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:
$$\begin{aligned} L(\alpha^*) &= \{x_1x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in L(\alpha),\ 1 \leq i \leq n\} \\ &= L(\alpha)^0 \cup L(\alpha)^1 \cup L(\alpha)^2 \cup \cdots \end{aligned}$$

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

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:

$$\begin{aligned} L(\alpha^+) &= \{x_1x_2\cdots x_n \mid n \geq 1 \text{ and } x_i \in L(\alpha),\ 1 \leq i \leq n\} \\ &= L(\alpha)^1 \cup L(\alpha)^2 \cup L(\alpha)^3 \cup \cdots \\ &= L(\alpha)^+. \end{aligned}$$

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

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

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:

  $$\boldsymbol{\epsilon} + @b. \qquad \square$$

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 $x \in a^*b^*$

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 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

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

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;

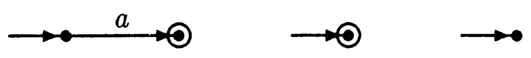
(iii) $A = L(\alpha)$ for some regular expression α.

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 $\varnothing$ 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 $a \in \Sigma$;	(vi)	$\beta + \gamma$;
(ii)	ϵ;	(vii)	$\beta \cap \gamma$;
(iii)	$\emptyset$;	(viii)	$\beta\gamma$;
(iv)	$\#$;	(ix)	$\sim\beta$;
(v)	$@$;	(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. □

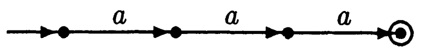
Example 8.2 Let's convert the regular expression

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

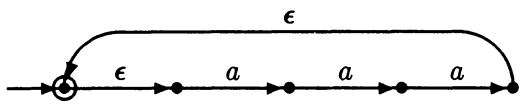
for the set

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

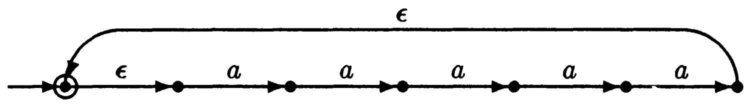
to an equivalent NFA. First we show how to construct an automaton for $(aaa)^*$. We take an automaton accepting only the string aaa, say



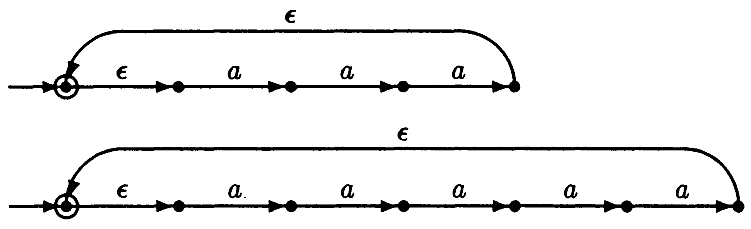
Applying the construction of Lecture 6, we add a new start state and ϵ-transitions from the new start state to all the old start states and from all the old accept states to the new start state. We let the new start state be the only accept state of the new automaton. This gives



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:



□