Another reduction from NFA-Λ's to regular expressions

The proof of Kleene’s theorem implies a construction of regular expressions from FA’s. But it’s not very nice to carry out by hand.

We’ll specify another, nicer construction — one that takes any NFA-Λ to a corresponding regular expression . . .

Without going into details, imagine that we extend our notion of NFA-Λ to allow transitions labeled with arbitrary regular expressions.

We will specify a construction that takes any such “extended” NFA-Λ (and so, in particular, any standard NFA-Λ) to one with only one transition and two states: an initial, non-accepting state and an accepting state.

The label of the transition will be a regular expression for the language accepted by the original NFA-Λ . . .

Repeat until all original states are eliminated:

1. Select an original state q to eliminate.

2. For each pair \( (p, r) \) of states s.t. \( p \neq q \) and \( r \neq q \), replace \( e(p, r) \) with

\[
e(p, r) + e(p, q) e(q, q)^* e(q, r).
\]

After all original states are eliminated in this manner, the only remaining states are the new start state and the new final state, and the transition from the former to the latter is labeled with a regular expression for the language of the original NFA-Λ.

In practice, you’ll probably want to simplify the regular expressions obtained in step 2 as you go.

For example, if \( e(q, q) = \emptyset \), then the new \( e(p, r) \) is

\[
e(p, r) + e(p, q) e(q, r).
\]

Similarly, if either \( e(p, q) \) or \( e(q, r) \) is \( \emptyset \), then \( e(p, r) \) will be unchanged.

We transform the original NFA-Λ as follows:

First, if there is more than one transition from state \( q \) to state \( r \), for any states \( q \) and \( r \), replace it with a single transition labeled with an appropriate regular expression. (Use + to combine the relevant labels.)

(From now on in this construction we assume there is at most one labeled transition from state \( p \) to state \( q \), for all states \( p \) and \( q \).)

For each pair \( (p, q) \) of states, let \( e(p, q) \) denote the label on the transition from \( p \) to \( q \) — if there is no such transition, \( e(p, q) \) stands for the regular expression \( \emptyset \).

Next, add a new start state to the NFA-Λ, with a \( \Lambda \)-transition from the new start state to the former start state. Also add another new state, which will be the unique final state, and add \( \Lambda \)-transitions from the former final states to new final state.

The construction is completed by eliminating all of the original states, one by one, as follows:

Definition For any \( L \subseteq \Sigma \), \( I_L \) is the relation on \( \Sigma \) s.t. for all \( x, y \in \Sigma \),

\[
x I_L y \iff x \text{ and } y \text{ are indistinguishable wrt } L.
\]

Recall: \( x \) and \( y \) are distinguishable wrt \( L \) iff there is a \( z \in \Sigma^* \) s.t. \( x z \in L \) iff \( y z \in L \).

So \( x I_L y \) iff, for all \( z \in \Sigma^* \), \( x z \in L \) iff \( y z \in L \).

Lemma 5.1 For any \( L \subseteq \Sigma \), \( I_L \) is an equivalence relation.

Proof It is clear that \( I_L \) is reflexive and symmetric. To see that it is transitive, assume \( x I_L y \) and \( y I_L z \). Hence, for any \( w \in \Sigma^* \), \( x w \in L \) iff \( y w \in L \), and \( y w \in L \) iff \( z w \in L \), from which it follows that \( x w \in L \) iff \( z w \in L \). Consequently, \( x \) and \( z \) are indistinguishable wrt \( L \). That is, \( x I_L z \), from which we can conclude that \( I_L \) is transitive.

We will be interested in the equivalence classes of \( I_L \).

For any \( x \in \Sigma^* \), we will write \( [x] \) as an abbreviation for \( [x]_{I_L} \).
Example Consider the language
\[ L = \{(0 + 1)(0 + 1)^*\}. \]
Even length strings are indistinguishable wrt \( L \). Similarly, odd length strings are indistinguishable wrt \( L \). Hence,
\[ I_L = \{(x, y) \in \{0, 1\}^* \times \{0, 1\}^* \mid |xy| \text{ is even} \}. \]

Observation Recall the language \( pal \) over \( \{a, b\} \). Notice that
\[ I_{pal} = \{(x, x) \mid x \in \{a, b\}^*\}, \]
since all strings over \( \{a, b\} \) are distinguishable from one another wrt \( pal \). Therefore, for all \( x \in \{a, b\}^* \), \( |x| = |x| \).

Theorem 5.1 For any language \( L \) over \( \Sigma \), let
\[ Q_L = \{ |x| \mid x \in \Sigma^* \}, \quad A_L = \{ |x| \mid x \in L \} \]
and let \( \delta_L : Q_L \times \Sigma \rightarrow Q_L \) be the unique function s.t. for all \( x \in \Sigma^* \) and \( a \in \Sigma \)
\[ \delta_L(|x|, a) = |x|, \]
If \( Q_L \) is finite, then
\[ M_L = (Q_L, \Sigma, A_L, \delta_L) \]
is an FA that recognizes \( L \). Moreover, no FA that recognizes \( L \) has fewer states than \( M_L \).

Proof. By Lemma 5.2.1, there is indeed a unique function \( \delta_L \) as specified. And since \( Q_L \) is finite (by assumption), \( A_L \subseteq Q_L \) and \( |A| \in Q_L \), \( M_L \) is indeed an FA.
To show that \( L(M_L) = L \), we first show that, for all \( x \in \Sigma^* \),
\[ \delta_L(|x|, x) = |x|, \]
by structural induction on \( x \).

Case 1: \( \delta_L(|\lambda|, \lambda) = |\lambda| \) by the definition of \( \delta_L \) for FA's.

Case 2 \( x \in \Sigma^* \), \( a \in \Sigma \)
IH: \( \delta_L(|x|, x) = |x| \).
To show: \( \delta_L(|x|, ax) = |x| \).
\[ \delta_L(|x|, ax) = \delta_L(\delta_L(|x|, x), a) \quad \text{(def \( \delta_L \) for FA)} \]
\[ = \delta_L(|x|, a) \quad \text{(IH)} \]
\[ = |x| \quad \text{(def \( \delta_L \))} \]

Lemma 5.2 For any language \( L \) over \( \Sigma \), for all \( x, y \in \Sigma^* \) and \( a \in \Sigma \), if \( |x| = |y| \), then \( |xa| = |ya| \).

Proof. Assume \( |x| = |y| \). Hence, \( x \) and \( y \) are indistinguishable wrt \( L \). That is, for all \( x \in \Sigma^* \), \( xz \in L \) iff \( yz \in L \). It follows that for all \( a \in \Sigma \) and \( x, z \in \Sigma^* \), \( xza \in L \) iff \( yza \in L \). Hence, for any \( a \in \Sigma \), \( xa \) and \( ya \) are indistinguishable wrt \( L \). Therefore, \( |xa| = |ya| \).

Lemma 5.2.1 For any language \( L \) over \( \Sigma \), let
\[ Q_L = \{ |x| \mid x \in \Sigma^* \}, \quad A_L = \{ |x| \mid x \in L \} \]
and let \( \delta_L : Q_L \times \Sigma \rightarrow Q_L \) be the unique function s.t. for all \( x \in \Sigma^* \) and \( a \in \Sigma \)
\[ \delta_L(|x|, a) = |xa|, \]

Proof. It is clear that there can be at most one such function, so it remains only to show that such a function exists. Consider any \( x, y \in \Sigma^* \) s.t. \( |x| = |y| \). By Lemma 5.2, we know that, for any \( a \in \Sigma \), \( |xa| = |ya| \), from which we can conclude that there is such a function.

To complete the proof that \( L(M_L) = L \), consider any \( x \in \Sigma^* \).
\( M \) accepts \( x \) iff \( \delta_L(|\lambda|, x) \in A_L \) (def FA acceptance)
\[ \text{iff } |x| \in A_L \]
\[ \text{iff } x \in L \] (def \( A_L \))

So \( L(M_L) = L \).
To see that no FA accepting \( L \) has fewer states than \( M_L \), notice that for all \( x, y \in \Sigma^* \), if \( |x| \neq |y| \), then \( x \) and \( y \) are distinguishable wrt \( L \). It follows by Theorem 3.2 that any FA recognizing \( L \) must have at least \( |Q_L| \) states.

Corollary 5.1 A language \( L \) is regular iff the set of equivalence classes of \( I_L \) is finite.

Proof. Assume that the set of equivalence classes of \( I_L \) is infinite. Then from Theorem 3.2 it follows that there is no FA that accepts \( L \). And by Kleene’s Theorem, \( L \) is not regular.
Assume that the set of equivalence classes of \( I_L \) is finite. Then, by Theorem 5.1, there is an FA that accepts \( L \). And by Kleene’s Theorem, \( L \) is regular.
Theorem 5.1 suggests a precise description of what an FA must remember in order to recognize a language: it can forget everything about the prefix it has seen thus far except what equivalence class of $I_L$ it belongs to.

This is the main idea behind a construction that takes any FA $M$ to a minimal FA that accepts $L(M)$.

To begin, we can eliminate all states that are not reachable from the initial state. (It is straightforward to show that this can be done without affecting the language accepted.)

So we restrict our consideration in what follows to FA's such that, for every $q \in Q$ there is an $x \in \Sigma$ s.t. $\delta(q_0, x) = q$.

Given an FA that accepts $L$ (in which every state is reachable from the initial state), we would like to construct a minimal FA to recognize $L$.

**Claim** For all $q \in Q$, $L_q$ is a subset of a unique equivalence class of $I_{L(M)}$.

**Proof.** By the previous claim, $L_q$ is a subset of some equivalence class of $I_{L(M)}$. Suppose $L_q$ is a subset of both $[x]$ and $[y]$. Since $L_q$ is nonempty, there is a $z \in L_q$. Since $z \in [x]$, $|z| = |x|$. Since $z \in [y]$, $|z| = |y|$. Consequently $|x| = |y|$.

But in the minimal FA's constructed in Theorem 5.1, for every $q \in Q$, $L_q$ is equal to some equivalence class under $I_{L(M)}$.

So here's the idea: If $L_p$ and $L_q$ are subsets of the same equivalence class of $I_{L(M)}$, we should be able to "merge" states $p$ and $q$ without changing the language accepted by $M$.

**Definition** Given an FA $M = (Q, \Sigma, q_0, A, \delta)$, and any $p, q \in Q$, we will write

$p = q$ if $L_p$ and $L_q$ are subsets of the same equivalence class of $I_{L(M)}$.

We can verify that this $=$ is an equivalence relation on $Q$.

- Reflexivity and symmetry are immediate.
- For transitivity: Assume $p = q$ and $q = r$. So $L_p$ and $L_q$ are subsets of the same (unique!) equivalence class of $I_{L(M)}$. Similarly $L_q$ and $L_r$ are subsets of the same (unique!) equivalence class of $I_{L(M)}$. Consequently, $L_p$ and $L_r$ are subsets of the same equivalence class of $I_{L(M)}$. That is, $p = r$.

Given an FA $M = (Q, \Sigma, q_0, A, \delta)$, for any $q \in Q$, take

$L_q = \{ x \in \Sigma : \delta(q_0, x) = q \}$.

Notice that

$[ L_q : q \in Q ]$

is a partition of $\Sigma$. (Also notice that every element of this partition is nonempty since every state is reachable from the initial state.)

Of course

$\{ |x| : x \in \Sigma^* \}$

is also a partition of $\Sigma^*$. We'll see that every element of the first partition is a subset of an element of the second partition.

**Claim** For all $q \in Q$, for all $x \in L_q$, $L_q \subseteq [x]$.

**Proof.** Consider any $q \in Q$, and any $x \in L_q$. By the definition of $L_q$, for every $y \in L_q$, $\delta(q_0, y) = \delta(q_0, x)$. Hence, by Lemma 2.1, every member of $L_q$ is indistinguishable from $x$ wrt $I_{L(M)}$. It follows by the definition of $[x]$ that every member of $L_q$ belongs to $[x]$.

It follows from this claim that, for every $q \in Q$, there is an $x \in \Sigma^*$ s.t. $L_q \subseteq [x]$.

That is, for every $q \in Q$, $L_q$ is a subset of some equivalence class of $I_{L(M)}$.

In fact, since $L_q$ is nonempty (for all $q \in Q$), we can say something stronger.

The following lemma suggests that merging equivalent states can indeed yield a minimal FA.

**Lemma** Let $L$ be a regular language over $\Sigma$ that is recognized by an FA $M = (Q, \Sigma, q_0, A, \delta)$. As before, let $Q_n$ be the set of equivalence classes of $L_n$. Let $Q_m$ be the set of equivalence classes of $\equiv$. Then $Q_n = Q_m$.

**Proof.** Consider any $q \in Q$ and $x \in L_q$. Recall that $L_q \subseteq [x]$. It is not difficult to check that, for any $p \in Q$, if $p \equiv q$, then $L_p \subseteq [x]$.

Hence

$\bigcup_{x \in L_q} L_p \subseteq [x]$.

Consider any $y \in [x]$. There is a $p \in Q$ s.t. $y \in L_p$ and $L_p \subseteq [x]$. Hence $p \equiv q$, and $y \in \bigcup_{x \in L_q} L_p$. So

$\bigcup_{x \in L_q} L_p \subseteq [x]$.

Let $f : Q_m \rightarrow Q_n$ be s.t. $f(x) = \bigcup_{x \in L_q} L_p$. (The previous result shows that there is such a function from $Q_m$ to $Q_n$.) We complete the proof by showing that $f$ is bijective.

To see that $f$ is surjective, consider any $x \in \Sigma^*$. We need to show that $[x]$ is in the range of $f$. There is a $q \in Q$ s.t. $x \in L_q$ and $L_q \subseteq [x]$. And there is an $X \subseteq Q_n$ s.t. $q \in X$ and $f(X) = [x]$.

To see that $f$ is injective, consider $X, Y \subseteq Q_n$ s.t. $X \neq Y$. We need to show that $f(X) \neq f(Y)$. There is an $x \in \Sigma^*$ s.t. $f(x) = [x]$. It follows that there is a $q \in X$ s.t. $x \in L_q$. Since $X \neq Y$, and $X$ and $Y$ are equivalence classes, we can conclude that $X$ and $Y$ are disjoint. Hence $q \notin Y$. Since $q$ is the only state for which $x \in L_q$, we can conclude that $x \notin f(Y)$. Hence $f(Y) \neq f(X)$. 

Theorem Given an FA $M = (Q, \Sigma, q_0, A, \delta)$, let

$M_m = (Q_m, \Sigma, \{q_0\}, A_m, \delta_m)$,

where

$Q_m = \{ q \mid q \in Q \}$, $A_m = \{ q \mid q \in A \}$,

and $\delta_m : Q_m \times \Sigma \to Q_m$ is s.t. for all $q \in Q$ and $a \in \Sigma$,

$\delta_m([q_0], a) = [\delta(q, a)]$.

Then $L(M_m) = L(M)$, and no FA with fewer states than $M_m$ recognizes $L(M)$.

Proof. First we must show that the definition of $\delta_m$ is sound; that is, such a function exists and is unique.

In fact, uniqueness is obvious.

To prove that such a function exists, we need to show that for all $p, q \in Q$ and $a \in \Sigma$, if $\delta(p, a) = \delta(q, a)$, then $\delta(p, a) = \delta(q, a)$. That is, if $p = q$, then $\delta(p, a) = \delta(q, a)$.

So assume that $p = q$. Consider any $x \in L_p$ and $y \in L_q$. (Recall: these sets are nonempty since we assume that all states are reachable.) Clearly $x = y \in L_{\delta(p, a)}$ and $y = y \in L_{\delta(q, a)}$. It follows that $L_{\delta(p, a)} \subseteq \{x\}$ and $L_{\delta(q, a)} \subseteq \{y\}$. We wish to show that $x = y$, from which it follows that $\delta(p, a) = \delta(q, a)$. Since $p = q$, $L_p$ and $L_q$ are subsets of the same equivalence class of $L$. Hence $|x| = |y|$. By Lemma 5.2, if $|x| = |y|$, then $|x| = |y|$.

Theorem Given an FA $M = (Q, \Sigma, q_0, A, \delta)$, let

$M_m = (Q_m, \Sigma, \{q_0\}, A_m, \delta_m)$,

where

$Q_m = \{ q \mid q \in Q \}$, $A_m = \{ q \mid q \in A \}$,

and $\delta_m : Q_m \times \Sigma \to Q_m$ is s.t. for all $q \in Q$ and $a \in \Sigma$,

$\delta_m([q_0], a) = [\delta(q, a)]$.

Then $L(M_m) = L(M)$, and no FA with fewer states than $M_m$ recognizes $L(M)$.

So for all $x \in \Sigma^*$, $\delta_m([q_0], x) = [\delta(q_0, x)]$.

Using this result, we can show that $L(M_m) = L(M)$.

$M_m$ accepts $x$ iff $\delta_m([q_0], x) \in A_m$ (def acceptance)

iff $[\delta(q_0, x)] \in A_m$ (previous result)

iff exist $q \in A$ s.t. $[q] = [\delta(q_0, x)]$ (def $A_m$)

iff $\delta(q_0, x) \in A$ (def =)

iff $M$ accepts $x$ (def acceptance)

So $M_m$ accepts $L(M)$, or we see that no FA with fewer states accepts $L(M)$, recall that by Theorem 5.1, any FA that accepts $L(M)$ must have at least as many states as there are equivalence classes of $L$. By the lemma we proved earlier, that is exactly how many states $M_m$ has.

The main difficulty in applying the construction in this theorem is the computation of the set of states in the minimal FA; that is, the set of equivalence classes of $m$.

The following lemma can be understood to suggest an algorithm for this.

Lemma 5.3 For any $p, q \in Q$, $p \neq q$ iff there is a $z \in \Sigma^*$ s.t. $\delta(p, z) \in A$ iff $\delta(q, z) \notin A$.

Two observations based on this lemma:

1. It follows from this lemma that, if $\delta(p, A) \in A$ iff $\delta(q, A) \notin A$, then $p \neq q$.

That is, if $p \in A$ iff $q \notin A$, then $p \neq q$.

2. Now consider any $p, q \in Q$ and $a \in \Sigma$ s.t. $\delta(p, a) \neq \delta(q, a)$. By the lemma, there is a $z \in \Sigma^*$ s.t. $\delta(p, z) \in A$ iff $\delta(q, z) \notin A$. It follows that $\delta(p, z) \in A$ iff $\delta(q, z) \notin A$.

Hence, by the lemma, $p \neq q$.

That is, if $\delta(p, a) \neq \delta(q, a)$, then $p \neq q$.

These two observations begin to explain the following recursive definition of the set $S = \{ [p, q] \mid p \neq q \}$. 

14

15
Let $S$ be the smallest subset of $Q \times Q$ s.t.

1. $\{(p, q) \in Q \times Q \mid p \in A \iff q \notin A \} \subseteq S$
2. For any $p, q \in Q$ and $a \in \Sigma$,
   
   if $((\delta(p, a), \delta(q, a)) \in S$, then $(p, q) \in S$.

One can show that $S = \{(p, q) \in Q \times Q \mid p \neq q\}$.

This recursive characterization is nice because it suggests an algorithm for computing $S$, from which one can obtain the set of equivalence classes of $\equiv$. (How?)

**Example**

Step 0: List all pairs $(p, q)$ of states such that exactly one of $p, q$ is accepting.

Step $n + 1$: For each pair $(p, q)$ of states not listed in a previous step, list $(p, q)$ if there is a symbol $a$ such that $(\delta(p, a), \delta(q, a))$ was listed in step $n$.

Stop after the first step that yields no additional pairs of states.

The listed pairs of states are those that can’t be merged (i.e. the elements of $S$).

---

The Pumping Lemma

By Kleene’s Theorem, every regular language is recognized by an FA.

Consider an arbitrary FA $M = (Q, \Sigma, q_0, A, \delta)$, with $|Q| = n$.

Roughly speaking, when $M$ reads a string of length $n$, $M$ must “visit” some state twice, since $M$ has only $n$ states.

Let’s express this precisely...

**Observation** Let $w$ be a string over $\Sigma$, with $|w| = n$. Consider any $q \in Q$, $w$ can be written in the form $xyz$ with $y \neq \Lambda$, s.t. for some state $q_x \in Q$,

- $\delta'(q_x, x) = q_y$.
- $\delta'(q_x, y) = q_x$.

Again roughly speaking, there is a “loop” from $q_x$ to $q_x$, and reading $y$ takes $M$ through that loop. Reading $y$ again would take $M$ through that loop again.

So, for any $m \in N$, $\delta'(q_x, y^m) = q_x$ and $\delta'(q_x, y^m) = q_x$.

Now, consider an arbitrary FA $M = (Q, \Sigma, q_0, A, \delta)$, with $|Q| = n$, and any string $w \in L(M)$ s.t. $|w| \geq n$.

Let $q_0$ be the element of $A$ s.t. $\delta'(q_0, w) = q_x$.

In light of our previous observations, $w$ can be written in the form $xyz$ with $y \neq \Lambda$, s.t. for some state $q_x \in Q$,

- $\delta'(q_x, x) = q_y$.
- $\delta'(q_x, y^m) = q_x$ for all $m \in N$.
- $\delta'(q_x, z) = q_x$.

It follows that for all $m \in N$, $xy^mz \in L(M)$.

Moreover, it is clear that $x$ and $y$ can be chosen so that $|xy| \leq n$.

Recall: By Kleene’s Theorem, every regular language is recognized by some FA (which of course has some number $n$ of states!).

These observations yield the following theorem...
Theorem 5.2 Let $L$ be a language recognized by an FA with exactly $n$ states. For any $w \in L$ s.t. $|w| \geq n$, there are strings $x, y, z$ s.t.

$$w = xyz$$

$$y \neq \Lambda$$

$$|y| \leq n$$

for all $m \in \mathbb{N}$, $xy^mz \in L$.

We can eliminate the reference to an FA that recognizes $L$.

Theorem 5.2a (Pumping Lemma) For any regular language $L$, there is an $n \in \mathbb{N}$ s.t. for any $w \in L$ with $|w| \geq n$ there are strings $x, y, z$ s.t.

$$w = xyz$$

$$y \neq \Lambda$$

$$|y| \leq n$$

for all $m \in \mathbb{N}$, $xy^mz \in L$.

This simple result can be used to show that a language is not regular...

---

Another Pumping Example

Consider $pal = \{x \in \{a,b\}^* \mid x = \text{rev}(x)\}$.

Suppose that $pal$ is regular.

Let $n$ be the natural number guaranteed by the Pumping Lemma. Take

$$w = a^nba^m$$

Clearly $w \in pal$ and $|w| \geq n$. So there are strings $x, y, z$ s.t. $w = xyz$, $y \neq \Lambda$, $|y| \leq n$, and for all $n \in \mathbb{N}$, $xy^nz \in pal$.

Since $|xy| \leq n$, $xy$ is a prefix of $a^n$. And since $y \neq \Lambda$, $y = a^m$, for some $m$ (0 $m \leq n$). Therefore,

$$xy^nz = a^{m}ba^n$$

with $n - m < n$. Consequently, $xy^nz \notin pal$. Contradiction.

We can conclude that $pal$ is not regular.

---

A Pumping Example

Consider $L = \{a^n b^n \mid n \in \mathbb{N}\}$.

Suppose that $L$ is regular. (We will derive a contradiction.)

Let $n$ be the natural number guaranteed by the Pumping Lemma. Take

$$w = a^n b^n$$

Notice that $w \in L$ and $|w| \geq n$. Hence, by the Pumping Lemma, there are strings $x, y, z$ s.t.

$$w = xyz$$

$$y \neq \Lambda$$

$$|y| \leq n$$

$$xy^mz \in L \text{ for all } m \in \mathbb{N}.$$ 

Since $|xy| \leq n$, we know that $xy = a^m$ for some $m \leq n$. And since $y \neq \Lambda$, $y = a^i$ for some $i$ (0 $i \leq n$). So $xy^i z = a^m b^n$, with $n - i < n$. Consequently, $xy^i z \notin L$. Contradiction.

So we can conclude that $L$ is not regular.

Show that $L = \{w \mid w \in \{a,b\}^* \}$ is not regular.

Suppose that $L$ is regular.

Let $n$ be the natural number guaranteed by the Pumping Lemma. Take

$$w = a^n b^n$$

Clearly $w \in L$ and $|w| \geq n$. So there are strings $x, y, z$ s.t. $w = xyz$, $y \neq \Lambda$, $|xy| \leq n$, and for all $n \in \mathbb{N}$, $xy^nz \in L$.

Since $|xy| \leq n$, $xy$ is a prefix of $a^n$. And since $y \neq \Lambda$, $y = a^m$, for some $m$ (0 $m \leq n$). Since

$$xy^i z = a^m b^n$$

with $n - m < n$, $xy^i z \notin L$. Contradiction.

We can conclude that $L$ is not regular.
Show that $L = \{a^p \mid p \text{ is prime} \}$ is not regular.

Suppose that $L$ is regular.

Let $n$ be the natural number guaranteed by the Pumping Lemma. Take

$$w = a^p \text{ for some } p \geq n \text{ s.t. } p \text{ is prime}.$$  

Clearly $w \in L$ and $|w| \geq n$. So there are strings $x, y, z$ s.t. $w = xyz$, $y \neq \Lambda$, $|xy| \leq n$, and for all $n \in N$, $xy^nz \in L$.

Let

$$w' = xy^{p+1}z.$$  

So

$$w' = a^{zp+1}.$$  

Since $p + ph = p(1 + [h])$, $p + ph = 0$ if and only if $p = 0$. But $y \neq \Lambda$, so $|y| \neq 0$. Hence $w' \notin L$.

On the other hand, for all $n \in N$, $xy^n z \in L$. Hence, $w' = xy^{p+1}z \in L$. Contradiction.

So $L$ is not regular.

An example of such a language is

$$L = \{a^p b^q c^r \mid 1, j, k \geq 0 \}.$$  

1. There is an $n \in N$ s.t. for any $w \in L$ with $|w| \geq n$ there are strings $x, y, z$ s.t.

$$w = xyz$$

$$y \neq \Lambda$$

$$|xy| \leq n$$

for all $m \in N$, $xy^nz \in L$.

In fact, you can take $n = 1$. So consider any nonempty string $w \in L$. There are two cases.

Case 1: $w \in \{a^p b^q c^r \mid 1, j, k \geq 0 \}$.

Take $x = \Lambda$, $y = a$, $z = a^{m-1} b^l c^l$. So $w = xyz$, $y \neq \Lambda$, and $|xy| \leq n$.

Moreover, for any $m \in N$, $xy^nz$ has the form $a^p b^q c^r$ for some $l \in N$.

All such strings belong to $L$ (even when $l = 0$).

Case 2: $w \in \{b^k a^l \mid k, l \geq 0 \}$.

Again take $x = \Lambda$, and let $y$ consist of the first symbol of $w$, with $z$ chosen so that $w = yz$. So $w = xyz$, $y \neq \Lambda$, and $|xy| \leq n$.

Moreover, for any $m \in N$, $xy^nz \in L$.

The Pumping Lemma tells us that any regular language can be “pumped”: that is, there is a natural number $n$ s.t. for any $w \in L$ with $|w| \geq n$ there are strings $x, y, z$ s.t.

$$w = xyz$$

$$y \neq \Lambda$$

$$|xy| \leq n$$

for all $m \in N$, $xy^nz \in L$.

So if we can show that a language cannot be pumped in this fashion, we can conclude that it is not regular.

The converse does not hold: that is, some non-regular languages can be pumped...

\[ L = \{a^{\beta \delta^j} \mid 1, j \geq 0 \} \cup \{b^k \delta^l \mid k, l \geq 0 \} \]

2. On the other hand, $L$ is not regular.

To prove this, we need to recall Corollary 5.1.

**Corollary 5.1** $L$ is regular if the set of equivalence classes of $I_L$ is finite.

So what we need to show is that the set of equivalence classes of $I_L$ is not finite. That is, we need to show that there are infinitely many strings in $L$ that are distinguishable from one another wrt $L$.

To see this, notice that for any two strings $ab^m$, $ab^n$ with $m \neq n$, $ab^m c^r \notin L$ while $ab^n c^r \in L$. 

$L = \{a^{\beta \delta^j} \mid 1, j \geq 0 \} \cup \{b^k \delta^l \mid k, l \geq 0 \}$.
Regular languages and programming languages

Programming languages are not regular.

Example The set of C programs is not regular.

Proof. Again we can use Corollary 5.1. So we need to show that there are infinitely many C programs that are distinguishable with the set of all C programs. Consider the strings of the form

\[
\text{main } \Theta \{m \leq \} \quad (m \in \mathbb{N})
\]

Each is distinguishable from all the others, since

\[
\text{main } \Theta \{m \leq n \} \quad (m, n \in \mathbb{N})
\]

is a C program iff \( m = n \).

But each of the various kinds of tokens available in C — identifiers, literals, operators, reserved words, and punctuation — can be described by a regular expression.

The first phase in compiling a C program is called lexical analysis — in this phase the string of symbols is essentially converted into a string of tokens.

A program for doing this can be generated by a lexical analyzer generator, such as lex, which takes as input a set of regular expressions describing the structure of tokens.

The next phase in compiling a C program is parsing — roughly, extracting the syntactic structure of the string of tokens. This syntactic structure is described by a "context-free grammar" — we'll study these next.

A parser of this kind can be generated by a parser generator, such as yacc, which takes as input a context-free grammar.