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

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

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, $L(a) = \{a\}$;
- $\boldsymbol{\epsilon}$, matched only by ϵ, the null string; in symbols, $L(\boldsymbol{\epsilon}) = \{\epsilon\}$;
- $\boldsymbol{\emptyset}$, matched by nothing; in symbols, $L(\boldsymbol{\emptyset}) = \emptyset$, 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:

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

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

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

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

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

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.,

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

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:

  $$\boldsymbol{\epsilon} + @b.$$ □

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 $\emptyset$. The objects $\boldsymbol{\epsilon}$ and $\boldsymbol{\emptyset}$ are *symbols* in the language of patterns, whereas ϵ and $\emptyset$ 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 $\emptyset$ 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 $\emptyset$ 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 $x \in a^*b^*$.

Lecture 8

Pattern Matching and Regular Expressions

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 $\boldsymbol{\epsilon}$ since it is equivalent to $\sim(\#@)$ and also to $\boldsymbol{\emptyset}^*$. 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

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

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

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

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 $a \in \Sigma$, $\boldsymbol{\epsilon}$, $\boldsymbol{\emptyset}$, and operators $+$, $\cdot$, and *. Patterns using only these symbols are called *regular expressions*. Actually, as we have observed, even $\boldsymbol{\epsilon}$ 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,

$$\begin{aligned} L(\alpha + (\beta + \gamma)) &= L((\alpha + \beta) + \gamma), \\ L(\alpha(\beta\gamma)) &= L((\alpha\beta)\gamma), \end{aligned}$$

we can write

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

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

$$\alpha + \beta\gamma$$

could be interpreted as either

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

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 $a \in \Sigma$, $\boldsymbol{\epsilon}$, $\boldsymbol{\emptyset}$, 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 $\varnothing$ 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) $\boldsymbol{\epsilon}$;
(iii) $\boldsymbol{\emptyset}$;
(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 $a \in \Sigma$, $L(\boldsymbol{\epsilon}) = \{\epsilon\}$, and $L(\boldsymbol{\emptyset}) = \varnothing$, 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. □

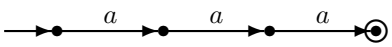
Example 8.2 Let's convert the regular expression

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

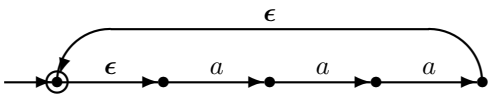
for the set

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

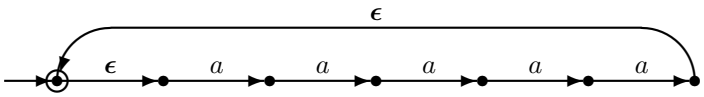
to an equivalent NFA. First we show how to construct an automaton for $(aaa)^*$. We take an automaton accepting only the string aaa, say



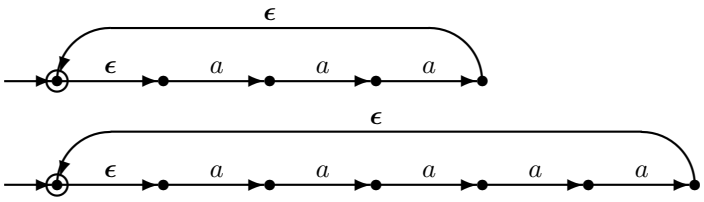
Applying the construction of Lecture 6, we add a new start state and ϵ-transitions from the new start state to all the old start states and from all the old accept states to the new start state. We let the new start state be the only accept state of the new automaton. This gives



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.

$$\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma \tag{9.1}$$
$$\alpha + \beta \equiv \beta + \alpha \tag{9.2}$$
$$\alpha + \boldsymbol{\emptyset} \equiv \alpha \tag{9.3}$$
$$\alpha + \alpha \equiv \alpha \tag{9.4}$$
$$\alpha(\beta\gamma) \equiv (\alpha\beta)\gamma \tag{9.5}$$
$$\boldsymbol{\epsilon}\alpha \equiv \alpha\boldsymbol{\epsilon} \equiv \alpha \tag{9.6}$$
$$\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma \tag{9.7}$$
$$(\alpha + \beta)\gamma \equiv \alpha\gamma + \beta\gamma \tag{9.8}$$
$$\boldsymbol{\emptyset}\alpha \equiv \alpha\boldsymbol{\emptyset} \equiv \boldsymbol{\emptyset} \tag{9.9}$$
$$\boldsymbol{\epsilon} + \alpha\alpha^* \equiv \alpha^* \tag{9.10}$$
$$\boldsymbol{\epsilon} + \alpha^*\alpha \equiv \alpha^* \tag{9.11}$$
$$\beta + \alpha\gamma \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma \tag{9.12}$$
$$\beta + \gamma\alpha \leq \gamma \Rightarrow \beta\alpha^* \leq \gamma \tag{9.13}$$

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.

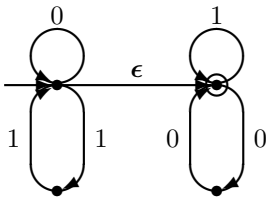
Here are some useful equations that follow from (9.1) through (9.13) that you can use to simplify expressions.

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

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



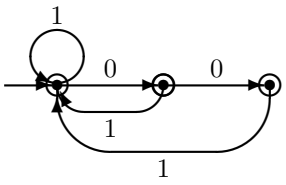
This expression is simple enough that the easiest thing to do is eyeball it. The mechanical method described in Lecture 8 would give more states and ϵ-transitions than shown here. The two states connected by an ϵ-transition cannot be collapsed into one state, since then 10 would be accepted, which does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

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

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

$$\begin{aligned}(1+01+001)^*(\epsilon+0+00) &\equiv ((\epsilon+0+00)1)^*(\epsilon+0+00)\\ &\equiv ((\epsilon+0)(\epsilon+0)1)^*(\epsilon+0)(\epsilon+0).\end{aligned}$$

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

$$\alpha_{uv}^X$$

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

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

and for $u = v$, take

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

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

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

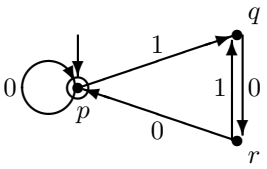
The sum of all expressions of the form

$$\alpha_{sf}^{Q},$$

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

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

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

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

By similar informal reasoning, we can take

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

Thus we can take

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

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):

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

□

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].

Supplementary Lecture A

Kleene Algebra and Regular Expressions

In Lecture 9, we gave a combinatorial proof that every finite automaton has an equivalent regular expression. Here is an algebraic proof that generalizes that argument. It is worth looking at because it introduces the notion of *Kleene algebra* and the use of matrices. We will show how to use matrices and Kleene algebra to solve systems of linear equations involving sets of strings.

Kleene algebra is named after Stephen C. Kleene, who invented the regular sets [70].

Kleene Algebra

We have already observed in Lecture 9 that the set operations $\cup$, $\cdot$, and * on subsets of Σ^*, along with the distinguished subsets $\varnothing$ and $\{\epsilon\}$, satisfy certain important algebraic properties. These were listed in Lecture 9, axioms (9.1) through (9.13). Let us call any algebraic structure satisfying these properties a *Kleene algebra*. In general, a Kleene algebra $\mathcal{K}$ consists of a nonempty set with two distinguished constants 0 and 1, two binary operations $+$ and $\cdot$ (usually omitted in expressions), and a unary operation

* satisfying the following axioms.

$$
\begin{array}{rcll}
a+(b+c) & = & (a+b)+c & \text{associativity of } + & \text{(A.1)}\\
a+b & = & b+a & \text{commutativity of } + & \text{(A.2)}\\
a+a & = & a & \text{idempotence of } + & \text{(A.3)}\\
a+0 & = & a & 0 \text{ is an identity for } + & \text{(A.4)}\\
a(bc) & = & (ab)c & \text{associativity of } \cdot & \text{(A.5)}\\
a1 & = & 1a=a & 1 \text{ is an identity for } \cdot & \text{(A.6)}\\
a0 & = & 0a=0 & 0 \text{ is an annihilator for } \cdot & \text{(A.7)}\\
a(b+c) & = & ab+ac & \text{distributivity} & \text{(A.8)}\\
(a+b)c & = & ac+bc & \text{distributivity} & \text{(A.9)}\\
1+aa^* & = & a^* & & \text{(A.10)}\\
1+a^*a & = & a^* & & \text{(A.11)}\\
b+ac\le c & \Rightarrow & a^*b\le c & & \text{(A.12)}\\
b+ca\le c & \Rightarrow & ba^*\le c & & \text{(A.13)}
\end{array}
$$

In (A.12) and (A.13), $\le$ refers to the naturally defined order

$$a\le b \quad\stackrel{\text{def}}{\Longleftrightarrow}\quad a+b=b.$$

In 2^{Σ^*}, $\le$ is just set inclusion $\subseteq$.

Axioms (A.1) through (A.9) discuss the properties of addition and multiplication in a Kleene algebra. These properties are the same as those of ordinary addition and multiplication, with the addition of the idempotence axiom (A.3). These axioms can be summed up briefly by saying that $\mathcal{K}$ is an *idempotent semiring*. The remaining axioms (A.10) through (A.13) discuss the properties of the operator *. They say essentially that * behaves like the asterate operator on sets of strings or the reflexive transitive closure operator on binary relations.

It follows quite easily from the axioms that $\le$ is a partial order; that is, it is reflexive ($a\le a$), transitive ($a\le b$ and $b\le c$ imply $a\le c$), and antisymmetric ($a\le b$ and $b\le a$ imply $a=b$). Moreover, $a+b$ is the least upper bound of a and b with respect to $\le$. All the operators are monotone with respect to $\le$; in other words, if $a\le b$, then $ac\le bc$, $ca\le cb$, $a+c\le b+c$, and $a^*\le b^*$.

By (A.10) and distributivity, we have

$$b+aa^*b \quad\le\quad a^*b,$$

which says that a^*b satisfies the inequality $b+ac\le c$ when substituted for c. The implication (A.12) says that a^*b is the $\le$-least element of $\mathcal{K}$ for which this is true. It follows that

Lemma A.1 *In any Kleene algebra, a^*b is the $\le$-least solution of the equation $x=ax+b$.*

Proof. Miscellaneous Exercise 21. □

Instead of (A.12) and (A.13), we might take the equivalent axioms

$$ac \leq c \quad \Rightarrow \quad a^*c \leq c, \tag{A.14}$$

$$ca \leq c \quad \Rightarrow \quad ca^* \leq c \tag{A.15}$$

(see Miscellaneous Exercise 22).

Here are some typical theorems of Kleene algebra. These can be derived by purely equational reasoning from the axioms above (Miscellaneous Exercise 20).

$$\begin{aligned} a^*a^* &= a^* \\ a^{**} &= a^* \\ (a^*b)^*a^* &= (a+b)^* && \text{denesting rule} && (A.16) \\ a(ba)^* &= (ab)^*a && \text{shifting rule} && (A.17) \\ a^* &= (aa)^* + a(aa)^* \end{aligned}$$

Equations (A.16) and (A.17), the *denesting rule* and the *shifting rule*, respectively, turn out to be particularly useful in simplifying regular expressions.

The family 2^{Σ^*} of all subsets of Σ^* with constants $\varnothing$ and $\{\epsilon\}$ and operations $\cup, \cdot$, and * forms a Kleene algebra, as does the family of all regular subsets of Σ^* with the same operations. As mentioned in Lecture 9, it can be shown that an equation $\alpha = \beta$ is a theorem of Kleene algebra, that is, is derivable from axioms (A.1) through (A.13), if and only if α and β are equivalent as regular expressions [73].

Another example of a Kleene algebra is the family of all binary relations on a set X with the empty relation for 0, the identity relation

$$\iota \stackrel{\text{def}}{=} \{(u,u) \mid u \in X\}$$

for 1, $\cup$ for $+$, relational composition

$$R \circ S \stackrel{\text{def}}{=} \{(u,w) \mid \exists v \in X \ (u,v) \in R \text{ and } (v,w) \in S\}$$

for $\cdot$, and reflexive transitive closure for *:

$$R^* \stackrel{\text{def}}{=} \bigcup_{n \geq 0} R^n,$$

where

$$\begin{aligned} R^0 &\stackrel{\text{def}}{=} \iota, \\ R^{n+1} &\stackrel{\text{def}}{=} R^n \circ R. \end{aligned}$$

Still another example is the family of $n \times n$ Boolean matrices with the zero matrix for 0, the identity matrix for 1, componentwise Boolean matrix

addition and multiplication for $+$ and $\cdot$, respectively, and reflexive transitive closure for *. This is really the same as the previous example, where the set X has n elements.

Matrices

Given an arbitrary Kleene algebra $\mathcal{K}$, the set of $n \times n$ matrices over $\mathcal{K}$, which we will denote by $\mathcal{M}(n, \mathcal{K})$, also forms a Kleene algebra. In $\mathcal{M}(2, \mathcal{K})$, for example, the identity elements for $+$ and $\cdot$ are

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

respectively, and the operations $+$, $\cdot$, and * are given by

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix},$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}, \quad \text{and}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* \stackrel{\text{def}}{=} \begin{bmatrix} (a+bd^*c)^* & (a+bd^*c)^*bd^* \\ (d+ca^*b)^*ca^* & (d+ca^*b)^* \end{bmatrix}, \tag{A.18}$$

respectively. In general, $+$ and $\cdot$ in $\mathcal{M}(n, \mathcal{K})$ are ordinary matrix addition and multiplication, respectively, the identity for $+$ is the zero matrix, and the identity for $\cdot$ is the identity matrix.

To define E^* for a given $n \times n$ matrix E over $\mathcal{K}$, we proceed by induction on n. If $n = 1$, the structure $\mathcal{M}(n, \mathcal{K})$ is just $\mathcal{K}$, so we are done. For $n > 1$, break E up into four submatrices

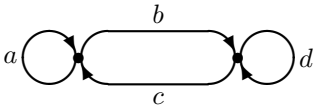
$$E \;=\; \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

such that A and D are square, say $m \times m$ and $(n-m) \times (n-m)$, respectively. By the induction hypothesis, $\mathcal{M}(m, \mathcal{K})$ and $\mathcal{M}(n-m, \mathcal{K})$ are Kleene algebras, so it makes sense to form the asterates of any $m \times m$ or $(n-m) \times (n-m)$ matrix over $\mathcal{K}$, and these matrices will satisfy all the axioms for *. This allows us to define

$$E^* \;\stackrel{\text{def}}{=}\; \left[\begin{array}{c|c} (A+BD^*C)^* & (A+BD^*C)^*BD^* \\ \hline (D+CA^*B)^*CA^* & (D+CA^*B)^* \end{array}\right]. \tag{A.19}$$

Compare this definition to (A.18).

The expressions on the right-hand sides of (A.18) and (A.19) may look like they were pulled out of thin air. Where did we get them from? The answer will come to you if you stare really hard at the following mandala:



It can be shown that $\mathcal{M}(n, \mathcal{K})$ is a Kleene algebra under these definitions:

Lemma A.2 *If $\mathcal{K}$ is a Kleene algebra, then so is $\mathcal{M}(n, \mathcal{K})$.*

Proof. Miscellaneous Exercise 24. We must verify that $\mathcal{M}(n, \mathcal{K})$ satisfies the axioms (A.1) through (A.13) of Kleene algebra assuming only that $\mathcal{K}$ does. □

If E is a matrix of indeterminates, and if the inductive construction of E^* given in (A.19) is carried out *symbolically*, then the entries of the resulting matrix E^* will be regular expressions in those indeterminates. This construction generalizes the construction of Lecture 9, which corresponds to the case $m = 1$.

Systems of Linear Equations

It is possible to solve systems of linear equations over a Kleene algebra $\mathcal{K}$. Suppose we are given a set of n variables $x_1, \ldots, x_n$ ranging over $\mathcal{K}$ and a system of n equations of the form

$$x_i \quad = \quad a_{i1}x_1 + \cdots + a_{in}x_n + b_i, \quad 1 \leq i \leq n,$$

where the a_{ij} and b_i are elements of $\mathcal{K}$. Arranging the a_{ij} in an $n \times n$ matrix A, the b_i in a vector b of length n, and the x_i in a vector x of length n, we obtain the matrix-vector equation

$$x \quad = \quad Ax + b. \tag{A.20}$$

It is now not hard to show

Theorem A.3 *The vector A^*b is a solution to (A.20); moreover, it is the $\leq$-least solution in $\mathcal{K}^n$.*

Proof. Miscellaneous Exercise 25. □

Now we use this to give a regular expression equivalent to an arbitrarily given deterministic finite automaton

$$M \quad = \quad (Q, \Sigma, \delta, s, F).$$

Assume without loss of generality that $Q = \{1, 2, \ldots, n\}$. For each $q \in Q$, let X_q denote the set of strings in Σ^* that would be accepted by M if q were the start state; that is,

$$X_q \quad \stackrel{\text{def}}{=} \quad \{x \in \Sigma^* \mid \widehat{\delta}(q, x) \in F\}.$$

The X_q satisfy the following system of equations:

$$X_q \quad = \quad \begin{cases} \sum_{a\in\Sigma} aX_{\delta(q,a)} & \text{if } q \notin F, \\ \sum_{a\in\Sigma} aX_{\delta(q,a)} + 1 & \text{if } q \in F. \end{cases}$$

Moreover, the X_q give the least solution with respect to $\subseteq$. As above, these equations can be arranged in a single matrix-vector equation of the form

$$X \quad = \quad AX + b, \tag{A.21}$$

where A is an $n \times n$ matrix containing sums of elements of Σ, b is a 0-1 vector of length n, and X is a vector consisting of $X_1, \ldots, X_n$. The vector X is the least solution of (A.21). By Theorem A.3,

$$X \quad = \quad A^*b.$$

Compute the matrix A^* symbolically according to (A.19), so that its entries are regular expressions, then multiply by b. A regular expression for $L(M)$ can then be read off from the sth entry of A^*b, where s is the start state of M.

Historical Notes

Salomaa [108] gave the first complete axiomatization of the algebra of regular sets. The algebraic theory was developed extensively in the monograph of Conway [27]. Many others have contributed to the theory, including Redko [103], Backhouse [6], Bloom and Ésik [10], Boffa [11, 12], Gécseg and Peák [41], Krob [74], Kuich and Salomaa [76], and Salomaa and Soittola [109]. The definition of Kleene algebra and the complete axiomatization given here is from Kozen [73].

Lecture 10

Homomorphisms

A *homomorphism* is a map $h : \Sigma^* \to \Gamma^*$ such that for all $x, y \in \Sigma^*$,

$$h(xy) = h(x)h(y), \tag{10.1}$$
$$h(\epsilon) = \epsilon. \tag{10.2}$$

Actually, (10.2) is a consequence of (10.1):

$$\begin{aligned} |h(\epsilon)| &= |h(\epsilon\epsilon)| \\ &= |h(\epsilon)h(\epsilon)| \\ &= |h(\epsilon)| + |h(\epsilon)|; \end{aligned}$$

subtracting $|h(\epsilon)|$ from both sides, we have $|h(\epsilon)| = 0$, therefore $h(\epsilon) = \epsilon$.

It follows from these properties that any homomorphism defined on Σ^* is uniquely determined by its values on Σ. For example, if $h(a) = ccc$ and $h(b) = dd$, then

$$h(abaab) = h(a)h(b)h(a)h(a)h(b) = cccddccccccdd.$$

Moreover, any map $h : \Sigma \to \Gamma^*$ extends uniquely by induction to a homomorphism defined on all of Σ^*. Therefore, in order to specify a homomorphism completely, we need only say what values it takes on elements of Σ.

If $A \subseteq \Sigma^*$, define

$$h(A) \stackrel{\text{def}}{=} \{h(x) \mid x \in A\} \subseteq \Gamma^*,$$

and if $B \subseteq \Gamma^*$, define

$$h^{-1}(B) \stackrel{\text{def}}{=} \{x \mid h(x) \in B\} \subseteq \Sigma^*.$$

The set $h(A)$ is called the *image* of A under h, and the set $h^{-1}(B)$ is called the *preimage* of B under h.

We will show two useful closure properties of the regular sets: any homomorphic image or homomorphic preimage of a regular set is regular.

Theorem 10.1 *Let $h : \Sigma^* \to \Gamma^*$ be a homomorphism. If $B \subseteq \Gamma^*$ is regular, then so is its preimage $h^{-1}(B)$ under h.*

Proof. Let $M = (Q, \Gamma, \delta, s, F)$ be a DFA such that $L(M) = B$. Create a new DFA $M' = (Q, \Sigma, \delta', s, F)$ for $h^{-1}(B)$ as follows. The set of states, start state, and final states of M' are the same as in M. The input alphabet is Σ instead of Γ. The transition function δ' is defined by

$$\delta'(q, a) \stackrel{\text{def}}{=} \widehat{\delta}(q, h(a)).$$

Note that we have to use $\widehat{\delta}$ on the right-hand side, since $h(a)$ need not be a single letter.

Now it follows by induction on $|x|$ that for all $x \in \Sigma^*$,

$$\widehat{\delta'}(q, x) = \widehat{\delta}(q, h(x)). \tag{10.3}$$

For the basis $x = \epsilon$, using (10.1),

$$\widehat{\delta'}(q, \epsilon) = q = \widehat{\delta}(q, \epsilon) = \widehat{\delta}(q, h(\epsilon)).$$

For the induction step, assume that $\widehat{\delta'}(q, x) = \widehat{\delta}(q, h(x))$. Then

$$\begin{aligned}
\widehat{\delta'}(q, xa) &= \delta'(\widehat{\delta'}(q, x), a) && \text{definition of } \widehat{\delta'} \\
&= \delta'(\widehat{\delta}(q, h(x)), a) && \text{induction hypothesis} \\
&= \widehat{\delta}(\widehat{\delta}(q, h(x)), h(a)) && \text{definition of } \delta' \\
&= \widehat{\delta}(q, h(x)h(a)) && \text{Homework 1, Exercise 3} \\
&= \widehat{\delta}(q, h(xa)) && \text{property (10.2) of homomorphisms.}
\end{aligned}$$

Now we can use (10.3) to prove that $L(M') = h^{-1}(L(M))$. For any $x \in \Sigma^*$,

$$\begin{aligned}
x \in L(M') &\iff \widehat{\delta'}(s, x) \in F && \text{definition of acceptance} \\
&\iff \widehat{\delta}(s, h(x)) \in F && \text{by (10.3)} \\
&\iff h(x) \in L(M) && \text{definition of acceptance} \\
&\iff x \in h^{-1}(L(M)) && \text{definition of } h^{-1}(L(M)).
\end{aligned}$$

□

Theorem 10.2 *Let $h : \Sigma^* \to \Gamma^*$ be a homomorphism. If $A \subseteq \Sigma^*$ is regular, then so is its image $h(A)$ under h.*

Proof. For this proof, we will use regular expressions. Let α be a regular expression over Σ such that $L(\alpha) = A$. Let α' be the regular expression obtained by replacing each letter $a \in \Sigma$ appearing in α with the string $h(a) \in \Gamma^*$. For example, if $h(a) = ccc$ and $h(b) = dd$, then

$$((a+b)^*ab)' \quad = \quad (ccc+dd)^*cccdd.$$

Formally, α' is defined by induction:

$$\begin{aligned} a' &= h(a), \quad a \in \Sigma, \\ \boldsymbol{\emptyset}' &= \boldsymbol{\emptyset}, \\ (\beta+\gamma)' &= \beta' + \gamma', \\ (\beta\gamma)' &= \beta'\gamma', \\ \beta^{*\prime} &= \beta'^*. \end{aligned}$$

We claim that for any regular expression β over Σ,

$$L(\beta') \quad = \quad h(L(\beta)); \tag{10.4}$$

in particular, $L(\alpha') = h(A)$. This can be proved by induction on the structure of β. To do this, we will need two facts about homomorphisms: for any pair of subsets $C, D \subseteq \Sigma^*$ and any family of subsets $C_i \subseteq \Sigma^*$, $i \in I$,

$$h(CD) \quad = \quad h(C)h(D), \tag{10.5}$$

$$h(\bigcup_{i\in I} C_i) \quad = \quad \bigcup_{i\in I} h(C_i). \tag{10.6}$$

To prove (10.5),

$$\begin{aligned} h(CD) &= \{h(w) \mid w \in CD\} \\ &= \{h(yz) \mid y \in C,\ z \in D\} \\ &= \{h(y)h(z) \mid y \in C,\ z \in D\} \\ &= \{uv \mid u \in h(C),\ v \in h(D)\} \\ &= h(C)h(D). \end{aligned}$$

To prove (10.6),

$$\begin{aligned} h(\bigcup_i C_i) &= \{h(w) \mid w \in \bigcup_i C_i\} \\ &= \{h(w) \mid \exists i\ w \in C_i\} \\ &= \bigcup_i \{h(w) \mid w \in C_i\} \\ &= \bigcup_i h(C_i). \end{aligned}$$

Now we prove (10.4) by induction. There are two base cases:

$$L(a') \quad = \quad L(h(a)) = \{h(a)\} = h(\{a\}) = h(L(a))$$

and

$$L(\boldsymbol{\emptyset}') \quad = \quad L(\boldsymbol{\emptyset}) = \varnothing = h(\varnothing) = h(L(\boldsymbol{\emptyset})).$$

The case of $\boldsymbol{\epsilon}$ is covered by the other cases, since $\epsilon = \varnothing^*$.
There are three induction cases, one for each of the operators $+$, $\cdot$, and *. For $+$,

$$\begin{array}{rcll} L((\beta+\gamma)') & = & L(\beta'+\gamma') & \text{definition of } ' \\ & = & L(\beta') \cup L(\gamma') & \text{definition of } + \\ & = & h(L(\beta)) \cup h(L(\gamma)) & \text{induction hypothesis} \\ & = & h(L(\beta) \cup L(\gamma)) & \text{property (10.6)} \\ & = & h(L(\beta+\gamma)) & \text{definition of } +. \end{array}$$

The proof for $\cdot$ is similar, using property (10.5) instead of (10.6). Finally, for *,

$$\begin{array}{rcll} L(\beta^{*\prime}) & & & \\ & = & L(\beta'^*) & \text{definition of } ' \\ & = & L(\beta')^* & \text{definition of regular expression operator } ^* \\ & = & h(L(\beta))^* & \text{induction hypothesis} \\ & = & \bigcup_{n\geq 0} h(L(\beta))^n & \text{definition of set operator } ^* \\ & = & \bigcup_{n\geq 0} h(L(\beta)^n) & \text{property (10.5)} \\ & = & h(\bigcup_{n\geq 0} L(\beta)^n) & \text{property (10.6)} \\ & = & h(L(\beta)^*) & \text{definition of set operator } ^* \\ & = & h(L(\beta^*)) & \text{definition of regular expression operator } ^*. \quad \square \end{array}$$

Warning: It is *not* true that A is regular whenever $h(A)$ is. This is not what Theorem 10.1 says. We will show later that the set $\{a^n b^n \mid n \geq 0\}$ is not regular, but the image of this set under the homomorphism $h(a) = h(b) = a$ is the regular set $\{a^n \mid n \text{ is even}\}$. The preimage $h^{-1}(\{a^n \mid n \text{ is even}\})$ is not $\{a^n b^n \mid n \geq 0\}$, but $\{x \in \{a,b\}^* \mid |x| \text{ is even}\}$, which is regular.

Automata with ϵ-transitions

Here is an example of how to use homomorphisms to give a clean treatment of ϵ-transitions. Define an *NFA with ϵ-transitions* to be a structure

$$M \quad = \quad (Q,\, \Sigma,\, \boldsymbol{\epsilon},\, \Delta,\, S,\, F)$$

such that $\boldsymbol{\epsilon}$ is a special symbol not in Σ and

$$M_{\boldsymbol{\epsilon}} \quad = \quad (Q,\, \Sigma \cup \{\boldsymbol{\epsilon}\},\, \Delta,\, S,\, F)$$

is an ordinary NFA over the alphabet $\Sigma \cup \{\boldsymbol{\epsilon}\}$. We define acceptance for automata with ϵ-transitions as follows: for any $x \in \Sigma^*$, M *accepts* x if there exists $y \in (\Sigma \cup \{\boldsymbol{\epsilon}\})^*$ such that

- $M_{\boldsymbol{\epsilon}}$ accepts y under the ordinary definition of acceptance for NFAs, and
- x is obtained from y by erasing all occurrences of the symbol $\boldsymbol{\epsilon}$; that is, $x = h(y)$, where

  $$h : (\Sigma \cup \{\boldsymbol{\epsilon}\})^* \quad \to \quad \Sigma^*$$

  is the homomorphism defined by

  $$\begin{aligned} h(a) \quad &\stackrel{\text{def}}{=} \quad a, \quad a \in \Sigma, \\ h(\boldsymbol{\epsilon}) \quad &\stackrel{\text{def}}{=} \quad \epsilon. \end{aligned}$$

In other words,

$$L(M) \quad \stackrel{\text{def}}{=} \quad h(L(M_{\boldsymbol{\epsilon}})).$$

This definition and the definition involving ϵ-closure described in Lecture 6 are equivalent (Miscellaneous Exercise 10). It is immediate from this definition and Theorem 10.2 that the set accepted by any finite automaton with ϵ-transitions is regular.

Hamming Distance

Here is another example of the use of homomorphisms. We can use them to give slick solutions to Exercise 3 of Homework 2 and Miscellaneous Exercise 8, the problems involving Hamming distance. Let $\Sigma = \{0, 1\}$ and consider the alphabet

$$\Sigma \times \Sigma \quad = \quad \left\{ \quad , \quad , \quad , \quad \right\}. \qquad \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array} \; \begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline \end{array} \; \begin{array}{|c|}\hline 1 \\ \hline 0 \\ \hline \end{array} \; \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline \end{array}$$

The elements of $\Sigma \times \Sigma$ are ordered pairs, but we write the components one on top of the other. Let $\mathbf{top} : \Sigma \times \Sigma \to \Sigma$ and $\mathbf{bottom} : \Sigma \times \Sigma \to \Sigma$ be the two projections

$$\mathbf{top}\left(\begin{array}{|c|}\hline \\ \hline \\ \hline \end{array}\right) = a, \qquad \begin{array}{|c|}\hline a \\ \hline b \\ \hline \end{array}$$

$$\mathbf{bottom}\left(\begin{array}{|c|}\hline \\ \hline \\ \hline \end{array}\right) = b. \qquad \begin{array}{|c|}\hline a \\ \hline b \\ \hline \end{array}$$

These maps extend uniquely to homomorphisms $(\Sigma \times \Sigma)^* \to \Sigma^*$, which we also denote by **top** and **bottom**. For example,

$$\mathbf{top}\left(\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & & & \\ \hline \end{array}\right) = 0010, \qquad \begin{array}{|c|c|c|c|}\hline 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ \hline \end{array}$$

$$\mathbf{bottom}\left(\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & & & \\ \hline \end{array}\right) = 0111. \qquad \begin{array}{|c|c|c|c|}\hline 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ \hline \end{array}$$

Thus we can think of strings in $(\Sigma \times \Sigma)^*$ as consisting of two tracks, and the homomorphisms **top** and **bottom** give the contents of the top and bottom track, respectively.

For fixed k, let D_k be the set of all strings in $(\Sigma \times \Sigma)^*$ containing no more than k occurrences of

$$\begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline \end{array} \quad \text{or} \quad \begin{array}{|c|}\hline 1 \\ \hline 0 \\ \hline \end{array}.$$

This is certainly a regular set. Note also that

$$D_k = \{x \in (\Sigma \times \Sigma)^* \mid H(\mathbf{top}(x), \mathbf{bottom}(x)) \leq k\},$$

where H is the Hamming distance function. Now take any regular set $A \subseteq \Sigma^*$, and consider the set

$$\mathbf{top}(\mathbf{bottom}^{-1}(A) \cap D_k). \tag{10.7}$$

Believe it or not, this set is exactly $N_k(A)$, the set of strings in Σ^* of Hamming distance at most k from some string in A. The set $\mathbf{bottom}^{-1}(A)$ is the set of strings whose bottom track is in A; the set $\mathbf{bottom}^{-1}(A) \cap D_k$ is the set of strings whose bottom track is in A and whose top track is of Hamming distance at most k from the bottom track; and the set (10.7) is the set of top tracks of all such strings.

Moreover, the set (10.7) is a regular set, because the regular sets are closed under intersection, homomorphic image, and homomorphic preimage.

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

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

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

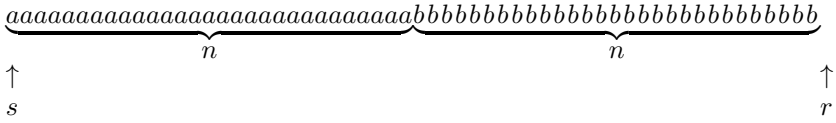
Intuitively, in order to accept the set B, an automaton scanning a string of the form a^*b^* would have to remember when passing the center point between the a's and b's how many a's it has seen, since it would have to compare that with the number of b's and accept iff the two numbers are the same.

$$aaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbb$$
$$\uparrow$$
$$q$$

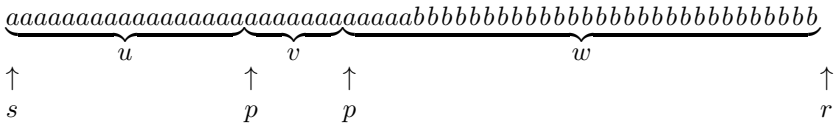
Moreover, it would have to do this for arbitrarily long strings of a's and b's, much longer than the number of states. This is an unbounded amount of information, and there is no way it can remember this with only finite memory. All it "knows" at that point is represented in the state q it is in, which 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:

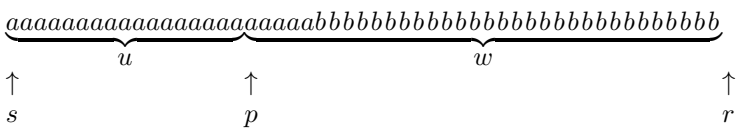


Let $j = |v| > 0$. In this example, $j = 7$. Then

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

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

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



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

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

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

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

$$
\begin{array}{l}
\underbrace{aaaaaaaaaaaaaaaaa}_{i}\underbrace{aaaaaaa}_{j}\underbrace{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}_{m} \\
\uparrow \qquad\qquad\qquad\qquad\quad \uparrow \qquad\quad \uparrow \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \uparrow \\
s \qquad\qquad\qquad\qquad\quad\; p \qquad\quad\; p \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\; r
\end{array}
$$

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:

$$\begin{aligned} 2^n + j &\leq 2^n + n \\ &< 2^n + 2^n \\ &= 2^{n+1} \end{aligned}$$

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.*

($\neg$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 ($\neg$P) holds of A. Because of the alternating "for all/there exists" form of ($\neg$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 ($\neg$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 ($\neg$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].

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

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

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:

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

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

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

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

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

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

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

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

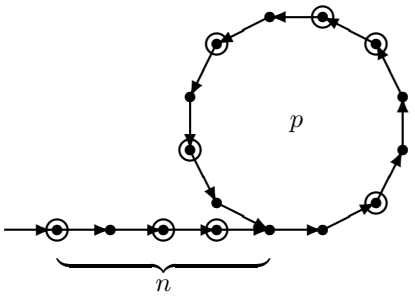
$$\{0, 3, 7, 11, 19, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, \ldots\}.$$

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:

Theorem 12.3 *Let $A \subseteq \{a\}^*$. Then A is regular if and only if the set $\{m \mid a^m \in A\}$, the set of lengths of strings in A, is ultimately periodic.*

Proof. If A is regular, then any DFA for it consists of a finite tail of some length, say $n \geq 0$, followed by a loop of length $p > 0$ (plus possibly some inaccessible states, which can be thrown out).

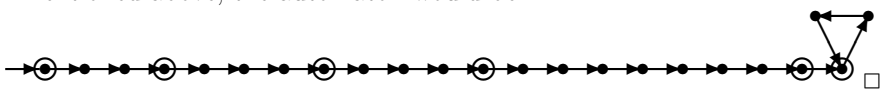


To see this, consider any DFA for A. Since the alphabet is $\{a\}$ and the machine is deterministic, there is exactly one edge out of each state, and it has label a. Thus there is a unique path through the automaton starting at the start state. Follow this path until the first time you see a state that you have seen before. Since the collection of states is finite, eventually this must happen. The first time this happens, we have discovered a loop. Let p be the length of the loop, and let n be the length of the initial tail preceding the first time we enter the loop. For all strings a^m with $m \geq n$, the automaton is in the loop part after scanning a^m. Then a^m is accepted iff a^{m+p} is, since the automaton moves around the loop once under the last p a's of a^{m+p}. Thus it is in the same state after scanning both strings. Therefore, the set of lengths of accepted strings is ultimately periodic.

Conversely, given any ultimately periodic set U, let p be the period and let n be the starting point of the periodic behavior. Then one can build an automaton with a tail of length n and loop of length p accepting exactly the set of strings in $\{a\}^*$ whose lengths are in U. For example, for the ultimately periodic set

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

mentioned above, the automaton would be



□

Corollary 12.4 *Let A be any regular set over any finite alphabet Σ, not necessarily consisting of a single letter. Then the set*

$$\textbf{lengths}\ A \quad = \quad \{|x| \mid x \in A\}$$

of lengths of strings in A is ultimately periodic.

Proof. Define the homomorphism $h : \Sigma \to \{a\}$ by $h(b) = a$ for all $b \in \Sigma$. Then $h(x) = a^{|x|}$. Since h preserves length, we have that **lengths** $A =$ **lengths** $h(A)$. But $h(A)$ is a regular subset of $\{a\}^*$, since the regular sets are closed under homomorphic image; therefore, by Theorem 12.3, **lengths** $h(A)$ is ultimately periodic. □

Historical Notes

A general treatment of ultimate periodicity and regularity-preserving functions is given in Seiferas and McNaughton [113]; see Miscellaneous Exercise 34.