COMP5190 – Theory of Computing – Lecture 6 – Regular Languages

Recognising languages:

* A way to decide whether a string of text is part of a language or not
* Binary decision: yes or no
* Works with different languages: English, Java, Machine code, etc.
* Can also be used to filter strings, e.g. recognising valid email addresses or phone numbers
* Characterises computational power
* Different types of languages require different computational models to be recognised
* Some languages can be recognised using fixed finite memory
* Some may require much more than that

Formal languages:

* In order to recognise a language, we need a way to define it:
  + **Alphabet** is a collection of characters that can appear in the language
  + **Word** is a finite string of characters from the alphabet
  + **Language** is a set of words
  + **Grammar** is a set of rules that define a language

Example:

* Consider the alphabet {0, 1}
* We can make words such as "0101", "11111", "0", ε
* We can define languages, e.g. a language of all binary strings that end in 00
* "011100", "1001", "100000", "00", "11010", "0", ε
* Languages can be empty too
* L = {}
* Not particularly interesting now but sometimes comes up when set operations are applied to languages, more on that in later lectures
* A language of legal Java identifiers:
  + "reglang“
  + "reg\_lang\_1"
  + "abcdefghijklmnopqrstuvwxyz1234567890“
  + "whileifelse"
  + etc.
* A language of words consisting of balanced round brackets
  + ()
  + ((()))
  + (()((())())(()))
  + ε
  + etc.

Formalising languages:

* We need a formal way to describe languages
* So that we can communicate them to each other
* So that we can program computers to recognise them
* So that we can analyse their properties
* We need a "language“ that describes other languages

Languages are a set of words:

* "... of words", so we need a way of turning a word into a language
* "... a set ...", so we need a way to apply set theory operations to languages
* Languages can be infinite, so we need some form of iteration or recursion:
  + Regular languages offer iteration
  + Context-free languages offer recursion

Regular expressions:

* A widely used way of expressing regular languages
  + ABC
  + (A|B)
  + A\*
  + (A|B)\*B
  + etc.
* IEEE POSIX standard provides extensions that can technically go outside of the domain or regular languages
  + ^A+[0-9]\*B\{2,4\}$

Regular languages:

* A language that can be defined by a regular expression is a regular language
* Regular expressions are powerful, but have limitations
* E.g. A language of balanced round brackets is not regular as it cannot be defined by a regular expression

Automata and regular languages:

* Regular languages can be defined using automata, more about those in the next few lectures.