Lecture 4

More on Regular Sets

Here is another example of a regular set that is a little harder than the example given last time. Consider the set

$$\{x \in \{0,1\}^* \mid x \text{ represents a multiple of three in binary}\} \tag{4.1}$$

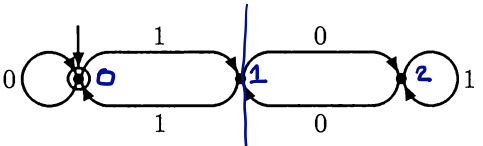
(leading zeros permitted, ϵ represents the number 0). For example, the following binary strings represent multiples of three and should be accepted:

Binary	*Decimal equivalent*
0	0
0011	3
110	6
1001	9
1100	12
1111	15
10010	18
⋮	⋮

Strings not representing multiples of three should be rejected. Here is an automaton accepting the set (4.1):

		0	1
→	**0***F*	**0**	**1**
	1	**2**	**0**
	2	**1**	**2**

The states **0**, **1**, **2** are written in boldface to distinguish them from the input symbols 0, 1.



In the diagram, the states are **0**, **1**, **2** from left to right. We prove that this automaton accepts exactly the set (4.1) by induction on the length of the input string. First we associate a meaning to each state:

if the number represented by the string scanned so far is[1]	*then the machine will be in state*
0 mod 3	**0**
1 mod 3	**1**
2 mod 3	**2**

Let $\#x$ denote the number represented by string x in binary. For example,

$$\#\epsilon = 0,$$
$$\#0 = 0,$$
$$\#11 = 3,$$
$$\#100 = 4,$$

and so on. Formally, we want to show that for any string x in $\{0,1\}^*$,

$$\hat{\delta}(\mathbf{0},x) = \mathbf{0} \text{ iff } \#x \equiv 0 \bmod 3, \tag{4.2}$$
$$\hat{\delta}(\mathbf{0},x) = \mathbf{1} \text{ iff } \#x \equiv 1 \bmod 3,$$
$$\hat{\delta}(\mathbf{0},x) = \mathbf{2} \text{ iff } \#x \equiv 2 \bmod 3,$$

or in short,

$$\hat{\delta}(\mathbf{0},x) = \#x \bmod 3. \tag{4.3}$$

This will be our induction hypothesis. The final result we want, namely (4.2), is a weaker consequence of (4.3), but we need the more general statement (4.3) for the induction hypothesis.

We have by elementary number theory that

$$\#(x0) = 2(\#x) + 0,$$

[1] Here $a \bmod n$ denotes the remainder when dividing a by n using ordinary integer division. We also write $a \equiv b \bmod n$ (read: a is congruent to b modulo n) to mean that a and b have the same remainder when divided by n; in other words, that n divides $b - a$. Note that $a \equiv b \bmod n$ should be parsed $(a \equiv b) \bmod n$, and that in general $a \equiv b \bmod n$ and $a = b \bmod n$ mean different things. For example, $7 \equiv 2 \bmod 5$ but not $7 = 2 \bmod 5$.

$$\#(x1) = 2(\#x) + 1,$$

or in short,

$$\#(xc) = 2(\#x) + c \tag{4.4}$$

for $c \in \{0,1\}$. From the machine above, we see that for any state $q \in \{0,1,2\}$ and input symbol $c \in \{0,1\}$,

$$\delta(q,c) = (2q + c) \bmod 3. \tag{4.5}$$

This can be verified by checking all six cases corresponding to possible choices of q and c. (In fact, (4.5) would have been a great way to *define* the transition function formally—then we wouldn't have had to prove it!) Now we use the inductive definition of $\hat{\delta}$ to show (4.3) by induction on $|x|$.

Basis

For $x = \epsilon$,

$$\begin{aligned} \hat{\delta}(\mathbf{0},\epsilon) &= \mathbf{0} && \text{by definition of } \hat{\delta} \\ &= \#\epsilon && \text{since } \#\epsilon = 0 \\ &= \#\epsilon \bmod 3. \end{aligned}$$

Induction step

Assuming that (4.3) is true for $x \in \{0,1\}^*$, we show that it is true for xc, where $c \in \{0,1\}$.

$$\begin{aligned} \hat{\delta}(\mathbf{0},xc) &= \delta(\hat{\delta}(\mathbf{0},x),c) && \text{definition of } \hat{\delta} \\ &= \delta(\#x \bmod 3, c) && \text{induction hypothesis} \\ &= (2(\#x \bmod 3) + c) \bmod 3 && \text{by (4.5)} \\ &= (2(\#x) + c) \bmod 3 && \text{elementary number theory} \\ &= \#xc \bmod 3 && \text{by (4.4).} \end{aligned}$$

Note that each step has its reason. We used the definition of δ, which is specific to this automaton; the definition of $\hat{\delta}$ from δ, which is the same for all automata; and elementary properties of numbers and strings.

Some Closure Properties of Regular Sets

For $A, B \subseteq \Sigma^*$, recall the following definitions:

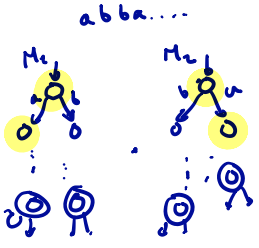
$$\begin{aligned} A \cup B &= \{x \mid x \in A \text{ or } x \in B\} && \text{union} \\ A \cap B &= \{x \mid x \in A \text{ and } x \in B\} && \text{intersection} \\ \sim A &= \{x \in \Sigma^* \mid x \notin A\} && \text{complement} \end{aligned}$$

$$AB = \{xy \mid x \in A \text{ and } y \in B\} \qquad \text{concatenation}$$
$$A^* = \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in A,\ 1 \leq i \leq n\}$$
$$= A^0 \cup A^1 \cup A^2 \cup A^3 \cup \cdots \qquad \text{asterate.}$$

Do not confuse set concatenation with string concatenation. Sometimes $\sim A$ is written $\Sigma^* - A$.

We show below that if A and B are regular, then so are $A \cup B$, $A \cap B$, and $\sim A$. We'll show later that AB and A^* are also regular.

The Product Construction



Assume that A and B are regular. Then there are automata

$$M_1 = (Q_1,\ \Sigma,\ \delta_1,\ s_1,\ F_1),$$
$$M_2 = (Q_2,\ \Sigma,\ \delta_2,\ s_2,\ F_2)$$

with $L(M_1) = A$ and $L(M_2) = B$. To show that $A \cap B$ is regular, we will build an automaton M_3 such that $L(M_3) = A \cap B$.

Intuitively, M_3 will have the states of M_1 and M_2 encoded somehow in its states. On input $x \in \Sigma^*$, it will simulate M_1 and M_2 simultaneously on x, accepting iff both M_1 and M_2 would accept. Think about putting a pebble down on the start state of M_1 and another on the start state of M_2. As the input symbols come in, move both pebbles according to the rules of each machine. Accept if both pebbles occupy accept states in their respective machines when the end of the input string is reached.

Formally, let

$$M_3 = (Q_3,\ \Sigma,\ \delta_3,\ s_3,\ F_3),$$

where

$$Q_3 = Q_1 \times Q_2 = \{(p,q) \mid p \in Q_1 \text{ and } q \in Q_2\},$$
$$F_3 = F_1 \times F_2 = \{(p,q) \mid p \in F_1 \text{ and } q \in F_2\},$$
$$s_3 = (s_1, s_2),$$

and let

$$\delta_3 : Q_3 \times \Sigma \to Q_3$$

be the transition function defined by

$$\delta_3((p,q),a) = (\delta_1(p,a), \delta_2(q,a)).$$

The automaton M_3 is called the *product* of M_1 and M_2. A state (p,q) of M_3 encodes a configuration of pebbles on M_1 and M_2.

Recall the inductive definition (3.1) and (3.2) of the extended transition function $\widehat{\delta}$ from Lecture 2. Applied to δ_3, this gives

$$\widehat{\delta}_3((p,q),\epsilon) = (p,q),$$
$$\widehat{\delta}_3((p,q),xa) = \delta_3(\widehat{\delta}_3((p,q),x),a).$$

Lemma 4.1 *For all $x \in \Sigma^*$,*

$$\widehat{\delta}_3((p,q),x) = (\widehat{\delta}_1(p,x),\widehat{\delta}_2(q,x)).$$

Proof. By induction on $|x|$.

Basis

For $x = \epsilon$,

$$\widehat{\delta}_3((p,q),\epsilon) = (p,q) = (\widehat{\delta}_1(p,\epsilon),\widehat{\delta}_2(q,\epsilon)).$$

Induction step

Assuming the lemma holds for $x \in \Sigma^*$, we show that it holds for xa, where $a \in \Sigma$.

$$\begin{aligned}
&\widehat{\delta}_3((p,q),xa) \\
&= \delta_3(\widehat{\delta}_3((p,q),x),a) && \text{definition of } \widehat{\delta}_3 \\
&= \delta_3((\widehat{\delta}_1(p,x),\widehat{\delta}_2(q,x)),a) && \text{induction hypothesis} \\
&= (\delta_1(\widehat{\delta}_1(p,x),a),\delta_2(\widehat{\delta}_2(q,x),a)) && \text{definition of } \delta_3 \\
&= (\widehat{\delta}_1(p,xa),\widehat{\delta}_2(q,xa)) && \text{definition of } \widehat{\delta}_1 \text{ and } \widehat{\delta}_2.
\end{aligned}$$ □

Theorem 4.2 $L(M_3) = L(M_1) \cap L(M_2)$.

Proof. For all $x \in \Sigma^*$,

$$\begin{aligned}
&x \in L(M_3) \\
&\iff \widehat{\delta}_3(s_3,x) \in F_3 && \text{definition of acceptance} \\
&\iff \widehat{\delta}_3((s_1,s_2),x) \in F_1 \times F_2 && \text{definition of } s_3 \text{ and } F_3 \\
&\iff (\widehat{\delta}_1(s_1,x),\widehat{\delta}_2(s_2,x)) \in F_1 \times F_2 && \text{Lemma 4.1} \\
&\iff \widehat{\delta}_1(s_1,x) \in F_1 \text{ and } \widehat{\delta}_2(s_2,x) \in F_2 && \text{definition of set product} \\
&\iff x \in L(M_1) \text{ and } x \in L(M_2) && \text{definition of acceptance} \\
&\iff x \in L(M_1) \cap L(M_2) && \text{definition of intersection.}
\end{aligned}$$ □

To show that regular sets are closed under complement, take a deterministic automaton accepting A and interchange the set of accept and nonaccept states. The resulting automaton accepts exactly when the original automaton would reject, so the set accepted is $\sim A$.

Once we know regular sets are closed under $\cap$ and $\sim$, it follows that they are closed under $\cup$ by one of the De Morgan laws:

$$A \cup B = \sim(\sim A \cap \sim B).$$

If you use the constructions for $\cap$ and $\sim$ given above, this gives an automaton for $A \cup B$ that looks exactly like the product automaton for $A \cap B$, except that the accept states are

$$F_3 = \{(p, q) \mid p \in F_1 \text{ or } q \in F_2\} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$

instead of $F_1 \times F_2$.

Historical Notes

Finite-state transition systems were introduced by McCulloch and Pitts in 1943 [84]. Deterministic finite automata in the form presented here were studied by Kleene [70]. Our notation is borrowed from Hopcroft and Ullman [60].

Lecture 5

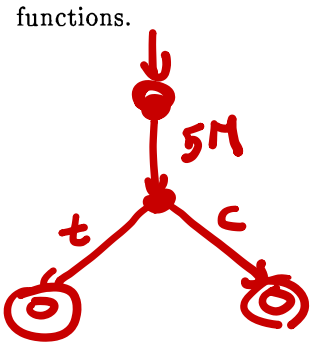
Nondeterministic Finite Automata

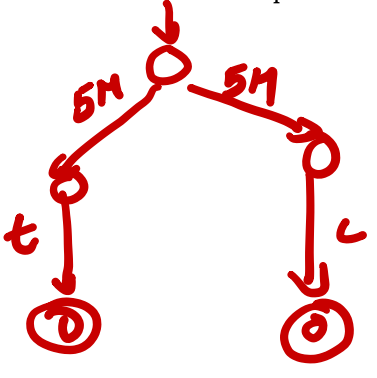
Nondeterminism

Nondeterminism is an important abstraction in computer science. It refers to situations in which the next state of a computation is not uniquely determined by the current state. Nondeterminism arises in real life when there is incomplete information about the state or when there are external forces at work that can affect the course of a computation. For example, the behavior of a process in a distributed system might depend on messages from other processes that arrive at unpredictable times with unpredictable contents.

Nondeterminism is also important in the design of efficient algorithms. There are many instances of important combinatorial problems with efficient nondeterministic solutions but no known efficient deterministic solution. The famous $P = NP$ problem—whether all problems solvable in nondeterministic polynomial time can be solved in deterministic polynomial time—is a major open problem in computer science and arguably one of the most important open problems in all of mathematics.

In nondeterministic situations, we may not know how a computation will evolve, but we may have some idea of the range of possibilities. This is modeled formally by allowing automata to have multiple-valued transition functions.





In this lecture and the next, we will show how nondeterminism is incorporated naturally in the context of finite automata. One might think that adding nondeterminism might increase expressive power, but in fact for finite automata it does not: in terms of the sets accepted, nondeterministic finite automata are no more powerful than deterministic ones. In other words, for every nondeterministic finite automaton, there is a deterministic one accepting the same set. However, nondeterministic machines may be exponentially more succinct.

Nondeterministic Finite Automata

A *nondeterministic finite automaton* (NFA) is one for which the next state is not necessarily uniquely determined by the current state and input symbol. In a deterministic automaton, there is exactly one start state and exactly one transition out of each state for each symbol in Σ. In a nondeterministic automaton, there may be one, more than one, or zero. The set of *possible* next states that the automaton may move to from a particular state q in response to a particular input symbol a is part of the specification of the automaton, but there is no mechanism for deciding which one will actually be taken. Formally, we won't be able to represent this with a function $\delta : Q \times \Sigma \to Q$ anymore; we will have to use something more general. Also, a nondeterministic automaton may have many start states and may start in any one of them.

Informally, a nondeterministic automaton is said to *accept* its input x if it is possible to start in some start state and scan x, moving according to the transition rules and making choices along the way whenever the next state is not uniquely determined, such that when the end of x is reached, the machine is in an accept state. Because the start state is not determined and because of the choices along the way, there may be several possible paths through the automaton in response to the input x; some may lead to accept states while others may lead to reject states. The automaton is said to *accept* x if *at least one* computation path on input x starting from *at least one* start state leads to an accept state. The automaton is said to *reject* x if *no* computation path on input x from *any* start state leads to an accept state. Another way of saying this is that x is accepted iff there exists a path with label x from some start state to some accept state. Again, there is no mechanism for determining which state to start in or which of the possible next moves to take in response to an input symbol.

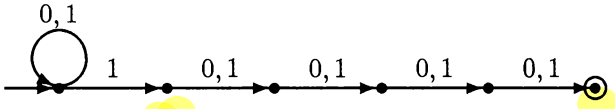
It is helpful to think about this process in terms of *guessing* and *verifying*. On a given input, imagine the automaton *guessing* a successful computation or proof that the input is a "yes" instance of the decision problem, then *verifying* that its guess was indeed correct.

For example, consider the set

$$A = \{x \in \{0,1\}^* \mid \text{the fifth symbol from the right is } 1\}.$$

Thus $11010010 \in A$ but $11000010 \notin A$.

Here is a six-state nondeterministic automaton accepting A:



There is only one start state, namely the leftmost, and only one accept state, namely the rightmost. The automaton is not deterministic, because there are two transitions from the leftmost state labeled 1 (one back to itself and one to the second state) and no transitions from the rightmost state. This automaton accepts the set A, because for any string x whose fifth symbol from the right is 1, *there exists* a sequence of legal transitions leading from the start state to the accept state (it moves from the first state to the second when it scans the fifth symbol from the right); and for any string x whose fifth symbol from the right is 0, there is *no possible* sequence of legal transitions leading to the accept state, no matter what choices it makes (recall that to accept, the machine must be in an accept state when the end of the input string is reached).

Intuitively, we can think of the machine in the leftmost state as *guessing*, every time it sees a 1, whether that 1 is the fifth letter from the right. It might be and it might not be—the machine doesn't know, and there is no way for it to tell at that point. If it guesses that it is not, then it goes around the loop again. If it guesses that it is, then it commits to that guess by moving to the second state, an irrevocable decision. Now it must *verify* that its guess was correct; this is the purpose of the tail of the automaton leading to the accept state. If the 1 that it guessed was fifth from the right really is fifth from the right, then the machine will be in its accept state exactly when it comes to the end of the input string, therefore it will accept the string. If not, then maybe the symbol fifth from the right is a 0, and *no* guess would have worked; or maybe the symbol fifth from the right was a 1, but the machine just guessed the wrong 1.

Note, however, that for any string $x \in A$ (that is, for any string with a 1 fifth from the right), *there is* a lucky guess that leads to acceptance; whereas for any string $x \notin A$ (that is, for any string with a 0 fifth from the right), *no* guess can possibly lead to acceptance, no matter how lucky the automaton is.

In general, to show that a nondeterministic machine accepts a set B, we must argue that for any string $x \in B$, there is a lucky sequence of guesses that leads from a start state to an accept state when the end of x is reached;

but for any string $x \notin B$, *no* sequence of guesses leads to an accept state when the end of x is reached, no matter how lucky the automaton is.

Keep in mind that this process of *guessing and verifying* is just an intuitive aid. The formal definition of nondeterministic acceptance will be given in Lecture 6.

There does exist a deterministic automaton accepting the set A, but any such automaton must have at least $2^5 = 32$ states, since a deterministic machine essentially has to remember the last five symbols seen.

The Subset Construction

We will prove a rather remarkable fact: in terms of the sets accepted, nondeterministic finite automata are no more powerful than deterministic ones. In other words, for every nondeterministic finite automaton, there is a deterministic one accepting the same set. The deterministic automaton, however, may require more states.

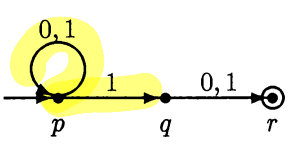
This theorem can be proved using the *subset construction.* Here is the intuitive idea; we will give a formal treatment in Lecture 6. Given a nondeterministic machine N, think of putting pebbles on the states to keep track of all the states N could possibly be in after scanning a prefix of the input. We start with pebbles on all the start states of the nondeterministic machine. Say after scanning some prefix y of the input string, we have pebbles on some set P of states, and say P is the set of all states N could possibly be in after scanning y, depending on the nondeterministic choices that N could have made so far. If input symbol b comes in, pick the pebbles up off the states of P and put a pebble down on each state reachable from a state in P under input symbol b. Let P' be the new set of states covered by pebbles. Then P' is the set of states that N could possibly be in after scanning yb.

Although for a state q of N, there may be many possible next states after scanning b, note that the set P' is uniquely determined by b and the set P. We will thus build a deterministic automaton M *whose states are these sets.* That is, a state of M will be a *set* of states of N. The start state of M will be the *set* of start states of N, indicating that we start with one pebble on each of the start states of N. A final state of M will be any set P containing a final state of N, since we want to accept x if it is possible for N to have made choices while scanning x that lead to an accept state of N.

It takes a stretch of the imagination to regard a set of states of N as a single state of M. Let's illustrate the construction with a shortened version of the example above.

Example 5.1 Consider the set

$$A = \{x \in \{0,1\}^* \mid \text{the second symbol from the right is } 1\}.$$



Label the states p, q, r from left to right, as illustrated. The states of M will be *subsets* of the set of states of N. In this example there are eight such subsets:

$$\varnothing,\ \{p\},\ \{q\},\ \{r\},\ \{p,q\},\ \{p,r\},\ \{q,r\},\ \{p,q,r\}.$$

Here is the deterministic automaton M:

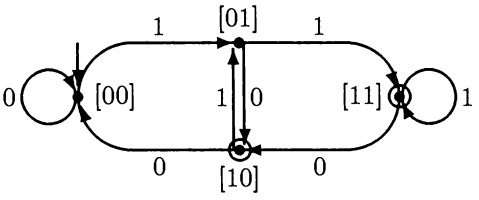
		0	1
	$\varnothing$	$\varnothing$	$\varnothing$
$\to$	$\{p\}$	$\{p\}$	$\{p,q\}$
	$\{q\}$	$\{r\}$	$\{r\}$
	$\{r\}F$	$\varnothing$	$\varnothing$
	$\{p,q\}$	$\{p,r\}$	$\{p,q,r\}$
	$\{p,r\}F$	$\{p\}$	$\{p,q\}$
	$\{q,r\}F$	$\{r\}$	$\{r\}$
	$\{p,q,r\}F$	$\{p,r\}$	$\{p,q,r\}$

For example, if we have pebbles on p and q (the fifth row of the table), and if we see input symbol 0 (first column), then in the next step there will be pebbles on p and r. This is because in the automaton N, p is reachable from p under input 0 and r is reachable from q under input 0, and these are the only states reachable from p and q under input 0. The accept states of M (marked F in the table) are those sets containing an accept state of N. The start state of M is $\{p\}$, the set of all start states of N.

Following 0 and 1 transitions from the start state $\{p\}$ of M, one can see that states $\{q,r\}$, $\{q\}$, $\{r\}$, $\varnothing$ of M can never be reached. These states of M are *inaccessible*, and we might as well throw them out. This leaves

	0	1
$\to$ $\{p\}$	$\{p\}$	$\{p,q\}$
$\{p,q\}$	$\{p,r\}$	$\{p,q,r\}$
$\{p,r\}F$	$\{p\}$	$\{p,q\}$
$\{p,q,r\}F$	$\{p,r\}$	$\{p,q,r\}$

This four-state automaton is exactly the one you would have come up with if you had built a deterministic automaton directly to remember the last two bits seen and accept if the next-to-last bit is a 1:



Here the state labels $[bc]$ indicate the last two bits seen (for our purposes the null string is as good as having just seen two 0's). Note that these two automata are isomorphic (i.e., they are the same automaton up to the renaming of states):

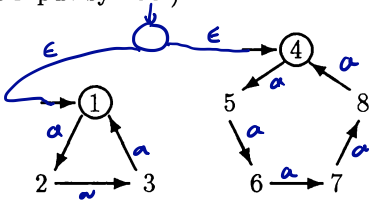
$$\begin{aligned} \{p\} &\approx [00], \\ \{p,q\} &\approx [01], \\ \{p,r\} &\approx [10], \\ \{p,q,r\} &\approx [11]. \end{aligned}$$

□

Example 5.2 Consider the set

$$\{x \in \{a\}^* \mid |x| \text{ is divisible by 3 or 5}\}. \tag{5.1}$$

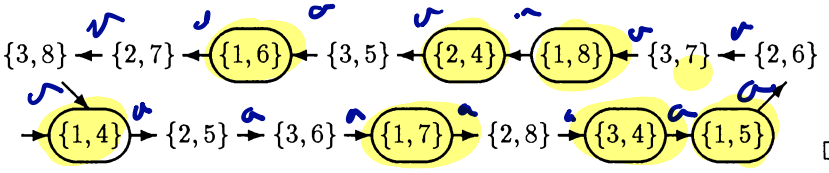
Here is an eight-state nondeterministic automaton N with two start states accepting this set (labels a on transitions are omitted since there is only one input symbol).



The only nondeterminism is in the choice of start state. The machine guesses at the outset whether to check for divisibility by 3 or 5. After that, the computation is deterministic.

Let Q be the states of N. We will build a deterministic machine M whose states are subsets of Q. There are $2^8 = 256$ of these in all, but most will be inaccessible (not reachable from the start state of M under any input). Think about moving pebbles—for this particular automaton, if you start with pebbles on the start states and move pebbles to mark all states the machine could possibly be in, you always have exactly two pebbles on N. This says that only subsets of Q with two elements will be accessible as states of M.

The subset construction gives the following deterministic automaton M with 15 accessible states:



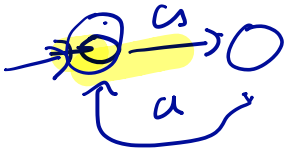
□

In the next lecture we will give a formal definition of nondeterministic finite automata and a general account of the subset construction.

$\Sigma = \{a, b\}$

$A = \{ x \in \Sigma^* \mid \#a(x) \text{ is divisible by } 3 \}$

$A = \{ x \in \Sigma^* \mid \#a(x) \text{ is divisible by } 3 \}$



a a a b a

$B = \{ a^m \mid m = 2^n \text{ for some } n \}$

$a^1 = a$

$a^2 = aa$

$a^n = \underbrace{aaa\ldots a}_{n \text{ times}}$

Lecture 6

The Subset Construction

Formal Definition of Nondeterministic Finite Automata

A *nondeterministic finite automaton (NFA)* is a five-tuple

$$N = (Q, \Sigma, \Delta, S, F),$$

where everything is the same as in a deterministic automaton, except for the following two differences.

- S is a *set* of states, that is, $S \subseteq Q$, instead of a single state. The elements of S are called *start states*.
- Δ is a function

  $$\Delta : Q \times \Sigma \to 2^Q,$$

  where 2^Q denotes the *power set* of Q or the set of all subsets of Q:

  $$2^Q \stackrel{\text{def}}{=} \{A \mid A \subseteq Q\}.$$

Intuitively, $\Delta(p, a)$ gives the set of all states that N is allowed to move to from p in one step under input symbol a. We often write

$$p \xrightarrow{a} q$$

if $q \in \Delta(p, a)$. The set $\Delta(p, a)$ can be the empty set $\varnothing$. The function Δ is called the *transition function*.

Now we define acceptance for NFAs. The function Δ extends in a natural way by induction to a function

$$\widehat{\Delta} : 2^Q \times \Sigma^* \to 2^Q$$

according to the rules

$$\widehat{\Delta}(A, \epsilon) \stackrel{\text{def}}{=} A, \tag{6.1}$$

$$\widehat{\Delta}(A, xa) \stackrel{\text{def}}{=} \bigcup_{q \in \widehat{\Delta}(A,x)} \Delta(q, a). \tag{6.2}$$

Intuitively, for $A \subseteq Q$ and $x \in \Sigma^*$, $\widehat{\Delta}(A, x)$ is the set of all states reachable under input string x from *some* state in A. Note that Δ takes a single state as its first argument and a single symbol as its second argument, whereas $\widehat{\Delta}$ takes a *set* of states as its first argument and a *string* of symbols as its second argument.

Equation (6.1) says that the set of all states reachable from a state in A under the null input is just A. In (6.2), the notation on the right-hand side means the union of all the sets $\Delta(q, a)$ for $q \in \widehat{\Delta}(A, x)$; in other words, $r \in \widehat{\Delta}(A, xa)$ if there exists $q \in \widehat{\Delta}(A, x)$ such that $r \in \Delta(q, a)$.

Thus $q \in \widehat{\Delta}(A, x)$ if N can move from some state $p \in A$ to state q under input x. This is the nondeterministic analog of the construction of $\widehat{\delta}$ for deterministic automata we have already seen.

Note that for $a \in \Sigma$,

$$\begin{aligned} \widehat{\Delta}(A, a) &= \bigcup_{p \in \widehat{\Delta}(A,\epsilon)} \Delta(p, a) \\ &= \bigcup_{p \in A} \Delta(p, a). \end{aligned}$$

The automaton N is said to *accept* $x \in \Sigma^*$ if

$$\widehat{\Delta}(S, x) \cap F \neq \varnothing.$$

In other words, N accepts x if there exists an accept state q (i.e., $q \in F$) such that q is reachable from a start state under input string x (i.e., $q \in \widehat{\Delta}(S, x)$).

We define $L(N)$ to be the set of all strings accepted by N:

$$L(N) = \{x \in \Sigma^* \mid N \text{ accepts } x\}.$$

Under this definition, every DFA

$$(Q, \Sigma, \delta, s, F)$$

is equivalent to an NFA

$$(Q, \Sigma, \Delta, \{s\}, F),$$

where $\Delta(p,a) \stackrel{\text{def}}{=} \{\delta(p,a)\}$. Below we will show that the converse holds as well: every NFA is equivalent to some DFA.

Here are some basic lemmas that we will find useful when dealing with NFAs. The first corresponds to Exercise 3 of Homework 1 for deterministic automata.

Lemma 6.1 *For any $x, y \in \Sigma^*$ and $A \subseteq Q$,*

$$\hat{\Delta}(A, xy) = \hat{\Delta}(\hat{\Delta}(A,x), y).$$

Proof. The proof is by induction on $|y|$.

Basis

For $y = \epsilon$,

$$\begin{aligned}\hat{\Delta}(A, x\epsilon) &= \hat{\Delta}(A,x) \\ &= \hat{\Delta}(\hat{\Delta}(A,x), \epsilon) \quad \text{by (6.1).}\end{aligned}$$

Induction step

For any $y \in \Sigma^*$ and $a \in \Sigma$,

$$\begin{aligned}\hat{\Delta}(A, xya) &= \bigcup_{q \in \hat{\Delta}(A,xy)} \Delta(q,a) && \text{by (6.2)} \\ &= \bigcup_{q \in \hat{\Delta}(\hat{\Delta}(A,x),y)} \Delta(q,a) && \text{induction hypothesis} \\ &= \hat{\Delta}(\hat{\Delta}(A,x), ya) && \text{by (6.2).}\end{aligned}$$

□

Lemma 6.2 *The function $\hat{\Delta}$ commutes with set union: for any indexed family A_i of subsets of Q and $x \in \Sigma^*$,*

$$\hat{\Delta}(\bigcup_i A_i, x) = \bigcup_i \hat{\Delta}(A_i, x).$$

Proof. By induction on $|x|$.

$\hat{\Delta}(A \cup B, x) = \hat{\Delta}(A,x) \cup \hat{\Delta}(B,x)$

Basis

By (6.1),

$$\hat{\Delta}(\bigcup_i A_i, \epsilon) = \bigcup_i A_i = \bigcup_i \hat{\Delta}(A_i, \epsilon).$$

Induction step

$$\begin{aligned}
\hat{\Delta}(\bigcup_i A_i, xa) &= \bigcup_{p \in \hat{\Delta}(\bigcup_i A_i, x)} \Delta(p, a) && \text{by (6.2)} \\
&= \bigcup_{p \in \bigcup_i \hat{\Delta}(A_i, x)} \Delta(p, a) && \text{induction hypothesis} \\
&= \bigcup_i \bigcup_{p \in \hat{\Delta}(A_i, x)} \Delta(p, a) && \text{basic set theory} \\
&= \bigcup_i \hat{\Delta}(A_i, xa) && \text{by (6.2).}
\end{aligned}$$

□

In particular, expressing a set as the union of its singleton subsets,

$$\hat{\Delta}(A, x) = \bigcup_{p \in A} \hat{\Delta}(\{p\}, x). \tag{6.3}$$

The Subset Construction: General Account

The subset construction works in general. Let

$$N = (Q_N, \Sigma, \Delta_N, S_N, F_N)$$

be an arbitrary NFA. We will use the subset construction to produce an equivalent DFA. Let M be the DFA

$$M = (Q_M, \Sigma, \delta_M, s_M, F_M),$$

where

$$\begin{aligned}
Q_M &\stackrel{\text{def}}{=} 2^{Q_N}, \\
\delta_M(A, a) &\stackrel{\text{def}}{=} \hat{\Delta}_N(A, a), \\
s_M &\stackrel{\text{def}}{=} S_N, \\
F_M &\stackrel{\text{def}}{=} \{A \subseteq Q_N \mid A \cap F_N \neq \varnothing\}.
\end{aligned}$$

Note that δ_M is a function from states of M and input symbols to states of M, as it should be, because states of M are *sets* of states of N.

Lemma 6.3 *For any $A \subseteq Q_N$ and $x \in \Sigma^*$,*

$$\hat{\delta}_M(A,x) = \hat{\Delta}_N(A,x).$$

Proof. Induction on $|x|$.

Basis

For $x = \epsilon$, we want to show

$$\hat{\delta}_M(A,\epsilon) = \hat{\Delta}_N(A,\epsilon).$$

But both of these are A, by definition of $\hat{\delta}_M$ and $\hat{\Delta}_N$.

Induction step

Assume that

$$\hat{\delta}_M(A,x) = \hat{\Delta}_N(A,x).$$

We want to show the same is true for xa, $a \in \Sigma$.

$$\begin{aligned}
\hat{\delta}_M(A,xa) &= \delta_M(\hat{\delta}_M(A,x),a) && \text{definition of } \hat{\delta}_M \\
&= \delta_M(\hat{\Delta}_N(A,x),a) && \text{induction hypothesis} \\
&= \hat{\Delta}_N(\hat{\Delta}_N(A,x),a) && \text{definition of } \delta_M \\
&= \hat{\Delta}_N(A,xa) && \text{Lemma 6.1.}
\end{aligned}$$

□

Theorem 6.4 *The automata M and N accept the same set.*

Proof. For any $x \in \Sigma^*$,

$$\begin{aligned}
& x \in L(M) \\
\iff & \hat{\delta}_M(s_M,x) \in F_M && \text{definition of acceptance for } M \\
\iff & \hat{\Delta}_N(S_N,x) \cap F_N \neq \varnothing && \text{definition of } s_M \text{ and } F_M, \text{ Lemma 6.3} \\
\iff & x \in L(N) && \text{definition of acceptance for } N.
\end{aligned}$$

□

ϵ-Transitions

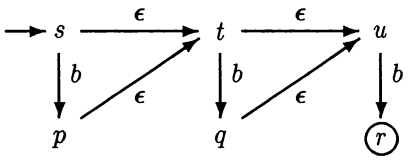
Here is another extension of finite automata that turns out to be quite useful but really adds no more power.

An *ϵ-transition* is a transition with label ϵ, a letter that stands for the null string ϵ:

$$p \xrightarrow{\epsilon} q.$$

The automaton can take such a transition anytime without reading an input symbol.

Example 6.5

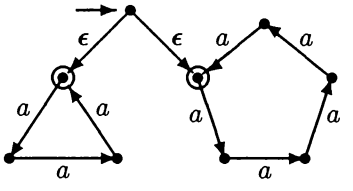


If the machine is in state s and the next input symbol is b, it can nondeterministically decide to do one of three things:

- read the b and move to state p;
- slide to t without reading an input symbol, then read the b and move to state q; or
- slide to t without reading an input symbol, then slide to u without reading an input symbol, then read the b and move to state r.

The set of strings accepted by this automaton is $\{b, bb, bbb\}$. □

Example 6.6 Here is a nondeterministic automaton with ϵ-transitions accepting the set $\{x \in \{a\}^* \mid |x| \text{ is divisible by 3 or 5}\}$:



The automaton chooses at the outset which of the two conditions to check for (divisibility by 3 or 5) and slides to one of the two loops accordingly without reading an input symbol. □

The main benefit of ϵ-transitions is convenience. They do not really add any power: a modified subset construction involving the notion of *ϵ-closure* can be used to show that every NFA with ϵ-transitions can be simulated by a DFA without ϵ-transitions (Miscellaneous Exercise 10); thus all sets accepted by nondeterministic automata with ϵ-transitions are regular. We will also give an alternative treatment in Lecture 10 using homomorphisms.

More Closure Properties

Recall that the concatenation of sets A and B is the set

$$AB = \{xy \mid x \in A \text{ and } y \in B\}.$$

$\Sigma = \{0, 1\}$

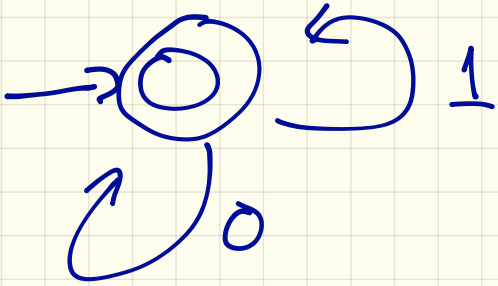
Let $S \subseteq \Sigma^*$.

S is regular

If

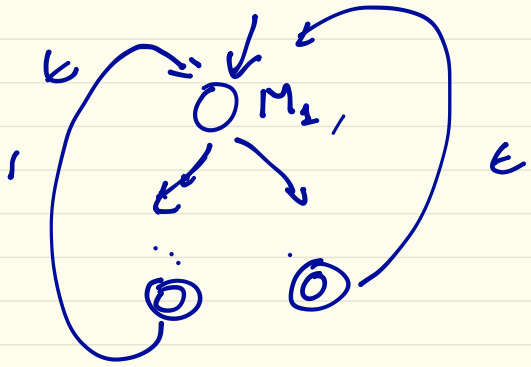
There exist a FSA A such that

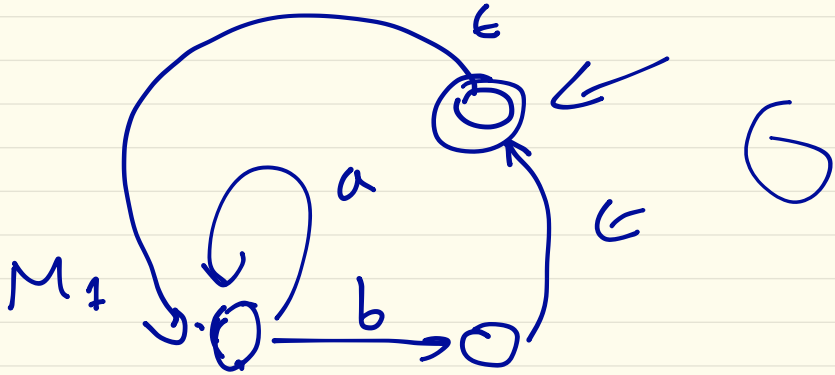
$S = L(A)$



If A is regular

then A^* is regular





$L(M_1) = \{a^n b \mid n \geq 0\}$

$A = a^* b$

$A^* = (a^* b)^*$

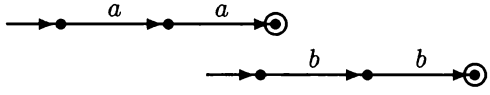
$A^* = \{\epsilon\} \cup \{\text{strings ending in } b\}$

For example,

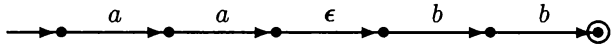
$$\{a, ab\}\{b, ba\} = \{ab, aba, abb, abba\}.$$

If A and B are regular, then so is AB. To see this, let M be an automaton for A and N an automaton for B. Make a new automaton P whose states are the union of the state sets of M and N, and take all the transitions of M and N as transitions of P. Make the start states of M the start states of P and the final states of N the final states of P. Finally, put ϵ-transitions from all the final states of M to all the start states of N. Then $L(P) = AB$.

Example 6.7 Let $A = \{aa\}$, $B = \{bb\}$. Here are automata for A and B:



Here is the automaton you get by the construction above for AB:



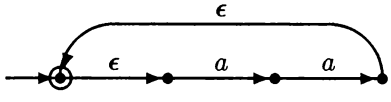
□

If A is regular, then so is its asterate:

$$\begin{aligned} A^* &= \{\epsilon\} \cup A \cup A^2 \cup A^3 \cup \cdots \\ &= \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in A, 1 \leq i \leq n\}. \end{aligned}$$

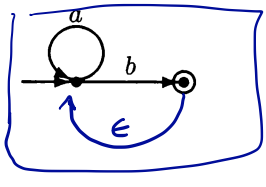
To see this, take an automaton M for A. Build an automaton P for A^* as follows. Start with all the states and transitions of M. Add a new state s. Add ϵ-transitions from s to all the start states of M and from all the final states of M to s. Make s the only start state of P and also the only final state of P (thus the start and final states of M are *not* start and final states of P). Then P accepts exactly the set A^*.

Example 6.8 Let $A = \{aa\}$. Consider the three-state automaton for A in Example 6.7. Here is the automaton you get for A^* by the construction above:



□

In this construction, you must add the new start/final state s. You might think that it suffices to put in ϵ-transitions from the old final states back to the old start states and make the old start states final states, but this doesn't always work. Here's a counterexample:



The set accepted is $\{a^n b \mid n \geq 0\}$. The asterate of this set is

$$\{\epsilon\} \cup \{\text{strings ending with } b\},$$

but if you put in an ϵ-transition from the final state back to the start state and made the start state a final state, then the set accepted would be $\{a, b\}^*$.

Historical Notes

Rabin and Scott [102] introduced nondeterministic finite automata and showed using the subset construction that they were no more powerful than deterministic finite automata.

Closure properties of regular sets were studied by Ginsburg and Rose [46, 48], Ginsburg [43], McNaughton and Yamada [85], and Rabin and Scott [102], among others.

$S = \{ s \in \{0,1\}^* \mid$
$|s|$ is divisible
by 2 $\}$

Lecture 7

Pattern Matching

What happens when one types `rm *` in UNIX? (If you don't know, don't try it to find out!) What if the current directory contains the files

```
a.tex  bc.tex  a.dvi  bc.dvi
```

and one types `rm *.dvi`? What would happen if there were a file named `.dvi`?

What is going on here is *pattern matching*. The * in UNIX is a pattern that matches any string of symbols, including the null string.

Pattern matching is an important application of finite automata. The UNIX commands **grep**, **fgrep**, and **egrep** are basic pattern-matching utilities that use finite automata in their implementation.

Let Σ be a finite alphabet. A *pattern* is a string of symbols of a certain form representing a (possibly infinite) set of strings in Σ^*. The set of patterns is defined formally by induction below. They are either *atomic patterns* or *compound patterns* built up inductively from atomic patterns using certain *operators*. We'll denote patterns by Greek letters α, β, γ, ...

As we define patterns, we will tell which strings $x \in \Sigma^*$ *match* them. The set of strings in Σ^* matching a given pattern α will be denoted $L(\alpha)$. Thus

$$L(\alpha) = \{x \in \Sigma^* \mid x \text{ matches } \alpha\}.$$

In the following, forget the UNIX definition of *. We will use the symbol * for something else.

The *atomic patterns* are

- a for each $a \in \Sigma$, matched by the symbol a only; in symbols, $L(a) = \{a\}$;
- ϵ, matched only by ϵ, the null string; in symbols, $L(\epsilon) = \{\epsilon\}$;
- $\emptyset$, matched by nothing; in symbols, $L(\emptyset) = \varnothing$, the empty set;
- #, matched by any symbol in Σ; in symbols, $L(\#) = \Sigma$;
- @, matched by any string in Σ^*; in symbols, $L(@) = \Sigma^*$.

Compound patterns are formed inductively using binary operators $+$, $\cap$, and $\cdot$ (usually not written) and unary operators $^+$, *, and $\sim$. If α and β are patterns, then so are $\alpha + \beta$, $\alpha \cap \beta$, α^*, α^+, $\sim\alpha$, and $\alpha\beta$. The last of these is short for $\alpha \cdot \beta$.

We also define inductively which strings match each pattern. We have already said which strings match the atomic patterns. This is the basis of the inductive definition. Now suppose we have already defined the sets of strings $L(\alpha)$ and $L(\beta)$ matching α and β, respectively. Then we'll say that

- x matches $\alpha + \beta$ if x matches either α or β:
$$L(\alpha + \beta) = L(\alpha) \cup L(\beta);$$
- x matches $\alpha \cap \beta$ if x matches both α and β:
$$L(\alpha \cap \beta) = L(\alpha) \cap L(\beta);$$
- x matches $\alpha\beta$ if x can be broken down as $x = yz$ such that y matches α and z matches β:
$$\begin{aligned} L(\alpha\beta) &= L(\alpha)L(\beta) \\ &= \{yz \mid y \in L(\alpha) \text{ and } z \in L(\beta)\}; \end{aligned}$$
- x matches $\sim\alpha$ if x does not match α:
$$\begin{aligned} L(\sim\alpha) &= \sim L(\alpha) \\ &= \Sigma^* - L(\alpha); \end{aligned}$$
- x matches α^* if x can be expressed as a concatenation of zero or more strings, all of which match α:
$$\begin{aligned} L(\alpha^*) &= \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in L(\alpha),\ 1 \leq i \leq n\} \\ &= L(\alpha)^0 \cup L(\alpha)^1 \cup L(\alpha)^2 \cup \cdots \end{aligned}$$

$L(\alpha^*) = L(\alpha)^*$.

The null string ϵ always matches α^*, since ϵ is a concatenation of zero strings, all of which (vacuously) match α.

- x matches α^+ if x can be expressed as a concatenation of one or more strings, all of which match α:

$$\begin{aligned} L(\alpha^+) &= \{x_1 x_2 \cdots x_n \mid n \geq 1 \text{ and } x_i \in L(\alpha),\ 1 \leq i \leq n\} \\ &= L(\alpha)^1 \cup L(\alpha)^2 \cup L(\alpha)^3 \cup \cdots \\ &= L(\alpha)^+. \end{aligned}$$

Note that patterns are just certain strings of symbols over the alphabet

$$\Sigma \cup \{\epsilon, \emptyset, \#, @, +, \cap, \sim, {}^*, {}^+, (,)\}.$$

Note also that the meanings of $\#$, $@$, and $\sim$ depend on Σ. For example, if $\Sigma = \{a, b, c\}$ then $L(\#) = \{a, b, c\}$, but if $\Sigma = \{a\}$ then $L(\#) = \{a\}$.

Example 7.1

- $\Sigma^* = L(@) = L(\#^*)$.
- Singleton sets: if $x \in \Sigma^*$, then x itself is a pattern and is matched only by the string x; i.e., $\{x\} = L(x)$.
- Finite sets: if $x_1, \ldots, x_m \in \Sigma^*$, then

$$\{x_1, x_2, \ldots, x_m\} = L(x_1 + x_2 + \cdots + x_m). \qquad \square$$

Note that we can write the last pattern $x_1 + x_2 + \cdots + x_m$ without parentheses, since the two patterns $(\alpha + \beta) + \gamma$ and $\alpha + (\beta + \gamma)$ are matched by the same set of strings; i.e.,

$$L((\alpha + \beta) + \gamma) = L(\alpha + (\beta + \gamma)).$$

Mathematically speaking, the operator $+$ is *associative*. The concatenation operator $\cdot$ is associative, too. Hence we can also unambiguously write $\alpha\beta\gamma$ without parentheses.

Example 7.2

- strings containing at least three occurrences of a:

  $@a@a@a@$;

- strings containing an a followed later by a b; that is, strings of the form $xaybz$ for some x, y, z:

  $@a@b@$;

- all single letters except a:

  $\# \cap \sim a$;

- strings with no occurrence of the letter a:

  $(\# \cap \sim a)^*$;

- strings in which every occurrence of a is followed sometime later by an occurrence of b; in other words, strings in which there are either no occurrences of a, or there is an occurrence of b followed by no occurrence of a; for example, aab matches but bba doesn't:

  $$(\# \cap \sim a)^* + @b(\# \cap \sim a)^*.$$

  If the alphabet is $\{a, b\}$, then this takes a much simpler form:

  $$\epsilon + @b.$$ □

Before we go too much further, there is a subtlety that needs to be mentioned. Note the slight difference in appearance between $\boldsymbol{\epsilon}$ and ϵ and between $\boldsymbol{\emptyset}$ and $\varnothing$. The objects $\boldsymbol{\epsilon}$ and $\boldsymbol{\emptyset}$ are *symbols* in the language of patterns, whereas ϵ and $\varnothing$ are *metasymbols* that we are using to name the null string and the empty set, respectively. These are different sorts of things: $\boldsymbol{\epsilon}$ and $\boldsymbol{\emptyset}$ are symbols, that is, strings of length one, whereas ϵ is a string of length zero and $\varnothing$ isn't even a string.

We'll maintain the distinction for a few lectures until we get used to the idea, but at some point in the near future we'll drop the boldface and use ϵ and $\varnothing$ exclusively. We'll always be able to infer from context whether we mean the symbols or the metasymbols. This is a little more convenient and conforms to standard usage, but bear in mind that they are still different things.

While we're on the subject of abuse of notation, we should also mention that very often you will see things like $x \in a^*b^*$ in texts and articles. Strictly speaking, one should write $x \in L(a^*b^*)$, since a^*b^* is a pattern, not a set of strings. But as long as you know what you really mean and can stand the guilt, it is okay to write $x \in a^*b^*$.

$\alpha^+ = \alpha \cdot \alpha^*$

$\alpha^* = \epsilon + \alpha^*$

$= \epsilon + \alpha + \alpha^*$

Lecture 8

Pattern Matching and Regular Expressions

rm *.txt

Here are some interesting and important questions:

- How hard is it to determine whether a given string x matches a given pattern α? This is an important practical question. There are very efficient algorithms, as we will see.
- Is every set represented by some pattern? Answer: no. For example, the set

  $$\{a^n b^n \mid n \geq 0\}$$

  is not represented by any pattern. We'll prove this later.
- Patterns α and β are *equivalent* if $L(\alpha) = L(\beta)$. How do you tell whether α and β are equivalent? Sometimes it is obvious and sometimes not.
- Which operators are redundant? For example, we can get rid of ϵ since it is equivalent to $\sim(\#@)$ and also to $\emptyset^*$. We can get rid of @ since it is equivalent to $\#^*$. We can get rid of unary $^+$ since α^+ is equivalent to $\alpha\alpha^*$. We can get rid of #, since if $\Sigma = \{a_1, \ldots, a_n\}$ then # is equivalent to the pattern

  $$a_1 + a_2 + \cdots + a_n.$$

$\alpha^+ = \alpha \cdot \alpha^*$

$+ \ldots$

$= 1 - 1 + 1 - 1 + 1 - 1 + \ldots$

$1 + 1 + 1 + 1 + \ldots$

$x \in \bigcup_{i=1}^{\infty} S_i = S_1 \cup S_2 \cup S_3 \cdots$

The operator $\cap$ is also redundant, by one of the De Morgan laws:

$$\alpha \cap \beta \text{ is equivalent to } \sim(\sim\alpha + \sim\beta).$$

Redundancy is an important question. From a user's point of view, we would like to have a lot of operators since this lets us write more succinct patterns; but from a programmer's point of view, we would like to have as few as possible since there is less code to write. Also, from a theoretical point of view, fewer operators mean fewer cases we have to treat in giving formal semantics and proofs of correctness.

An amazing and difficult-to-prove fact is that the operator $\sim$ is redundant. Thus every pattern is equivalent to one using only atomic patterns $a \in \Sigma$, ϵ, $\emptyset$, and operators $+$, $\cdot$, and *. Patterns using only these symbols are called *regular expressions*. Actually, as we have observed, even ϵ is redundant, but we include it in the definition of regular expressions because it occurs so often.

Our goal for this lecture and the next will be to show that the family of subsets of Σ^* represented by patterns is exactly the family of regular sets. Thus as a way of describing subsets of Σ^*, finite automata, patterns, and regular expressions are equally expressive.

Some Notational Conveniences

Since the binary operators $+$ and $\cdot$ are associative, that is,

$$L(\alpha + (\beta + \gamma)) = L((\alpha + \beta) + \gamma),$$
$$L(\alpha(\beta\gamma)) = L((\alpha\beta)\gamma),$$

we can write

$$\alpha + \beta + \gamma \quad \text{and} \quad \alpha\beta\gamma$$

without ambiguity. To resolve ambiguity in other situations, we assign precedence to operators. For example,

$$\alpha + \beta\gamma$$

could be interpreted as either

$$\alpha + (\beta\gamma) \quad \text{or} \quad (\alpha + \beta)\gamma,$$

which are not equivalent. We adopt the convention that the concatenation operator $\cdot$ has higher precedence than $+$, so that we would prefer the former interpretation. Similarly, we assign * higher precedence than $+$ or $\cdot$, so that

$$\alpha + \beta^*$$

is interpreted as

$$\alpha + (\beta^*)$$

and not as

$$(\alpha + \beta)^*.$$

All else failing, use parentheses.

Equivalence of Patterns, Regular Expressions, and Finite Automata

Patterns, regular expressions (patterns built from atomic patterns $a \in \Sigma$, ϵ, $\emptyset$, and operators $+$, *, and $\cdot$ only), and finite automata are all equivalent in expressive power: they all represent the regular sets.

Theorem 8.1 *Let $A \subseteq \Sigma^*$. The following three statements are equivalent:*

(i) A is regular; that is, $A = L(M)$ for some finite automaton M;

(ii) $A = L(\alpha)$ for some pattern α;

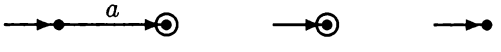
(iii) $A = L(\alpha)$ for some regular expression α.

Proof. The implication (iii) $\Rightarrow$ (ii) is trivial, since every regular expression is a pattern. We prove (ii) $\Rightarrow$ (i) here and (i) $\Rightarrow$ (iii) in Lecture 9.

The heart of the proof (ii) $\Rightarrow$ (i) involves showing that certain basic sets (corresponding to atomic patterns) are regular, and the regular sets are closed under certain closure operations corresponding to the operators used to build patterns. Note that

- the singleton set $\{a\}$ is regular, $a \in \Sigma$,
- the singleton set $\{\epsilon\}$ is regular, and
- the empty set $\emptyset$ is regular,

since each of these sets is the set accepted by some automaton. Here are nondeterministic automata for these three sets, respectively:



Also, we have previously shown that the regular sets are closed under the set operations $\cup$, $\cap$, $\sim$, $\cdot$, *, and $^+$; that is, if A and B are regular sets, then so are $A \cup B$, $A \cap B$, $\sim A = \Sigma^* - A$, AB, A^*, and A^+.

These facts can be used to prove inductively that (ii) $\Rightarrow$ (i). Let α be a given pattern. We wish to show that $L(\alpha)$ is a regular set. We proceed by

induction on the structure of α. The pattern α is of one of the following forms:

(i) a, where $a \in \Sigma$;
(ii) ϵ;
(iii) $\emptyset$;
(iv) $\#$;
(v) @;
(vi) $\beta + \gamma$;
(vii) $\beta \cap \gamma$;
(viii) $\beta\gamma$;
(ix) $\sim\beta$;
(x) β^*;
(xi) β^+.

There are five base cases (i) through (v) corresponding to the atomic patterns and six induction cases (vi) through (xi) corresponding to compound patterns. Each of these cases uses a closure property of the regular sets previously observed.

For (i), (ii), and (iii), we have $L(a) = \{a\}$ for $a \in \Sigma$, $L(\epsilon) = \{\epsilon\}$, and $L(\emptyset) = \varnothing$, and these are regular sets.

For (iv), (v), and (xi), we observed earlier that the operators #, @, and $^+$ were redundant, so we may disregard these cases since they are already covered by the other cases.

For (vi), recall that $L(\beta+\gamma) = L(\beta) \cup L(\gamma)$ by definition of the + operator. By the induction hypothesis, $L(\beta)$ and $L(\gamma)$ are regular. Since the regular sets are closed under union, $L(\beta + \gamma) = L(\beta) \cup L(\gamma)$ is also regular.

The arguments for the remaining cases (vii) through (x) are similar to the argument for (vi). Each of these cases uses a closure property of the regular sets that we have observed previously in Lectures 4 and 6. □

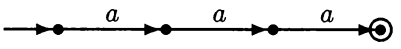
Example 8.2 Let's convert the regular expression

$$(aaa)^* + (aaaaa)^*$$

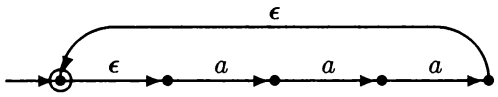
for the set

$$\{x \in \{a\}^* \mid |x| \text{ is divisible by either 3 or 5}\}$$

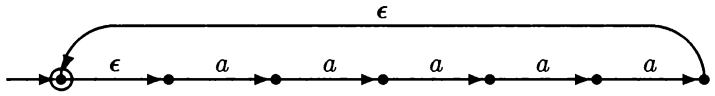
to an equivalent NFA. First we show how to construct an automaton for $(aaa)^*$. We take an automaton accepting only the string aaa, say



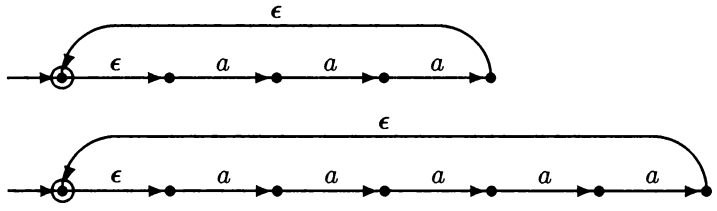
Applying the construction of Lecture 6, we add a new start state and ϵ-transitions from the new start state to all the old start states and from all the old accept states to the new start state. We let the new start state be the only accept state of the new automaton. This gives



The construction for $(aaaaa)^*$ is similar, giving



To get an NFA for $(aaa)^* + (aaaaa)^*$, we can simply take the disjoint union of these two automata:



□

Lecture 9

Regular Expressions and Finite Automata

Simplification of Expressions

For small regular expressions, one can often see how to construct an equivalent automaton directly without going through the mechanical procedure of the previous lecture. It is therefore useful to try to simplify the expression first.

For regular expressions α, β, if $L(\alpha) = L(\beta)$, we write $\alpha \equiv \beta$ and say that α and β are *equivalent*. The relation $\equiv$ on regular expressions is an equivalence relation; that is, it is

- reflexive: $\alpha \equiv \alpha$ for all α;
- symmetric: if $\alpha \equiv \beta$, then $\beta \equiv \alpha$; and
- transitive: if $\alpha \equiv \beta$ and $\beta \equiv \gamma$, then $\alpha \equiv \gamma$.

If $\alpha \equiv \beta$, one can substitute α for β (or vice versa) in any regular expression, and the resulting expression will be equivalent to the original.

Here are a few laws that can be used to simplify regular expressions.

$$\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma \tag{9.1}$$

$$\alpha + \beta \equiv \beta + \alpha \tag{9.2}$$

$\forall$ FSA M

$\exists$ pattern α_M

$L(M) = L(\alpha_M)$

$$\alpha + \emptyset \equiv \alpha \tag{9.3}$$
$$\alpha + \alpha \equiv \alpha \tag{9.4}$$
$$\alpha(\beta\gamma) \equiv (\alpha\beta)\gamma \tag{9.5}$$
$$\epsilon\alpha \equiv \alpha\epsilon \equiv \alpha \tag{9.6}$$
$$\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma \tag{9.7}$$
$$(\alpha + \beta)\gamma \equiv \alpha\gamma + \beta\gamma \tag{9.8}$$
$$\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset \tag{9.9}$$
$$\epsilon + \alpha\alpha^* \equiv \alpha^* \tag{9.10}$$
$$\epsilon + \alpha^*\alpha \equiv \alpha^* \tag{9.11}$$
$$\beta + \alpha\gamma \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma \tag{9.12}$$
$$\beta + \gamma\alpha \leq \gamma \Rightarrow \beta\alpha^* \leq \gamma \tag{9.13}$$

In (9.12) and (9.13), $\leq$ refers to the subset order:

$$\begin{aligned} \alpha \leq \beta &\stackrel{\text{def}}{\iff} L(\alpha) \subseteq L(\beta) \\ &\iff L(\alpha + \beta) = L(\beta) \\ &\iff \alpha + \beta \equiv \beta. \end{aligned}$$

Laws (9.12) and (9.13) are not equations but rules from which one can derive equations from other equations. Laws (9.1) through (9.13) can be justified by replacing each expression by its definition and reasoning set theoretically.

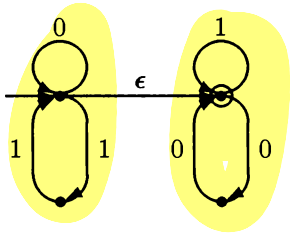
Here are some useful equations that follow from (9.1) through (9.13) that you can use to simplify expressions.

$$(\alpha\beta)^*\alpha \equiv \alpha(\beta\alpha)^* \tag{9.14}$$
$$(\alpha^*\beta)^*\alpha^* \equiv (\alpha + \beta)^* \tag{9.15}$$
$$\alpha^*(\beta\alpha^*)^* \equiv (\alpha + \beta)^* \tag{9.16}$$
$$(\epsilon + \alpha)^* \equiv \alpha^* \tag{9.17}$$
$$\alpha\alpha^* \equiv \alpha^*\alpha \tag{9.18}$$

An interesting fact that is beyond the scope of this course is that all true equations between regular expressions can be proved purely algebraically from the axioms and rules (9.1) through (9.13) plus the laws of equational logic [73].

To illustrate, let's convert some regular expressions to finite automata.

Example 9.1 $(11+0)^*(00+1)^*$



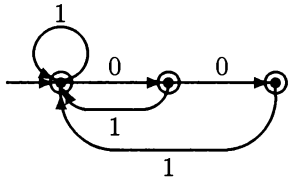
This expression is simple enough that the easiest thing to do is eyeball it. The mechanical method described in Lecture 8 would give more states and ϵ-transitions than shown here. The two states connected by an ϵ-transition cannot be collapsed into one state, since then 10 would be accepted, which does not match the regular expression. □

Example 9.2 $(1+01+001)^*(\epsilon+0+00)$

Using the algebraic laws above, we can rewrite the expression:

$$(1+01+001)^*(\epsilon+0+00) \equiv ((\epsilon+0+00)1)^*(\epsilon+0+00)$$
$$\equiv ((\epsilon+0)(\epsilon+0)1)^*(\epsilon+0)(\epsilon+0).$$

It is now easier to see that the set represented is the set of all strings over $\{0,1\}$ with no substring of more than two adjacent 0's.



□

Just because all states of an NFA are accept states doesn't mean that all strings are accepted! Note that in Example 9.2, 000 is not accepted.

Converting Automata to Regular Expressions

To finish the proof of Theorem 8.1, it remains to show how to convert a given finite automaton M to an equivalent regular expression.

Given an NFA

$$M = (Q, \Sigma, \Delta, S, F),$$

a subset $X \subseteq Q$, and states $u, v \in Q$, we show how to construct a regular expression

$$\alpha_{uv}^{X} \qquad \sum_{u \in S, v \in F} \alpha_{u,v}^{Q}$$

for every FSA M there exists a pattern α s.t $L(\alpha) = L(M)$.

representing the set of all strings x such that there is a path from u to v in M labeled x (i.e., such that $v \in \widehat{\Delta}(\{u\}, x)$) and all states along that path, with the possible exception of u and v, lie in X.

The expressions are constructed inductively on the size of X. For the basis $X = \varnothing$, let $a_1, \ldots, a_k$ be all the symbols in Σ such that $v \in \Delta(u, a_i)$. For $u \neq v$, take

$$\alpha_{uv}^{\varnothing} \stackrel{\text{def}}{=} \begin{cases} a_1 + \cdots + a_k & \text{if } k \geq 1, \\ \emptyset & \text{if } k = 0; \end{cases}$$

and for $u = v$, take

$$\alpha_{uv}^{\varnothing} \stackrel{\text{def}}{=} \begin{cases} a_1 + \cdots + a_k + \epsilon & \text{if } k \geq 1, \\ \epsilon & \text{if } k = 0. \end{cases}$$

For nonempty X, we can choose any element $q \in X$ and take

$$\alpha_{uv}^{X} \stackrel{\text{def}}{=} \alpha_{uv}^{X-\{q\}} + \alpha_{uq}^{X-\{q\}} (\alpha_{qq}^{X-\{q\}})^* \alpha_{qv}^{X-\{q\}}. \tag{9.19}$$

To justify the definition (9.19), note that any path from u to v with all intermediate states in X either (i) never visits q, hence the expression

$$\alpha_{uv}^{X-\{q\}}$$

on the right-hand side of (9.19); or (ii) visits q for the first time, hence the expression

$$\alpha_{uq}^{X-\{q\}},$$

followed by a finite number (possibly zero) of loops from q back to itself without visiting q in between and staying in X, hence the expression

$$(\alpha_{qq}^{X-\{q\}})^*,$$

followed by a path from q to v after leaving q for the last time, hence the expression

$$\alpha_{qv}^{X-\{q\}}.$$

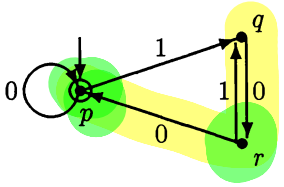
The sum of all expressions of the form

$$\sum_{s \in S,\, f \in F} \alpha_{sf}^{Q},$$

where s is a start state and f is a final state, represents the set of strings accepted by M.

As a practical rule of thumb when doing homework exercises, when choosing the $q \in X$ to drop out in (9.19), it is best to try to choose one that disconnects the automaton as much as possible.

Example 9.3 Let's convert the automaton



to an equivalent regular expression. The set accepted by this automaton will be represented by the inductively defined regular expression

$$\alpha_{pp}^{\{p,q,r\}},$$

since p is the only start and the only accept state. Removing the state q (we can choose any state we like here), we can take

$$\alpha_{pp}^{\{p,q,r\}} = \alpha_{pp}^{\{p,r\}} + \alpha_{pq}^{\{p,r\}}(\alpha_{qq}^{\{p,r\}})^*\alpha_{qp}^{\{p,r\}}.$$

Looking at the automaton, the only paths going from p to p and staying in the states $\{p, r\}$ are paths going around the single loop labeled 0 from p to p some finite number of times; thus we can take

$$\alpha_{pp}^{\{p,r\}} = 0^*.$$

By similar informal reasoning, we can take

$$\alpha_{pq}^{\{p,r\}} = 0^*1,$$

$$\begin{aligned}\alpha_{qq}^{\{p,r\}} &= \epsilon + 01 + 000^*1 \\ &\equiv \epsilon + 0(\epsilon + 00^*)1 \\ &\equiv \epsilon + 00^*1,\end{aligned}$$

$$\alpha_{qp}^{\{p,r\}} = 000^*.$$

Thus we can take

$$\alpha_{pp}^{\{p,q,r\}} = 0^* + 0^*1(\epsilon + 00^*1)^*000^*.$$

This is matched by the set of all strings accepted by the automaton. We can further simplify the expression using the algebraic laws (9.1) through (9.18):

$$\begin{aligned}&0^* + 0^*1(\epsilon + 00^*1)^*000^* \\ &\equiv 0^* + 0^*1(00^*1)^*000^* && \text{by (9.17)} \\ &\equiv \epsilon + 00^* + 0^*10(0^*10)^*00^* && \text{by (9.10) and (9.14)} \\ &\equiv \epsilon + (\epsilon + 0^*10(0^*10)^*)00^* && \text{by (9.8)} \\ &\equiv \epsilon + (0^*10)^*00^* && \text{by (9.10)} \\ &\equiv \epsilon + (0^*10)^*0^*0 && \text{by (9.18)} \\ &\equiv \epsilon + (0 + 10)^*0 && \text{by (9.15).}\end{aligned}$$

□

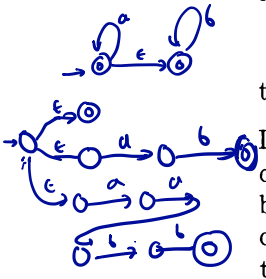
Historical Notes

Kleene [70] proved that deterministic finite automata and regular expressions are equivalent. A shorter proof was given by McNaughton and Yamada [85].

The relationship between right- and left-linear grammars and regular sets (Homework 5, Exercise 1) was observed by Chomsky and Miller [21].

Lecture 11

Limitations of Finite Automata



We have studied what finite automata can do; let's see what they cannot do. The canonical example of a nonregular set (one accepted by no finite automaton) is

$$B = \{a^n b^n \mid n \geq 0\} = \{\epsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\},$$

the set of all strings of the form a^*b^* with equally many a's and b's.

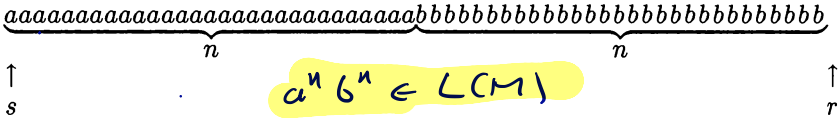
Intuitively, in order to accept the set B, an automaton scanning a string of the form a^*b^* would have to remember when passing the center point between the a's and b's how many a's it has seen, since it would have to compare that with the number of b's and accept iff the two numbers are the same.

$$aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbb$$
$$\uparrow$$
$$q$$

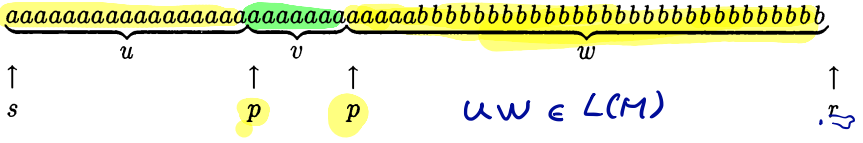
Moreover, it would have to do this for arbitrarily long strings of a's and b's, much longer than the number of states. This is an unbounded amount of information, and there is no way it can remember this with only finite memory. All it "knows" at that point is represented in the state q it is in, which is only a finite amount of information. You might at first think there may be some clever strategy, such as counting mod 3, 5, and 7, or something similar. But any such attempt is doomed to failure: you cannot

distinguish between infinitely many different cases with only finitely many states.

This is just an informal argument. But we can easily give a formal proof by contradiction that B is not regular. Assuming that B were regular, there would be a DFA M such that $L(M) = B$. Let k be the number of states of this alleged M. Consider the action of M on input $a^n b^n$, where $n \gg k$. It starts in its start state s. Since the string $a^n b^n$ is in B, M must accept it, thus M must be in some final state r after scanning $a^n b^n$.



Since $n \gg k$, by the pigeonhole principle there must be some state p that the automaton enters more than once while scanning the initial sequence of a's. Break up the string $a^n b^n$ into three pieces u, v, w, where v is the string of a's scanned between two occurrences of the state p, as illustrated in the following picture:

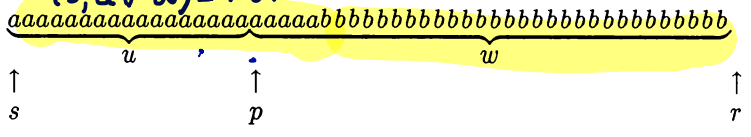


Let $j = |v| > 0$. In this example, $j = 7$. Then

$$\hat{\delta}(s, u) = p,$$
$$\hat{\delta}(p, v) = p,$$
$$\hat{\delta}(p, w) = r \in F.$$

The string v could be deleted and the resulting string would be erroneously accepted:

$$\begin{aligned}\hat{\delta}(s, uw) &= \hat{\delta}(\hat{\delta}(s, u), w) \\ &= \hat{\delta}(p, w) \\ &= r \in F.\end{aligned}$$



It's erroneous because after deleting v, the number of a's is strictly less than the number of b's: $uw = a^{n-j} b^n \in L(M)$, but $uw \notin B$. This contradicts our assumption that $L(M) = B$.

We could also insert extra copies of v and the resulting string would be erroneously accepted. For example, $uv^3w = a^{n+2j}b^n$ is erroneously accepted:

$$\begin{aligned}\widehat{\delta}(s, uvvvw) &= \widehat{\delta}(\widehat{\delta}(\widehat{\delta}(\widehat{\delta}(\widehat{\delta}(s,u),v),v),v),w)\\ &= \widehat{\delta}(\widehat{\delta}(\widehat{\delta}(\widehat{\delta}(p,v),v),v),w)\\ &= \widehat{\delta}(\widehat{\delta}(\widehat{\delta}(p,v),v),w)\\ &= \widehat{\delta}(\widehat{\delta}(p,v),w)\\ &= \widehat{\delta}(p,w)\\ &= r \in F.\end{aligned}$$

For another example of a nonregular set, consider

$$\begin{aligned}C &\stackrel{\text{def}}{=} \{a^{2^n} \mid n \geq 0\}\\ &= \{x \in \{a\}^* \mid |x| \text{ is a power of } 2\}\\ &= \{a, a^2, a^4, a^8, a^{16}, \ldots\}.\end{aligned}$$

This set is also nonregular. Suppose (again for a contradiction) that $L(M) = C$ for some DFA M. Let k be the number of states of M. Let $n \gg k$ and consider the action of M on input $a^{2^n} \in C$. Since $n \gg k$, by the pigeonhole principle the automaton must repeat a state p while scanning the first n symbols of a^{2^n}. Thus $2^n = i + j + m$ for some i, j, m with $0 < j \leq n$ and

$$\begin{aligned}\widehat{\delta}(s, a^i) &= p,\\ \widehat{\delta}(p, a^j) &= p,\\ \widehat{\delta}(p, a^m) &= r \in F.\end{aligned}$$

$$\underbrace{aaaaaaaaaaaaaaaa}_{i}\underbrace{aaaaaaa}_{j}\underbrace{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}_{m}$$

$$\begin{matrix}\uparrow & & \uparrow & & \uparrow & & \uparrow\\ s & & p & & p & & r\end{matrix}$$

As above, we could insert an extra a^j to get a^{2^n+j}, and this string would be erroneously accepted:

$$\begin{aligned}\widehat{\delta}(s, a^{2^n+j}) &= \widehat{\delta}(s, a^i a^j a^j a^m)\\ &= \widehat{\delta}(\widehat{\delta}(\widehat{\delta}(\widehat{\delta}(s,a^i),a^j),a^j),a^m)\\ &= \widehat{\delta}(\widehat{\delta}(\widehat{\delta}(p,a^j),a^j),a^m)\\ &= \widehat{\delta}(\widehat{\delta}(p,a^j),a^m)\\ &= \widehat{\delta}(p,a^m)\\ &= r \in F.\end{aligned}$$

This is erroneous because $2^n + j$ is not a power of 2:

$$2^n + j \leq 2^n + n$$

$< 2^n + 2^n$
$= 2^{n+1}$

and 2^{n+1} is the next power of 2 greater than 2^n.

The Pumping Lemma

We can encapsulate the arguments above in a general theorem called the *pumping lemma*. This lemma is very useful in proving sets nonregular. The idea is that whenever an automaton scans a long string (longer than the number of states) and accepts, there must be a repeated state, and extra copies of the segment of the input between the two occurrences of that state can be inserted and the resulting string is still accepted.

Theorem 11.1 **(Pumping lemma)** *Let A be a regular set. Then the following property holds of A:*

(P) *There exists $k \geq 0$ such that for any strings x, y, z with $xyz \in A$ and $|y| \geq k$, there exist strings u, v, w such that $y = uvw$, $v \neq \epsilon$, and for all $i \geq 0$, the string $xuv^iwz \in A$.*

Informally, if A is regular, then for any string in A and any sufficiently long substring y of that string, y has a nonnull substring v of which you can pump in as many copies as you like and the resulting string is still in A.

We have essentially already proved this theorem. Think of k as the number of states of a DFA accepting A. Since y is at least as long as the number of states, there must be a repeated state while scanning y. The string v is the substring between the two occurrences of that state. We can pump in as many copies of v as we want (or delete v—this would be the case $i = 0$), and the resulting string is still accepted.

Games with the Demon

The pumping lemma is often used to show that certain sets are nonregular. For this purpose we usually use it in its contrapositive form:

Theorem 11.2 **(Pumping lemma, contrapositive form)** *Let A be a set of strings. Suppose that the following property holds of A.*

(¬P) *For all $k \geq 0$ there exist strings x, y, z such that $xyz \in A$, $|y| \geq k$, and for all u, v, w with $y = uvw$ and $v \neq \epsilon$, there exists an $i \geq 0$ such that $xuv^iwz \notin A$.*

Then A is not regular.

To use the pumping lemma to prove that a given set A is nonregular, we need to establish that (¬P) holds of A. Because of the alternating "for all/there exists" form of (¬P), we can think of this as a game between you and a demon. You want to show that A is nonregular, and the demon wants to show that A is regular. The game proceeds as follows:

1. The demon picks k. (If A really is regular, the demon's best strategy here is to pick k to be the number of states of a DFA for A.)
2. You pick x, y, z such that $xyz \in A$ and $|y| \geq k$.
3. The demon picks u, v, w such that $y = uvw$ and $v \neq \epsilon$.
4. You pick $i \geq 0$.

You win if $xuv^iwz \notin A$, and the demon wins if $xuv^iwz \in A$.

The property (¬P) for A is equivalent to saying that you have a *winning strategy* in this game. This means that by playing optimally, you can always win no matter what the demon does in steps 1 and 3.

If you can show that you have a winning strategy, you have essentially shown that the condition (¬P) holds for A, therefore by Theorem 11.2, A is not regular.

We have thus reduced the problem of showing that a given set is nonregular to the puzzle of finding a winning strategy in the corresponding demon game. Each nonregular set gives a different game. We'll give several examples in Lecture 12.

Warning: Although there do exist stronger versions that give necessary and sufficient conditions for regularity (Miscellaneous Exercise 44), the version of the pumping lemma given here gives only a necessary condition; there exist sets satisfying (P) that are nonregular (Miscellaneous Exercise 43). You cannot show that a set *is* regular by showing that it satisfies (P). To show a given set *is* regular, you should construct a finite automaton or regular expression for it.

Historical Notes

The pumping lemma for regular sets is due to Bar-Hillel, Perles, and Shamir [8]. This version gives only a necessary condition for regularity. Necessary and sufficient conditions are given by Stanat and Weiss [117], Jaffe [62], and Ehrenfeucht, Parikh, and Rozenberg [33].

① Demon picks K.

$$A = \{a^p \mid p \text{ is prime}\}$$

② You pick $xyz \in A$, $|y| \geq K$

$K = 6$ $\quad y = a^p$ where p is the smallest prime $> K$

$z = 6$

③ Demon pick uvw. $y = uvw$. $v \neq \varepsilon$

$y = a^p$

④ You pick $i = p+1$ $\quad y = u\,v\,w$

$|u| = j$, $|v| = m$, $|w| = n$

You win if $uv^i w \notin A$ $\quad j + m + n = p$

Demon wins. otherwise

$$j + im + n = p + (i-1)m$$

Take $i = p+1$

$i = p-1$

$uv^i w \notin A$

$p + (i-1)m = p + pm$

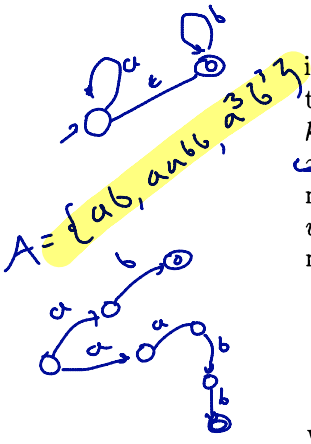
$= p(1+m)$

$a^Q_{1,2} \cup a^Q_{1,3}$

$+ = \cup$

Lecture 12

Using the Pumping Lemma



Example 12.1 Let's use the pumping lemma in the form of the demon game to show that the set

$$A = \{a^n b^m \mid n \geq m\}$$

is not regular. The set A is the set of strings in a^*b^* with no more b's than a's. The demon, who is betting that A is regular, picks some number k. A good response for you is to pick $x = a^k$, $y = b^k$, and $z = \epsilon$. Then $xyz = a^k b^k \in A$ and $|y| = k$; so far you have followed the rules. The demon must now pick u, v, w such that $y = uvw$ and $v \neq \epsilon$. Say the demon picks u, v, w of length j, m, n, respectively, with $k = j + m + n$ and $m > 0$. No matter what the demon picks, you can take $i = 2$ and you win:

$$\begin{aligned} xuv^2wz &= a^k b^j b^m b^m b^n \\ &= a^k b^{j+2m+n} \\ &= a^k b^{k+m}, \end{aligned}$$

which is not in A, because the number of b's is strictly larger than the number of a's.

This strategy always leads to victory for you in the demon game associated with the set A. As we argued in Lecture 11, this is tantamount to showing that A is nonregular. □

Example 12.2 For another example, take the set

$$C = \{a^{n!} \mid n \geq 0\}.$$

We would like to show that this set is not regular. This one is a little harder. It is an example of a nonregular set over a single-letter alphabet. Intuitively, it is not regular because the differences in the lengths of the successive elements of the set grow too fast.

Suppose the demon chooses k. A good choice for you is $x = z = \epsilon$ and $y = a^{k!}$. Then $xyz = a^{k!} \in C$ and $|y| = k! \geq k$, so you have not cheated. The demon must now choose u, v, w such that $y = uvw$ and $v \neq \epsilon$. Say the demon chooses u, v, w of length j, m, n, respectively, with $k! = j + m + n$ and $m > 0$. You now need to find i such that $xuv^iwz \notin C$; in other words, $|xuv^iwz| \neq p!$ for any p. Note that for any i,

$$|xuv^iwz| = j + im + n = k! + (i-1)m,$$

so you will win if you can choose i such that $k! + (i-1)m \neq p!$ for any p. Take $i = (k+1)! + 1$. Then

$$k! + (i-1)m = k! + (k+1)!m = k!(1 + m(k+1)),$$

and we want to show that this cannot be $p!$ for any p. But if

$$p! = k!(1 + m(k+1)),$$

then we could divide both sides by $k!$ to get

$$p(p-1)(p-2)\cdots(k+2)(k+1) = 1 + m(k+1),$$

which is impossible, because the left-hand side is divisible by $k+1$ and the right-hand side is not. □

A Trick

When trying to show that a set is nonregular, one can often simplify the problem by using one of the closure properties of regular sets. This often allows us to reduce a complicated set to a simpler set that is already known to be nonregular, thereby avoiding the use of the pumping lemma.

To illustrate, consider the set

$$D = \{x \in \{a, b\}^* \mid \#a(x) = \#b(x)\}.$$

To show that this set is nonregular, suppose for a contradiction that it were regular. Then the set

$$D \cap a^*b^*$$

would also be regular, since the intersection of two regular sets is always regular (the product construction, remember?). But

$$D \cap L(a^*b^*) = \{a^n b^n \mid n \geq 0\},$$

which we have already shown to be nonregular. This is a contradiction.

For another illustration of this trick, consider the set A of Example 12.1 above:

$$A = \{a^n b^m \mid n \geq m\},$$

the set of strings $x \in L(a^*b^*)$ with no more b's than a's. By Exercise 2 of Homework 2, if A were regular, then so would be the set

$$\mathbf{rev}\ A = \{b^m a^n \mid n \geq m\},$$

and by interchanging a and b, we would get that the set

$$A' = \{a^m b^n \mid n \geq m\}$$

is also regular. Formally, "interchanging a and b" means applying the homomorphism $a \mapsto b$, $b \mapsto a$. But then the intersection

$$A \cap A' = \{a^n b^n \mid n \geq 0\}$$

would be regular. But we have already shown using the pumping lemma that this set is nonregular. This is a contradiction.

Ultimate Periodicity

Let U be a subset of $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$, the natural numbers.

The set U is said to be *ultimately periodic* if there exist numbers $n \geq 0$ and $p > 0$ such that for all $m \geq n$, $m \in U$ if and only if $m + p \in U$. The number p is called a *period* of U.

In other words, except for a finite initial part (the numbers less than n), numbers are in or out of the set U according to a repeating pattern. For example, consider the set

$$\{0, 3, 7, 11, 19, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, \ldots\}.$$

Starting at 20, every third element is in the set, therefore this set is ultimately periodic with $n = 20$ and $p = 3$. Note that neither n nor p is unique; for example, for this set we could also have taken $n = 21$ and $p = 6$, or $n = 100$ and $p = 33$.

Regular sets over a single-letter alphabet $\{a\}$ and ultimately periodic subsets of $\mathbb{N}$ are strongly related: