Regular Languages and Finite Automata

8 lectures for CST Part IA

Prof. Andrew Pitts

Andrew.Pitts@cl.cam.ac.uk, WGB FC08

Course web page:

www.cl.cam.ac.uk/teaching/1213/RLFA/

Pattern matching

What happens if, at a Unix/Linux shell prompt, you type

```
ls *
```

and press return?

Suppose the current directory contains files called `regfla.tex`, `regfla.aux`, `regfla.log`, `regfla.dvi`, and (strangely) `.aux`. What happens if you type

```
ls *.aux
```

and press return?

Alphabets

An ***alphabet*** is specified by giving a finite set, Σ, whose elements are called ***symbols***. For us, any set qualifies as a possible alphabet, so long as it is finite.

Examples:

$\Sigma_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ — 10-element set of decimal digits.

$\Sigma_2 = \{a, b, c, \ldots, x, y, z\}$ — 26-element set of lower-case characters of the English language.

$\Sigma_3 = \{S \mid S \subseteq \Sigma_1\}$ — 2^{10}-element set of all subsets of the alphabet of decimal digits.

Non-example:

$\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ — set of all non-negative whole numbers is not an alphabet, because it is infinite.

Strings over an alphabet

A ***string of length*** n (≥ 0) over an alphabet Σ is just an ordered n-tuple of elements of Σ, written without punctuation.

Example: if $\Sigma = \{a, b, c\}$, then a, ab, aac, and $bbac$ are strings over Σ of lengths one, two, three and four respectively.

$\Sigma^* \stackrel{\text{def}}{=}$ set of all strings over Σ of any finite length.

N.B. there is a unique string of length zero over Σ, called the ***null string*** (or ***empty string***) and denoted $\boxed{\varepsilon}$ (no matter which Σ we are talking about).

Concatenation of strings

The ***concatenation*** of two strings $u, v \in \Sigma^*$ is the string uv obtained by joining the strings end-to-end.

Examples: If $u = ab$, $v = ra$ and $w = cad$, then $vu = raab$, $uu = abab$ and $wv = cadra$.

This generalises to the concatenation of three or more strings.
E.g. $uvwuv = abracadabra$.

Regular expressions over an alphabet Σ

- each symbol $a \in \Sigma$ is a regular expression
- ε is a regular expression
- $\emptyset$ is a regular expression
- if r and s are regular expressions, then so is $(r|s)$
- if r and s are regular expressions, then so is rs
- if r is a regular expression, then so is $(r)^*$

Every regular expression is built up inductively, by *finitely many* applications of the above rules.

(N.B. we assume ε, $\emptyset$, $($, $)$, $|$, and * are not symbols in Σ.)

Matching strings to regular expressions

- u matches $a \in \Sigma$ iff $u = a$
- u matches ε iff $u = \varepsilon$
- no string matches $\emptyset$
- u matches $r|s$ iff u matches either r or s
- u matches rs iff it can be expressed as the concatenation of two strings, $u = vw$, with v matching r and w matching s
- u matches r^* iff either $u = \varepsilon$, or u matches r, or u can be expressed as the concatenation of two or more strings, each of which matches r

Examples of matching, with $\Sigma = \{0, 1\}$

- $0|1$ is matched by each symbol in Σ
- $1(0|1)^*$ is matched by any string in Σ^* that starts with a '1'
- $((0|1)(0|1))^*$ is matched by any string of even length in Σ^*
- $(0|1)^*(0|1)^*$ is matched by any string in Σ^*
- $(\varepsilon|0)(\varepsilon|1)|11$ is matched by just the strings ε, 0, 1, 01, and 11
- $\emptyset 1|0$ is just matched by 0

Languages

A (formal) ***language*** L over an alphabet Σ is just a set of strings in Σ^*.

Thus any subset $L \subseteq \Sigma^*$ determines a language over Σ.

The ***language determined by a regular expression*** r over Σ is

$$L(r) \stackrel{\text{def}}{=} \{u \in \Sigma^* \mid u \text{ matches } r\}.$$

Two regular expressions r and s (over the same alphabet) are ***equivalent*** iff $L(r)$ and $L(s)$ are equal sets (i.e. have exactly the same members).

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r (over the same alphabet), computes whether or not u matches r?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s (over the same alphabet), computes whether or not they are equivalent? (Cf. Slide 8.)

(d) Is every language of the form $L(r)$?