COMP235 : Logic and Computation

Supplementary notes on languages

1 Defining Strings and Formal languages

What is a language?

We are all familiar with *natural languages*, such as English, Chinese, Latin and so on. These are very complicated in general (though surprisingly have many important structural features in common).

A simpler idea is that of a *formal language*, which can be defined using some clear set of rules, for example a *grammar*. Example: programming languages.

The theory of formal languages we are about to consider in detail is very important for the construction and theory of compilers, and for pattern matching (for example in a document). It is also important for understanding natural languages. For us, the main importance is the link with abstract machines (which are used to formalise the idea of a computation).

To define a formal language properly, we first need to consider some simpler concepts.

Strings

- *alphabet*: any finite set X of *symbols* (= characters, = letters);
- *string*: (= word) a finite sequence of symbols from a fixed alphabet X (and we say " a string over X";
- *concatenation of two strings*: given two strings over the same alphabet w_1, w_2, their concatenation is w_1w_2, the string obtained by "pasting together" w_1 and w_2 (in that order).

For example, let $X = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ be an alphabet (the "usual" one!). Then an example of a string over X is $w_1 = \mathtt{bird}$. But so is $w_2 = \mathtt{fghi}$. The concatenation of these two strings is $w_1w_2 = \mathtt{birdfghi}$. Note that this is not the same as their concatenation in the opposite order $w_2w_1 = \mathtt{fghibird}$.

A commonly used alphabet is $X = \{\mathtt{0}, \mathtt{1}\}$, and we say any string over this X is a *binary string*.

Some notation. Let X be any alphabet.

- λ denotes the *empty string* over X, which is the string with no symbols in it.
- If $\mathtt{a} \in X$, then for any integer $n > 0$, $\mathtt{a}^n$ is shorthand for $\mathtt{aaa}\cdots\mathtt{a}$ (n times). We also define $\mathtt{a}^0 = \lambda$. Note that the usual index laws of algebra work: for $m, n \geq 0$:

$$\mathtt{a}^m\mathtt{a}^n = \mathtt{a}^{m+n}.$$

Concatenation of strings is always *associative*:

$$(w_1w_2)w_3 = w_1(w_2w_3)$$

for all strings w_1, w_2, w_3 over an alphabet X. And λ acts as an identity element under concatenation: $w\lambda = \lambda w = w$ for all strings w over X.

Formal languages

We can now make the following definition. A *formal language* (or just *language*) over the alphabet X is any set of strings over a finite alphabet X.

For example, let $X = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ as before. Some examples of languages over X are as follows.

- $A = \{\mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}, \ldots\}$, the set of all non-empty strings consisting only of as. Note that A is an infinite language.
- $B = \{\mathtt{ab}, \mathtt{aabb}, \mathtt{aaabbb}, \ldots\}$.
- $C = \{\mathtt{fghi}, \mathtt{gk}, \mathtt{zwx}\}$.
- $D = \{\mathtt{bird}, \mathtt{cat}, \mathtt{dog}\}$.

Any language consisting of binary strings is called a *binary language*.

For any alphabet X, $\Lambda = \{\lambda\}$ is the language consisting only of the empty string. Note that this is different from $\emptyset$, the empty language, which has *no* strings in it.

Operations on languages

Let A, B be languages over the same alphabet X.

- The *sum* of A and B, $A+B$, is the *union* of the sets A, B and is also a language over X.
- The *concatenation of A with B* is

  $$AB = \{w_1w_2 \mid w_1 \in A, w_2 \in B\},$$

  all possible concatenations of strings from A with strings from B *in that order*.
- the *Kleene closure* of A, A^* is defined to be the infinite union

  $$A^* = \Lambda + A + AA + AAA + \cdots,$$

  which is the set of all strings made up of any number of strings from A concatenated together (including the empty string).

Notation: as for strings, we write $A^n = AAA\cdots A$ (n times) for any positive integer n, and define $A^0 = \Lambda$. This makes sense because concatenation of languages over a fixed alphabet is associative (just like concatenation of strings):

$$(AB)C = A(BC).$$

This notation allows us to write the Kleene closure of A as

$$A^* = \sum_{n \geq 0} A^n = \Lambda + A + A^2 + A^3 + \cdots$$

The Kleene closure operation is the most tightly bound operation, followed by concatenation, and then union. Thus in the absence of brackets, we are to read $A+BC^*$ as first computing C^*, then $B(C^*)$, then the sum $A+(B(C^*))$. Fortunately, this is quite similar with the usual convention in algebra!

Some examples. Let $X = \{0, 1\}$, with $A = \{0\}^* = \{\lambda, 0, 00, 000, \ldots\}$ and $B = \{1\}$ two languages over X, hence binary languages.

- $A + B = \{0^n \mid n \geq 0\} \cup \{1\}$, which in words is all strings consisting of any number of 0s or a single 1.
- $AB = \{0^n 1 \mid n \geq 0\}$, which is all strings consisting of any number of 0s followed by a single 1.
- $A(BB)^* = \{0^n 1^{2m} \mid n, m \geq 0\}$, which is all strings consisting of any number of 0s followed by an even number of 1s.

In fact the operations on languages satisfy a number of other laws, such as

- $A + B = B + A$
- $(A + B) + C = A + (B + C)$
- $A(BC) = (AB)C$
- $A^{**} = A^*$,

among others.

Elements of an alphabet X can themselves be viewed as strings of length 1 over X, and we view X as a language over itself (consisting of all strings of length 1). Because of this, a very convenient way to describe the set of all strings over an alphabet X is as X^*: this is all possible strings formed from the strings in X, i.e. the letters in X.

2 Regular Languages

These are algebraic expressions built out of the language operations.

The regular languages on an alphabet X can be defined recursively as follows:

1. $\emptyset$, Λ are regular languages
2. every set consisting of a single element of X is a regular language
3. if E and F are regular languages, so are EF, $E + F$, E^*.

The other point is that for $\mathtt{x} \in \mathtt{X}$, we abbreviate $\{\mathtt{x}\}$ to $\mathtt{x}$, allowing the context to tell us whether we mean the symbol $\mathtt{x}$ or the language $\{\mathtt{x}\}$ when we write simply '$\mathtt{x}$'. Then the language concatenation which is normally written as $\{\mathtt{1}\}\{\mathtt{1}\}$ can more simply be written as $\mathtt{11}$.

For example, if $X = \{\mathtt{0}, \mathtt{1}\}$ then $R = (\mathtt{0} + \mathtt{11})^*$ is a regular language over X, and then

$$\begin{aligned} R &= (\mathtt{0} + \mathtt{11})^* \\ &= (\{\mathtt{0}\} + \{\mathtt{1}\}\{\mathtt{1}\})^* \\ &= (\{\mathtt{0}\} + \{\mathtt{11}\})^* \\ &= \{\mathtt{0}, \mathtt{11}\}^* \\ &= \{\lambda, \mathtt{0}, \mathtt{00}, \mathtt{11}, \mathtt{011}, \mathtt{110}, \mathtt{000}, \mathtt{0011}, \mathtt{0110}, \cdots\} \\ &= \text{ all strings in which any } \mathtt{1}\text{s occur in pairs.} \end{aligned}$$

Every finite language is regular. To see this, consider the following example of a binary language: $A = \{\mathtt{0}, \mathtt{101}, \mathtt{1100}\}$. What regular expression determines this? Easy: $A = \mathtt{0} + \mathtt{101} + \mathtt{1100}$! (Just compute to verify this.) This generalises to any finite language in the obvious way.

Regular expressions are very useful in pattern matching (many Unix tools use them for instance).