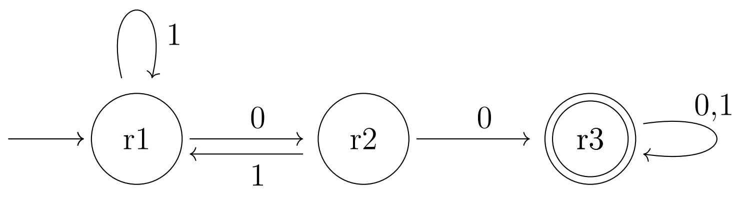
And that's the construction for the intersection.

For the union, all is the same except we accept if at least one of the first or the second component is accepting in it's respective machine,

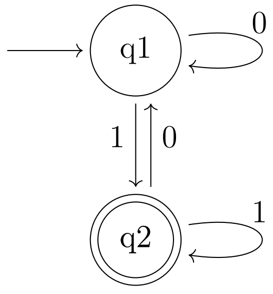
$$F_4 = (F_1 \times Q_2) \cup (Q_1 \times F_2).$$

and this is the machine of figure 2d. □

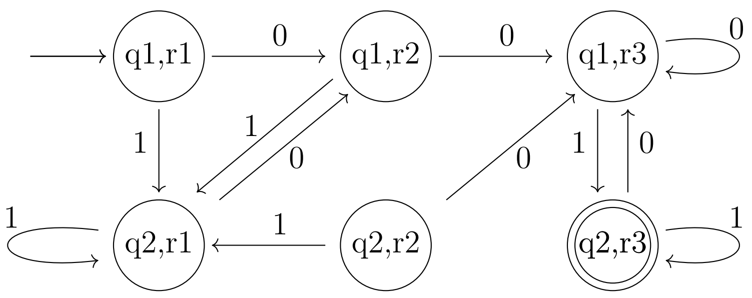
3.1. **Second example of the product machine construction.** Let S_1 be the set of strings containing 00, and S_2 be the set of strings ending in a 1. The product construction is given accepting the language $S_1 \cap S_2$.



Accepts strings over $\{0, 1\}$ containing the sequence 00.



Accepts strings over $\{\,0,1\,\}$ ending with a 1.



The product of the above two machines, with final state to accept the intersection.

3.2. **Discussion on intersection or union.** The machines produced by the product construction is a general construction, that can be performed mechanically. Think of a machine transformer $\mathcal{T}$ that takes machine arguments and gives a machine result,

$$\mathcal{T}(M_1, M_2) = M_3 \text{ such that } \mathcal{L}(M_3) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2).$$

because this transformer is the proof.

While completely general, the construction has many more states than either of the starting machines. The number of the states is the product of the number of states of the two machines. If we want to accept strings with opposite parity of 0's and 1's, this could be accomplished final state set $\{\,eD, dE\,\}$. However, there is another approach.

Theorem 3.2. The language

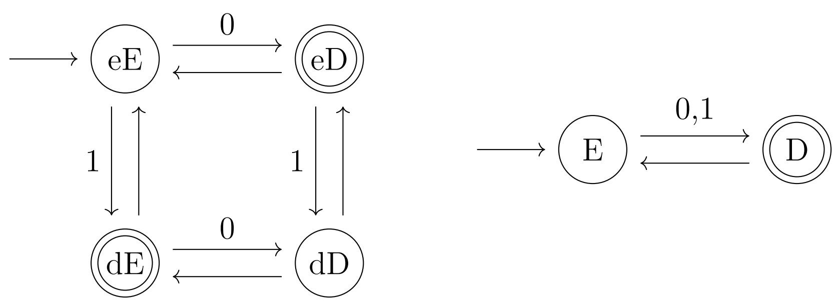
$$\{\,\sigma \in \{\,0,1\,\} \,|\, \text{parity of 1's differs from parity of 0's}\,\}$$

and be recognized by a machine with two states.

Proof: Let l by the length of σ, l_1 be the number of 1's in σ, and l_0 be the number of 0's in σ. Then from,

$$l = l_0 + l_1$$

we conclude that if l is odd, then one of l_0 and l_1 is odd and the other even, so σ is in the language. However if l is even then l_0, l_1 are both even or both odd, so σ is not in the language. So the language is the same as the language of the parity of the length of σ. □



(A) Using 4 states (B) Using 2 states

FIGURE 3. Parity of 1's and 0's differ

4. NONDETERMINISM

The nondeterministic machine allows the machine to consider alternatives in how it continues its computation. It is an interesting idea with applications across many types of algorithms. Probabilistic algorithms uses the alternatives by tossing a coin, and the program's result becomes a random variable. The result is now a function of input and the coin tosses. The output has a probability distribution inherited from the probability distribution of the randomness and how it is used.

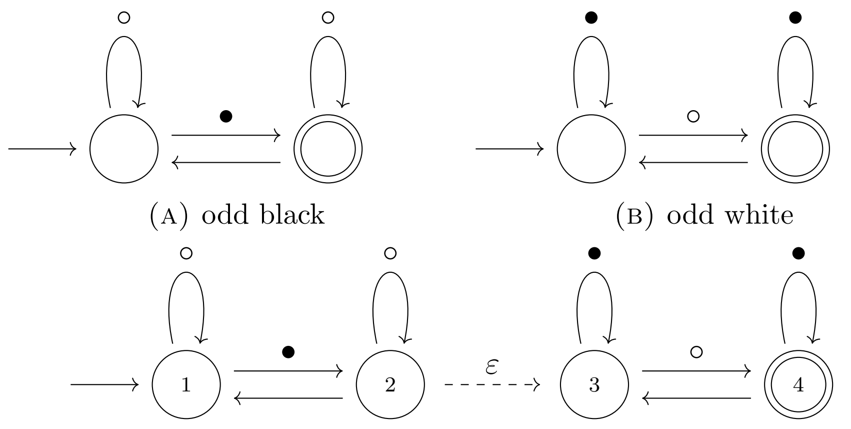
In finite automata, the result is still binary —a string is either in a set or not in a set. However a guess and check mindset appears. Rather than proving that a string is in a set, a alleged proof is given to the automata and the automata verifies that the proof is correct. Note however, that for finite automata, the nondeterminism is only a shortcut. We shall introduce and use nondeterministic machines and then show that any set recognized by a nondeterministic finite automata can be recognized by a deterministic finite automata. However, the deterministic automata might use a great deal more states.

There are two elements to nondeterminism, ε edges and multiple paths. We introduce these one at a time.

4.1. **Nondeterminism: ε edges.**

Theorem 4.1. If $S, T \subseteq \Sigma^*$ are regular languages, so is $S \circ T$.

The proof of this theorem is added by the introduction of a curious sort of machine called the *nondeterministic finite automata*. The motivating example will be the



(A) odd black (B) odd white

(C) concatenation finite automata

FIGURE 4. Non-deterministic solution with an ε move

concatenation of the two languages,

$$\begin{aligned} S &= \{\, \sigma \in \{\, \bullet, \circ \,\} \mid \text{ string } \sigma \text{ has an odd number of } \bullet \,\} \\ T &= \{\, \sigma \in \{\, \bullet, \circ \,\} \mid \text{ string } \sigma \text{ has an odd number of } \circ \,\} \\ R &= S \circ T \end{aligned}$$

and the string,

$$\rho = \bullet \circ \circ \bullet \circ \bullet \circ \bullet \circ \circ \bullet$$

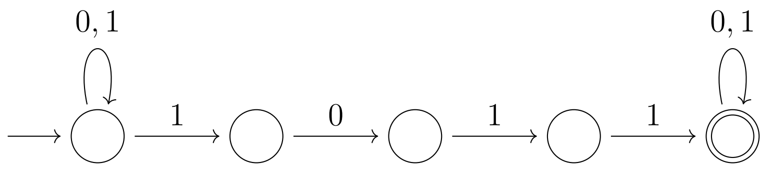
and $\rho \in R$ because,

$$\rho = (\bullet\circ)(\circ \bullet \circ \bullet \circ \bullet \circ \circ \bullet) = (\bullet \circ \circ \bullet \circ\bullet)(\circ \bullet \circ \circ \bullet)$$

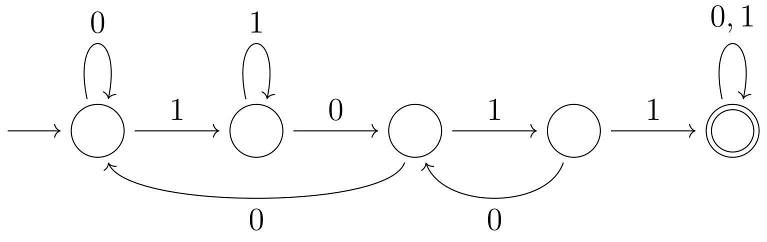
but for no other subdivisions of ρ into two strings. The question for a finite state automata is how is it to know where to parse ρ.

The machines are connected by a special edge marked with the symbol ε that does not consume a character and is non-deterministically followed, in fact followed at exactly the right moment in the computation to provide a proof that a string is in the language. For instance, the machine would opt to take the ε edge after the first white item, or would opt to take the ε edge immediately after the third black item.

Note as well, if the string is not in the language the ε edge will not cause the string to be accepted inappropriately. The computation before the use of the edge must be a string in S and the computation must continue after the use of the edge to demonstrate the remainder of the string is in T.



(A) non-deterministic



(B) deterministic

Figure 5. Matching the pattern 1011, using non-determinism

4.2. **Nondeterminism: multiple edges.** The non-determinist finite automata we are imagining can also have multiple outgoing edges from a state labeled with the same letter. In this case, the non-deterministic choice is made on which edge to take.

In figure 5a is a non-deterministic machine for matching the pattern 1011 anywhere in a string. The assuming the pattern is in the string, the machine non-deterministically guesses its location. It loops at the start state until reaching the start of the pattern and then proceeds with the four state transitions, ending in an absorbing final state.

If the pattern is not in the string, any attempt to pass from the start to the final state will get to a failure to pass to the next state, formally said, zero outgoing arrows for the letter, and that branch of the computation will fail.

A deterministic machine for the same language is given in figure 5b. It might come as a surprise how tricky the transition function is.

5. Formal definition of nondeterministic finite automata

Definition 5.1. A *nondeterministic finite automata* is a five tuple $\langle\, Q, \Sigma, \delta, q, F \,\rangle$ where Q is a finite set of states, Σ is a finite alphabet of a free algebra, $q \in Q$ is the start state and $F \subseteq Q$ is the set of accepting or final states, and,

$$\delta \;:\; Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$$

is the transition function from states and the input alphabet augmented by the symbol for the empty string of the free algebra,

$$\Sigma_\varepsilon = \Sigma \cup \{\, \varepsilon \,\}$$

(where the symbol ε is assumed to be a fresh symbol, not appearing in Σ) and $\mathcal{P}(Q)$ is the set of subsets of Q.

Definition 5.2. A computation of nondeterministic machine M on a string $\sigma \in \Sigma^*$ is a sequence of states

$$\langle\, q \,\rangle = q_1, q_2, \ldots, q_n \text{ with } q_i \in Q$$

and a $\sigma' \in \Sigma_\varepsilon^*$,

$$\sigma' = \sigma_1, \sigma_2, \ldots, \sigma_n \text{ where } \sigma' = \sigma$$

such that for every applicable i,

$$q_{i+1} \in \delta(q_i, \sigma_i).$$

We denotate this computation as $q_1 \xrightarrow{\sigma} q_n$.

Definition 5.3. The language of the nondeterministic finite automata $M = \langle\, Q, \Sigma, \delta, q_s, F \,\rangle$ is,

$$\mathcal{L}(M) = \{\, \sigma \in \Sigma^* \,|\, \text{there exists a computation } q_s \xrightarrow{\sigma} q_f \text{ with } q_f \in F \,\}$$

5.1. **Asymmetry of accepting and non-accepting.** A deterministic machine has only one computation path that serves two ends — it shows when a string is accepted and when a string is rejected.

A non-deterministic machine only demonstrates when a string is in the language, by providing a computation path that accepts the string. In general, there will be no single computation that rejects the string.

5.2. **Non-constructive language.** A deterministic machine is a recipe for creating a computation. The non-deterministic machine simply claims that *there exists* a computation, without asking how that computation was chosen.

5.3. **DFA's are NFA's too.** A deterministic finite automata (DFA) is a nondeterministic finite automata (NFA) when the NFA has the following two qualities,

(1) There are no ε edges, $\forall q \in Q,\ |\, \delta(q, \varepsilon) \,| = 0$.
(2) There is exactly one choice for any other edge, $\forall q \in Q, \sigma \in \Sigma, |\delta(q, \sigma)\,| = 1$.

Hence NFA's can recognize all regular languages.

6. The power set construction

An NFA can recognize all regular languages. We now set out to show that any language recognized by an NFA is regular. This is shown by building an DFA from the NFA that simulates in one computation path the trajectory of multiple computation paths in the NFA.

The diagrams for a DFA computation consisted of a single token, initially placed on the start node and moved along edges from node to node according to the consumed letter. If we allow tokens to be created and discarded, so that many tokens can be placed on many nodes, we can visualize the collection of all possible computations each step of the way. As the token meets several possible next moves, it spawns new tokens and takes them all.

We can then build a DFA where the states are the configuration of tokens on the NFA. That is, each state in the simulating DFA represents a subset of the states of the NFA — those with tokens on them. With the NFA,

$$N = \langle\, Q, \Sigma, \delta, q_o, F \,\rangle$$

we construct a DFA,

$$M = \langle\, Q', \Sigma, \delta', q_o', F' \,\rangle$$

Where $Q' = \mathcal{P}(Q)$, the power set of Q, that is the set of all subsets of Q.

Given a state $q' \in Q'$ it corresponds to a set of states Q' that have tokens on them in N. Given a letter σ, each token $q' \in Q'$ removed and replaced with tokens on all of $\delta(q', \sigma)$. For a deterministic machine, this is not a set but a single token, so the effect is the token "moves" to the prescribed state. In the case of a non-deterministic machine, the old token is discarded and new tokens a placed on each of several states, or no state. If $\delta(q', \sigma) = \emptyset$ then this computation path had ended.

Hence, the totality of new tokens given in response to letter σ with tokens on all sets in Q' is,

$$R = \bigcup_{q \in Q'} \delta(q, \sigma).$$

But we need to consider ε edges as well. To follow all possible computation paths, it must be that if the set Q' contains a state q', and there are ε edges leaving q', tokens must all be on everything in $\delta(q', \varepsilon)$. This brings us to the definition of the ε-closure operator,

Definition 6.1. Given an NFA $M = \langle\, Q, \Sigma, \delta, q_o, F \,\}$ and a set of states $S \subseteq Q$, the *ε-closure of S*,

$$E(S) : \mathcal{P}(Q) \to \mathcal{P}(Q)$$

is the smallest subset of Q such that,

(1) $S \subseteq E(S)$,
(2) and $\forall\, q \in E(S),\ \delta(q, \varepsilon) \subseteq E(S)$.

It is called a closure operator because $E(E(S)) = E(S)$.

Theorem 6.1. The ε-closure operator commutes with set union, $E(A \cup B) = E(A) \cup E(B)$.

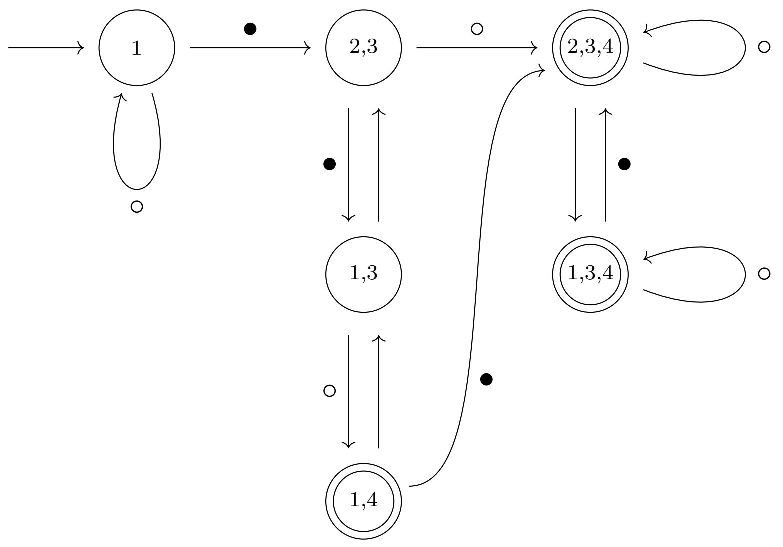


FIGURE 6. Power construction applied to figure 4c

Theorem 6.2. Given a nondeterministic finite automata $N = \langle\, Q, \Sigma, \delta, q_o, F \,\rangle$ the deterministic finite automata $M = \langle\, Q', \Sigma, \delta', q_o', F' \,\rangle$ simulates N, where the state set, the initial and final states are,

$$\begin{aligned} Q' &= \mathcal{P}(Q) \\ q_o' &= E\big(\{\, q_o \,\}\big) \\ F' &= \{\, q' \in Q' \,|\, F \cap q' \neq \emptyset \,\} \end{aligned}$$

and the transition function for $q' \in Q'$ is,

$$\delta'(q', \sigma) = E\left(\bigcup_{q \in q'} \delta(q, \sigma) \right)$$

Proof: It is best that the reader think this through for themselves. □

Despite being a lower case letter, q' is a set. In M it is a state, but it is a state representing a set of states in the machine N. Since $q' \in \mathcal{P}(Q)$ then any $q \in q'$ is a $q \in Q$.

7. EXAMPLES OF THE POWER SET CONSTRUCTION

The power set construction is applied to NFA of 4c resulting in the DFA of 6. There are some interesting discoveries in the result. Although there are 16 states in the DFA, only 6 are shown, for only those matter. The rest are configurations of tokens that are never achieved in normal operation.

Also, the two leftmost states could be collapsed into a single state without a change to the language, as they are both final states. This means that there are certain prefixes that place the string in the language not matter what follows. The state diagram shows that the simplest such prefix is $\bullet \circ$. The state diagram also shows that there are arbitrarily long strings containing both colors that are not in the language.