COMP11212 - Fundamentals of Computation

Todd Davies

January 30, 2014

Introduction

The building of real-life computing systems, e.g. mobile phone, tv/video remote control, internet shopping, air-traffic control, internet banking, etc., is always a complex task. Mistakes can be very annoying, costly and sometimes life threatening. Methods and techniques to support the building and understanding of such systems are essential. This course unit provides an introduction to the basic computer science ideas underlying such methods. It is also a part of, and an introduction to, the Modelling and Rigorous Development theme.

Aims

This course unit provides a first approach to answering the following questions. What methods are there that can help understanding complicated systems or programs? How can we make sure that a program does what we intend it to do? How do computers go about recognizing pieces of text? If there are two ways of solving the same problem, how can we compare them? How do we measure that one of them gives the solution faster? How can we understand what computers can do in principle, and are there problems that are not solvable by a computer?

Additional reading

None.

Contents

1 Terminology

In order to talk about Strings in any meaningful way, we must first define terminology that we can use to describe exactly what we mean. What follows is a list of the terminology used throughout the course:

- A **symbol** is basically a letter. They are the basic component of all the data we use in the course. Examples include: *a, A, (, $, 7*.
- An **alphabet** is a collection of symbols that we can think of as a set. Example alphabets may include binary (*0, 1*), latin letters ($a,\ldots,z,A,\ldots,Z$) etc.
- A **String** is a collection of symbols grouped together, sometimes called a word. Examples include *ababa* and *100101*.
- The *empty word* is a String consisting of no symbols. It is denoted by the letter ϵ.
- **Concentration** an operation that takes two Strings and combines them to create one longer String. For example concentrating *t* and *he* would create *the*. We can use the power notation to concentrate a String with itself any number of times. For example, ho^3 would give us *hohoho*.
- A **language** is a collection of Strings that can be thought of as a set. Examples of languages couls be $\{\emptyset\}$, $\{\epsilon\}$, $\{hot, hotter, hottest\}$ or $\{a^n | n \in \mathbb{N}\}$.

We also have notation for describing generic instances of such entities:

Entity	Generic notation
Symbol	x, y, z
Alphabet	Σ
String	s, t, u
Language	$\mathcal{L}$

Languages as sets Since languages are thought of as sets, we can perform all the usual set operations on them (see my `COMP11120` notes for more information on these operations). Languages can be concentrated as described above, however, an interesting case is when a languages is subject to concatenation zero times ($\mathcal{L}^0$), since this would return the empty word ϵ.

Sometimes, we may want to define any finite number of concatenations of a language, and this is known as the **Kleene star**. The notation is $(\mathcal{L})^*$.

> Note:
>
> *Kleene* is pronounced like *genie*.

2 Describing languages

2.1 Describing languages through patterns

A pattern describes a generic form that a set of Strings can take. If any String from the set is compared with the pattern then it will match the pattern. Any String from outside the set will not match the pattern.

For example, the pattern $(ab)^*$ would match Strings such as ϵ, ab, $abab$, $abab \ldots ab$.

> Note:
>
> ϵ is matched here since it satisfies the pattern $(ab)^0$. This comes about because, the Kleene star matches all concatenations of a String.

2.2 Regular expressions

The terms *pattern* and *regular expression* are pretty much synonomous. The operators allowed in a regular expression are:

Empty pattern	The character $\emptyset$ is a pattern.
Empty word	The character ϵ is a pattern.
Letters	Every letter in Σ is a pattern.
Concatenation	If x and y are patterns, then so is (xy).
Alternative	If x and y are patterns, then so is $(x\|y)$.
Kleene Star	If x is a pattern, then so is (x^*).

> Note:
>
> If we were to analyse a pattern in a recursive fasion, then in order to end our analysis, we would eventually have to find one of a selection of *base cases*.The highlighted rows here represent *step cases* (that is to say that at least another level of recursion is needed to finish our analysis), while the un-highlighted lines are base cases.

2.2.1 Discarding brackets

Just like in high-school mathematics, brackets can be discarded when unnecessary. For example, the pattern ((0|1)*0) is equivalent to (0|1)*0, and (2|(1|0)) is the same as (2|1|0).

2.2.2 Matching a regular expression

As was implied above, we can define a regular expression by recursively applying more operators to a 'base case' until we have the desired pattern.