CS 378 Intro to Theory of Computation **Fall 2015**

Chapter 1: Regular Languages

Professor Dale Skrien *Zach Schutzman*

NB: *These notes are a revised version of those taken during Dale Skrein's CS 378 course at Colby College in Fall 2015. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

1.1 Languages

Theory of Computation is the study of models of computation and what kind of problems those models can and cannot solve. We will build from the simplest models of computation to more complex ones, until we reach models that can solve the same problems as modern computers.

Definition 1.1 *An **alphabet** is a finite set of characters.*

In this course, we will commonly use as our alphabet $\{0,1\}$ and $\{a,b\}$ in consideration of modern computing being built on binary representations of data. Sometimes, however, we might want to consider an alphabet with more symbols, such as $\{0,1,2\}$, or the entire English alphabet $\{a,b,c,d\ldots z\}$ or an extended unicode alphabet. We usually denote our alphabet Σ. We denote the set of all strings on the alphabet Σ^*. More about the * operator will come when we learn about regular expressions.

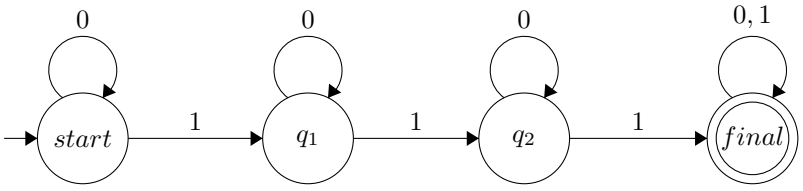
Definition 1.2 *A **string** is an ordered collection of characters from Σ. A string is finite in length.*

Definition 1.3 *A **language** is a subset of Σ^*. Equivalently, a language is a set of strings on an alphabet Σ.*

Just for clarity's sake, let's be explicit: Σ is a finite set. Σ^* is a countably infinite set of finite sets. An individual language is some subset of Σ^*, so it is at most countable. The number of languages on an alphabet Σ is equal to the size of the power set of Σ^*. This set is uncountably infinite.

1.2 Deterministic Finite Automata

Consider the following directed graph.



Given some string w on $0,1$, we begin at the start state and read each character of w one at a time, following the appropriate arrow to the next state (possibly the same one).

Definition 1.4 *An automaton* ***accepts*** *a string if and only if stepping through the automaton ends in a* ***final*** *or* ***accept state****. The language accepted by an automaton is exactly the set of strings on Σ whose traversals end in a final state. We say that an automaton* ***rejects*** *any string it does not accept. For an automaton M, we denote the language that M accepts as $\mathcal{L}(M)$.*

The above automaton accepts all strings on Σ that have at least three 1s.

Let's formalize this kind of automaton.

Definition 1.5 *A* ***deterministic finite automaton****, or DFA, is formally a five-tuple $(Q, \Sigma, \delta, q_0, F)$*

Q is the set of states, and is finite.

Σ is an alphabet.

δ is the transition function, which maps the current state and character to the next state.

q_0 is the start state and is an element of Q.

F is the set of final states and is a subset of Q.

Definition 1.6 *A language is* ***regular*** *if an only if there exists some DFA which accepts it.*

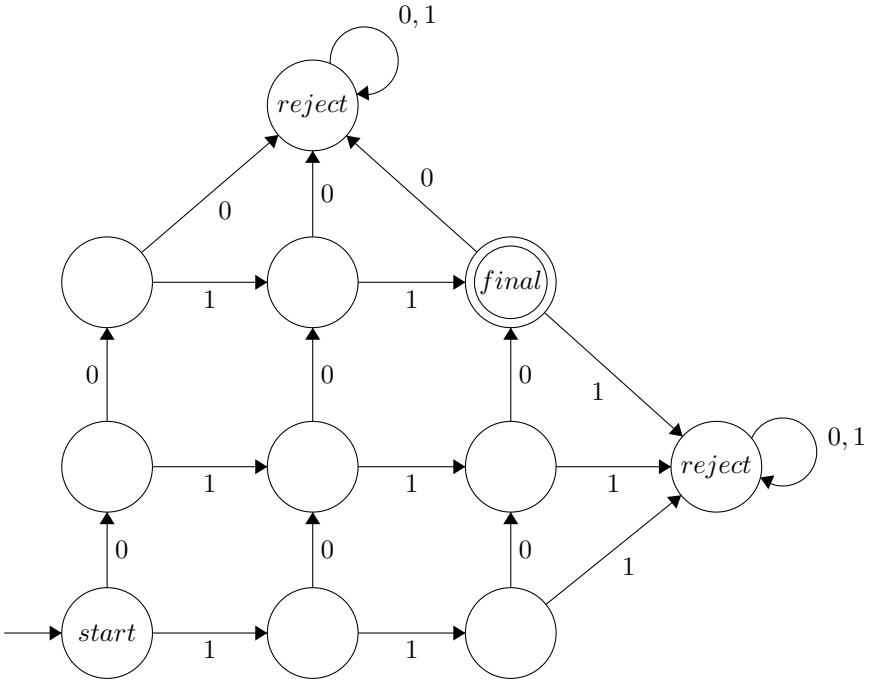
This is our first (of many) definition of a regular language. Now, the question arises of whether every language is regular (obviously not, or this class wouldn't have much to talk about).

Claim 1.7 *There exist non-regular languages.*

Proof: *Recall that we showed that the number of languages on an alphabet Σ is uncountably infinite. Note that in the definition of a DFA, we have a finite set of states Q, and a finite alphabet Σ, so δ must be finite as well, and all can be indexed by the natural numbers. Therefore, the number of possible DFAs is in bijection with $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ and is therefore countable. Therefore, there exist (uncountably many) non-regular languages.* ■

By the end of the chapter, we will have identified and constructed some non-regular languages.

Drawing the state diagram for a DFA is not always practical or feasible. Consider the language on $0, 1$ where each character appears exactly 100 times. Because DFAs read their inputs exactly once, this DFA requires over $10,000$ states to define. Below is a DFA accepting the language of strings where each character appears exactly twice.



The two states labeled reject are often called ***sinks****. Any string whose traversal enters a sink is not in* $\mathcal{L}(M)$*.*

Even if we cannot explicitly draw such a DFA, it is not hard to precisely describe what it might look like and assert that it exists. A formal proof that the language of strings that have exactly x_i *occurrences of character* i *proceeds by induction on the* x_i*. For two characters, the DFA looks rectangular, for 3 characters it looks like a cube, and beyond that it becomes hard to visualize. Inductive constructions like these will prove very useful moving forward.*

Claim 1.8 *The* ***empty language****, denoted* $\emptyset$*, and* Σ^* *are regular languages.*

Proof: *The empty language is the empty set of strings on an alphabet. Any DFA with no accept states by definition does not accept any strings, and therefore accepts the empty language. Similarly, a DFA where all states are accept states must accept every string on the alphabet, which is the definition of the set* Σ^**.* ■

Definition 1.9 *The* ***union*** *of two languages* A *and* B*, denoted* $A \cup B$*, is the set of strings* x *such that* $x \in A$ *or* $x \in B$ *(or both). This works just like the definition of set union.*

Definition 1.10 *The* ***intersection*** *of two languages* A *and* B*, denoted* $A \cap B$*, is the set of strings* x *such that* $x \in A$ *and* $x \in B$*. (Same as set intersection).*

Definition 1.11 *The* ***symmetric difference*** *of two languages* A *and* B *is the set of strings* x *such that* $x \in A$ *or* $x \in B$ *but* $x \notin A \cap B$*. Symbolically, the symmetric difference is* $A \cup B \backslash A \cap B$*. (Follows from definition of symmetric difference of two sets.)*

Definition 1.12 *The* ***reverse*** *of a language* $\mathcal{L}$*, denoted* $\mathcal{L}^R$ *is the set of strings* $x^r = x_k x_{k-1} \dots x_1$ *such that* $x = x_1 x_2 \dots x_{k-1} x_k$ *is a string in* $\mathcal{L}$*.*

Definition 1.13 *The* ***complement*** *of a language* $\mathcal{L}$*, denoted* $\mathcal{L}^C$ *or* $\bar{\mathcal{L}}$ *is the set of strings* $x \in \Sigma^*$ *such that* $x \notin \mathcal{L}$*.*

Definition 1.14 *The* ***concatenation*** *of two languages A and B, denoted $A \circ B$ is the set of strings of the form xy, where $x \in A$ and $y \in B$.*

Definition 1.15 *The* ***Kleene star*** *of a language L, written as L^* (the same as in Σ^*) is the set of all strings w such that w is element of $L \circ L \circ \ldots \circ L$ for some finite number of concatenations of L with itself (including zero, i.e. $\epsilon \in L^*$).*

We're ready to prove our first real theorem about a property of regular languages. It's going to turn out that the class of regular languages is closed under all seven of the operations above. In probably a different order, we will prove this for union, reverse, complement, and concatenation. Note that once we show closure under union and complement, DeMorgan's Laws gives us closure under intersection and symmetric difference for free, and the Kleene Star of a language is an inductive extension of concatenation.

Claim 1.16 *The class of regular languages is closed under complement. That is, the complement of a regular language is regular.*

Proof:

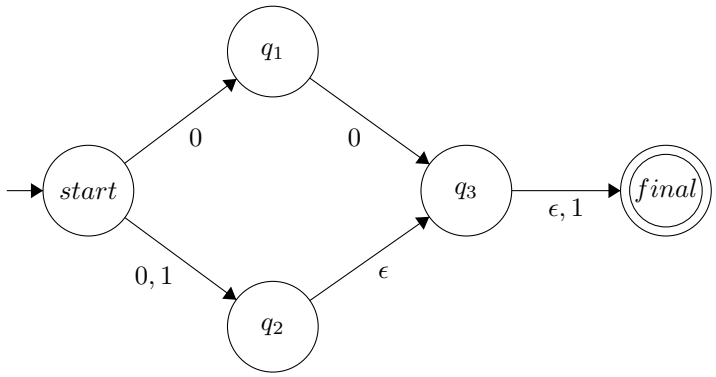
Recall that a language is regular if and only if a DFA accepts it. Given a DFA for a regular language L, we will construct a DFA that accepts exactly those strings on Σ not in L, i.e. L^C.

Given a DFA M that accepts L, note that any string that is not in L finishes its traversal of M in a non-final state and any string that is in L finishes its traversal in a final state. Intuitively, constructing $\bar{M}$ to be a copy of M where a state in $\bar{M}$ is a final state if and only if the corresponding state of M is a non-final state, and a state in $\bar{M}$ is a non-final state if and only if the corresponding state in M is a final state.

To show that $\bar{M}$ accepts the complement of L, we must show that every string in Σ^* is accepted by exactly one of M or $\bar{M}$. Let w be a string in Σ^* suppose (without loss of generality) that w is accepted by M. Then w finishes its traversal in some final state q_i. By construction, passing w to $\bar{M}$ results in the traversal finishing in the corresponding non-final state q_i. Therefore, M and $\bar{M}$ together accept all of Σ^* and no string is accepted by both, therefore $\bar{M}$ accepts L^C, which by definition must be regular. ■

1.3 Nondeterministic Finite Automata

We know there are languages that are not regular. Maybe we can find some by relaxing the definition of the DFA? Suppose we relax the definition of the transition function δ. We allow multiple transitions leaving a state on the same character, not every state must have a transition on every character in Σ, and transitions can happen without consuming an input character, called an ϵ-transition. These are called nondeterministic finite automata (NFAs), because we perform ϵ-transitions and choices from among multiple transitions nondeterministically.



This is an NFA which includes multiple transitions and ϵ-transitions. Sometimes these aren't as intuitive to grasp as DFAs. Let's note that if a string is unable to transition on the current input character, the string is rejected (we can think of an implied sink). For example, there is no transition from q_1 on the character 1, so a string currently in q_1 with the next character of 1 is rejected. Observe that whereas in a DFA the traversal of every string was unique, whereas in an NFA there could be more than one legal traversal. For example, the string 00 could go start $\to_0 q_1 \to_0 q_3 \to_\epsilon final$, where it is accepted, or start $\to_0 q_2 \to_\epsilon q_3 \to_\epsilon final$, where it is rejected because there is nowhere to transition on the remaining 0. So does this machine accept or reject 00? We say that an NFA accepts a string if and only if there exists at least one valid traversal which consumes all input characters and ends in a final state. Therefore, 00 is in the language of this machine.

Let's try to characterize the language accepted by this machine. There is no transition out of the final state, so this machine doesn't accept any string longer than three characters, so it should be pretty simple to list out the strings it does accept:

$00\epsilon = 00$

001

$0\epsilon\epsilon = 0$

$0\epsilon 1 = 01$

$1\epsilon\epsilon = 1$

$1\epsilon 1 = 11$

Note that the empty string is rejected and the strings 10, 111, 101, 011, 110, 100, 000, and 010 are not in the language, and together these lists form an exhaustive enumeration of the strings of length three or less, so we have fully characterized the language. Note that this language is also regular (every finite language is regular), so we didn't just stumble upon an example of an NFA that is more powerful than DFAs. Let's keep exploring.

Definition 1.17 *Formally, a NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$. $Q, \Sigma q_0, F$ are defined as in the formalization of a DFA. The difference in these definitions is in the transition function δ.*

The transition function δ for an NFA is a little tricker than that of a DFA. Whereas in the DFA case, δ was a function which mapped a state and a character to another state, in the NFA case, δ maps a state and a character (now including the empty character ϵ) to a set of states. In the example machine above, δ maps the pair $(start, 0)$ to the set $\{q_1, q_2\}$.

Let's observe that DFAs (almost) satisfy the definition of NFAs, we just need to tweak the elements of the codomain of δ to be singleton sets of states instead of states themselves. Thus, every DFA is also an NFA and the class of languages accepted by NFAs is a superset of the regular languages, but is it a proper superset?

The answer is no. The class of languages accepted by NFAs is exactly the regular languages. We will show this by demonstrating that any NFA can be transformed into a DFA accepting exactly the same language.

Claim 1.18 *Any NFA can be transformed into a proper DFA that accepts the same language.*

Proof: *Let M be an NFA. Suppose, without loss of generality, that M contains both multiple transitions and ϵ-transitions. The key to this construction is that the transition function for a DFA maps characters and states to individual states whereas that of an NFA maps to sets of states.*

Let Q be the set of states in M and δ the transition function. We will define a new set of states Q' for the DFA. States of the DFA (elements of Q') will correspond to subsets of Q and a new transition function δ' for the DFA which will properly states in Q' and characters in Σ to states in Q'.

First, we can construct δ as a table (remember, all the factor sets are finite, so δ still is as well). Make a row for each state of M and a column for each character in Σ, excluding ϵ. In each entry, list the states of M that the row state can transition to on the column character in M, including any possible ϵ-transitions. For example, in the NFA pictured before, the entry at row q_1, column 0 would be the set $\{q_3, final\}$. If no transition is possible, the entry is $\emptyset$.

Now, we can fully construct the DFA. Let q'_0, the start state, correspond to the singleton set containing the start state of M. Now, make a state in Q' for each subset corresponding to a row, column entry in the transition table (we can make one for every subset of Q, but those which do not appear in the table are unreachable and therefore we don't need to include them in the DFA).

For each corresponding transition in δ, construct a transition in δ' where a state q'_i transitions to state q'_j on character k if and only if there exists a transition in δ from some state $a \in Q$ to $b \in Q$ on character k where a is an element of the subset corresponding to the state q'_i and b is in the subset corresponding to q'_j. Finally, mark any state q'_t a final state if and only if its corresponding subset of Q contains at least one final state of M. For sake of completeness and explicitness, the state corresponding to $\emptyset$ is a sink.

It is obvious that this new machine we constructed, call it M', is a DFA, but it is not quite clear that $\mathcal{L}(M) = \mathcal{L}(M')$. We will hand-wave a bit to claim that this equality in fact holds.

If w is a string accepted by M, then there exists some valid traversal for w ending in a final state. Given such a traversal (an ordered list of states of M), we can construct the corresponding traversal in the DFA M' by grouping the ϵ-transitions in the travesral of M into subsets with the preceding character transition (each subset is the character transition grouped with all immediately following ϵ-transitions, if they exist), and mapping these subsets to the corresponding states in M'. Since the traversal in M ends in a final state, the traversal in M' ends in a state corresponding to a subset containing a final state, and M' accepts w. Similarly, any string not accepted by M ends its traversal in a non-final state of M', or in the sink corresponding to $\emptyset$.

Now let x be a string accepted by M'. Its accepting traversal in Q' is unique, but the traversal in M may not be. The list of valid traversals in M can be constructed by enumerating all of the paths generated by choosing one state from each of the subsets corresponding to the traversed states in Q', where ϵ-transitions are included with their preceding character transitions. Since the traversal in M' ends in a final state, at least one of the traversals in M must end in a final state as well, and therefore M accepts x. A similar argument shows that strings not accepted in M' are not accepted in M either.

Therefore, $\mathcal{L}(M) = \mathcal{L}(M')$.

■

1.4 Proving Closure Properties

While NFAs are a little trickier to grasp than DFAs, they allow for non-deterministic (i.e. multiple and ϵ) transitions, and since DFAs and NFAs accept the same class of language, we can prove things about regular languages by proving things about NFAs.

Claim 1.19 *The class of regular languages is closed under concatenation.*

This is tricky to prove without non-determinism. If we have two regular languages A, B and some string, determining whether the string is in the concatenation $A \circ B$ is tricky. We want to determine whether there is a way to split the string into two pieces so that the first part is in A and the second part is in B, but we have no guidance on how to choose where to split the string. It is possible that there are several possible ways to successfully split it, ways to split it such that the first part is in A and the second part is not in B, or the other way around. The string is in $A \circ B$ if there exists at least one valid split, but how do we choose which one?

Proof:

Let A and B be regular languages and denote their concatenation by $A \circ B$. We will construct an NFA that accepts $A \circ B$.

Let M_A be an NFA accepting A and M_B be an NFA accepting B. Perform the following constructive modification:

1. *For each final state in M_A, add an ϵ-transition to the start state of M_B.*
2. *Set the new final states to be only the final states of M_B.*
3. *Set the new start state to be only the start state of M_A.*

This new machine accepts exactly the strings in $A \circ B$. To see this, note that for the first part of the string to be in A, there must be at least one successful traversal that passes through one of the states that was originally a final h of M_A. For the second part to be in B, the successful traversal must then make an ϵ-transition to what was originally the start state of M_B and then end in a final state.

Since we now have an NFA to recognize $A \circ B$, it is regular, therefore the class of regular languages is closed under concatenation.

■

Claim 1.20 *The class of regular languages is closed under the Kleene Star operation.*

The proof of this is similar to the previous one. We will take an NFA that accepts a regular language L and convert it to one that accepts L^*, which is the set of strings that are in some finite number of concatenations of L with itself, including zero.

Proof:

Let L be a regular language and M_L be an NFA (or DFA) that accepts it. Perform the following constructive modification:

1. *Make ϵ-transitions from the final state of M_L to the start state*
2. *Make a new start state which is also a final state and add an ϵ-transition to the original start state.*

Note that this new machine accepts the empty string (zero concatenations of L with itself), as well as non-deterministically allowing an arbitrary (finite) number of splits in the string. If there is a valid way to split the string such that each part is in L, then it is in L^* and this machine accepts it.

■

Claim 1.21 *The class of regular languages is closed under union.*

We again perform a constructive modification to two NFAs accepting regular languages A and B to build one accepting $A \cup B$. We want to somehow construct an NFA that simultaneously runs a string through M_A and M_B and accepts if and only if at least one of these machines accept.

Proof:

Let A and B be regular languages with NFAs M_A, M_B. Add a new start state with ϵ-transitions to the start states of M_A and M_B, representing a non-deterministic choice of which language to check. Since this new machine will accept if and only if at least one of M_A and M_B accept, it recognizes the union of the languages.

■