Session Objectives

- Give an overview of language structure and parsing
- Define the role of formal languages in Computer Science
- Define regular grammars
- Analyse sets of words and define a regular grammar that generates them

Languages

A language is a set of symbol sequences (called sentences) that can be interpreted (i.e. have a meaning). Our main aims are to

- look at different ways of formally defining the structure of languages;
- define methods for parsing (that is, determining whether a sentence is well-formed) languages;
- show how these ideas can be applied in different domains of Computer Science.

Looking Forward

1. language theory (a broad overview of our aims);
2. regular languages and regular expressions (the simplest useful class of languages);
3. finite state automata (simple machines that can parse and generate regular languages);
4. language definitions and grammars (introducing more general methods of defining languages);
5. Backus–Naur form (using grammars to define some important CS languages).

Natural and Formal Languages

- We all use natural language to communicate with each other. In a natural language there is no strictly formal way of deciding whether the syntax of a sentence is acceptable (i.e. whether it is a legal structure).
- The reasons why it is important to use languages with a more formal structure are mainly based on the fact that if we want to automate reading (or parsing) a language, there must be a formal and unambiguous way of deciding whether a sentence is correctly structured.
- We shall show that for some classes of formal languages a machine can be constructed to parse sentences.

Applications of Formal Languages

Compilers Programming languages are formal languages. The first thing that a compiler must do is to parse your text file and decide if it is a legal program. Only then can it determine the semantics (i.e. define its meaning, by checking types, etc.) before translating it into assembly language.

File Parsing Many computer programs require some data to be stored in a file (for example, configuration information, results, data structures). While it is easy enough to write such files, unless some care is taken, it can be difficult to read the files back in robustly (i.e. checking that the file structure is legal). By defining a file language formally, both the writing and the parsing of the file can be automated, thus speeding up the development process and reducing the likelihood of programming errors.

Formal Definitions Definitions in formal specifications often require a language: for example, the structure of wffs in logic. It makes sense to make these definitions themselves formally (rather than the semi-formal way they were introduced earlier).

Network Data Distribution The eXtensible Markup Language (XML) is a developing standard for markup languages that allow data and information to be shared over the World-Wide Web. XML permits document authors to create markup for virtually any type of information: mathematics, chemical structures, genomics, music, financial data exchange etc. Processing an XML document requires an XML parser to check the syntax and convert the text to machine-useable data. To carry out the parsing, the structure of the markup language must be defined formally.

Regular Languages and Regular Expressions

- Regular languages are the simplest useful class of formal language.
- Efficient parsers have been implemented for regular expressions (that is, sentences in a regular language) and these are readily accessible in Unix text processing utilities and programming languages.
- We will define regular languages and show some applications in text processing.

Regular Expressions

- An alphabet, denoted by Σ, is just a finite set of formal symbols: for example, lower case letters, or numbers.
- From this we can form strings (or 'words'), which are finite sequences whose members are drawn from Σ.
- The empty sequence ϵ is also included.

If $\Sigma = \{0, 1\}$ then ϵ, 0, 1, 001001010 are all words.

Write down all words of length 2 for the alphabet $\{a, b\}$.

Solution

Write down all words of length 2 for the alphabet $\{a, b\}$.

aa, ab, ba, bb

Concatenation Operator

- The concatenation of two words is formed by juxtaposing the symbols that form the words.
- If $w_1 = car$ and $w_2 = e$, then the concatenation of w_1 and w_2 is $w_1 w_2 = care$.
- This idea can be extended to sets of words in a natural way. If W_1 and W_2 are two sets of words, then the concatenation $W_1 W_2$ or $W_1 \cdot W_2$, is the set of all words formed by concatenating a word in W_1 with a word in W_2. Formally, this is $W_1 W_2 = \{w_1 w_2 | w_1 \in W_1 \quad \text{and} \quad w_2 \in W_2\}$.
- For example, $\{\text{car}, \text{di}\} \cdot \{\text{d}, \text{e}, \text{ve}\} = \{\text{card}, \text{care}, \text{carve}, \text{did}, \text{die}, \text{dive}\}$.
- Powers of W are used to denote the concatenation of W with itself the appropriate number of times. For instance, $W^2 = WW$ and $W^3 = WWW$. In addition, we let $W^0 = \{\epsilon\}$ and $W^1 = W$.

Exercise

If $W_1 = \{a, b\}$ and $W_2 = \{0, 1\}$ write down

1. W_1W_2
2. W_1^2
3. W_2^3

Solution

If $W_1 = \{a, b\}$ and $W_2 = \{0, 1\}$ write down

1. W_1W_2:

   $a0, a1, b0, b1$.

2. W_1^2:

   aa, ab, ba, bb.

3. W_2^3:

   $000, 001, 010, 011, 100, 101, 110, 111$.

Language Closure

- We define W^*, the Kleene closure of W to be the union of all the finite powers of W:

$$W^* = \bigcup_{i=0}^{\infty} W^i = W^0 \cup W^1 \cup W^2 \cup W^3 \cdots \cup .$$

- This is the set of all words (including ϵ) that can be formed by concatenating words from W any number of times.
- If $\Sigma = \{0, 1\}$ and $W = \{0, 10\}$, then W^* consists of the empty word ϵ and all words that can be formed using 0 and the pair 10; that is, all words formed from 0s and 1s with the property that every 1 is followed by a 0.

Regular Expressions

The regular expressions over Σ are defined as follows:

1. If $a \in \Sigma$, then a is a regular expression designating the set $\{a\}$.
2. If E and F are regular expressions designating the sets A and B respectively then:
   (a) $E + F$ is a regular expression designating the set $A \cup B$.
   (b) EF is a regular expression designating the set AB, the concatenation of A and B.
   (c) E^* is a regular expression designating the set A^*, the Kleene closure of A.

Example

Let $\Sigma = \{a, b\}$. Then the following are regular expressions defining the given sets of words:

- a denotes the set $\{a\}$.
- a^* denotes the set $\{\epsilon, a, aa, aaa, \ldots\}$.
- $(ab)^*$ denotes the set $\{\epsilon, ab, abab, ababab, \ldots\}$.
- $a^*b(ab)^*$ denotes the set of all words that begin with any number (possibly zero) of a's followed by a single b, followed by any number (possibly zero) of pairs ab.

Regular Sets

A regular set or regular language over an alphabet Σ is either

1. the empty set;
2. the set consisting only of the empty word;
3. a set defined by some regular expression over Σ.

For each of the following sets, give a regular expression that defines it. In each case $\Sigma = \{a, b, c\}$.

1. $\{\epsilon, b, a, aa, aaa, \ldots\}$.
2. $\{a, ab, ab^2, ab^3, \ldots\}$.
3. The set of words beginning with any number of pairs ab followed by a c, followed by any number of triplets abc.

Solution

For each of the following sets, give a regular expression that defines it. In each case $\Sigma = \{a, b, c\}$.

1. $\{\epsilon, b, a, aa, aaa, \dots\}$.

   $b + a^*$

2. $\{a, ab, ab^2, ab^3, \dots\}$

   $a(b)^*$

3. The set of words beginning with any number of pairs ab followed by a c, followed by any number of triplets abc.

   $(ab)^*c(abc)^*$

Limitations of Regular Expressions

- Although regular expressions are powerful, it is important to be aware that they do not generate all possible languages.
- In fact, there are quite simple languages that cannot be generated by a regular expression.
- Consider the set of sequences of 0s and 1s where a certain number of 0s are followed by the same number of 1s:

  $$\{\epsilon, 01, 0011, 000111, 00001111, \ldots\}.$$

  This set cannot be generated by a regular expression; we will sketch a proof of this later.

Session Objectives

- Give an overview of language structure and parsing
- Define the role of formal languages in Computer Science
- Define regular grammars
- Analyse sets of words and define a regular grammar that generates them