MA2C03 - DISCRETE MATHEMATICS - TUTORIAL NOTES

Brian Tyrrell

05 & 06/12/2016

This write up covers in a bit more detail what we covered in the tutorials.

1 Grammars

Recall that given an alphabet A, A^n is the set of all words of length n, where a *word* is a string of characters in A. We then defined

$$A^+ = \bigcup_{i=1}^{\infty} A^i$$

In essence, A^+ is the set of all words of A of positive length. We also had

$$A^* = \{\epsilon\} \cup A^+$$

where ϵ is the *empty string*. In essence, A^* is the set of words of A of any length (including now "zero length").

Definition 1.1. Let V be a set of terminal and nonterminal symbols, A a set of terminal symbols and $< s >$ denote the "start" symbol. We have the following types of grammars:

- A *context-free grammar* $(V, A, < s >, P)$ where $P \subset (V \setminus A) \times V^*$ is a set of production rules taking nonterminals to words of V; $< T > \to v$.
- A *context-sensitive grammar* $(V, A, < s >, P)$ where P are production rules of the form

$$v < T > w \to v\gamma w$$

  where $v, w \in V^*$, $< T > \in V \setminus A$ and $\gamma \in V^+$.
- A *phrase structure grammar* $(V, A, < s >, P)$ where $P \subset (V^+ \setminus A^+) \times V^*$.

Two notions that follow for each of these grammars are *directly yields* and *generated language*:

Definition 1.2. We'll define these in terms of context-free grammars:

- Let w', w'' be words over V. We say w' *directly yields* w'' if there exist words $u, v \in V^*$ and a production rule $< T > \to w$ of the grammar such that

$$w' = u < T > v \qquad \text{and} \qquad w'' = uwv$$

  This is written $w' \Rightarrow w''$.

- We say w' *yields* w'' (written $w' \Rightarrow^* w''$) if there exists a sequence of words, each directly yielding to the next, which begins with w' and ends with w''.
- The language generated by the context-free grammar $(V, A, < s >, P)$ is a subset $L \subseteq A^*$ defined as:
$$L := \{w \in A^* : \;\; < s > \Rightarrow^* w\}$$
  This is known as a *context-free language.*

Example 1.1. Consider $A = \{0, 1\}$, a start symbol $< s >$ and production rules:

(1) $< s > \rightarrow 0 < s > 1$

(2) $< s > \rightarrow \epsilon$

This generates the language $\{0^n 1^n : n \in \mathbb{N}\}$; this is a context-free language. $\lozenge$

Remark 1.1. Note the production rules of a grammar are not unique to a particular type of grammar - for example, the rule
$$< s > \rightarrow 0 < s > 1$$
can also appear as a rule in a context-sensitive grammar, with $v = w = \epsilon$ and $\gamma = 0 < s > 1$. $\lozenge$

2 Regular Languages

There are many ways to approach regular languages; we will first approach them as the output of a definition involving operations on a language, then second as the result of a finite state acceptor, and third as the language of a regular grammar.

Using the first approach, we define:

Definition 2.1. Let A be an alphabet. A subset $L \subset A^*$ is called a *regular language over the alphabet* A if $L = L_m$ for some finite sequence $L_1, \ldots, L_m$ of subsets of A^* with the property that for $i = 1, 2, \ldots m$ L_i satisfies one of the following:

(1) L_i is a finite set.

(2) $L_i = L_j^*$ for some $1 \leq j < i$.

(3) $L_i = L_j \circ L_k$ for some $1 \leq j, k < i$ (concatenation).

(4) $L_i = L_j \cup L_k$ for some $1 \leq j, k < i$.

Example 2.1. Let $A = \{0, 1\}$. The language $L = \{0^m 1^n : m, n \in \mathbb{N}\}$ is regular; we obtain it from the sequence:

(1) $L_1 = \{0\}$

(2) $L_2 = L_1^* = \{0^m : m \in \mathbb{N}\}$

(3) $L_3 = \{1\}$

(4) $L_4 = L_3^* = \{1^n : n \in \mathbb{N}\}$

(5) $L = L_3 \circ L_4 = \{0^m 1^n : m, n \in \mathbb{N}\}$

◊

Remark 2.1. There is a difference between concatenation two languages, and taking the union of two languages. For example, if

$$L_1 = \{00, 11\} \text{ and } L_2 = \{01, 10\}$$

Then their concatenation:

$$L_1 \circ L_2 = \{0001, 0010, 1101, 1110\}$$

differs from their union:

$$L_1 \cup L_2 = \{00, 11, 01, 10\}$$

◊

One might notice *Definition 2.1* gives a very precise and formal definition of a regular language - so precise and formal a machine would understand it. As it turns out, we can construct simple machines known as *finite state acceptors* whose job is to recognise a regular language.

Definition 2.2. We want a machine that, given a language L, returns "YES" if an inputted word w is an element of L, and "NO" otherwise. We'll define:

- A *finite state acceptor* (S, A, i, t, F) consists of a finite set S of states, a finite input alphabet A, an initial state $i \in S$, a transition map $t : (S \times A) \to S$ and a set of final states $F \subseteq S$.
- A word $a_1 a_2 \dots a_n$ of length n over the alphabet A is said to be *recognised* or *accepted* if there exist states $s_0, \dots, s_n$ in S such that
  - $s_0 = i$
  - $s_i = t(s_{i-1}, a_i)$ for all $i = 1, 2, \dots, n$
  - $s_n \in F$
- A language L over A is said to be *recognised* or *accepted* by the finite state acceptor if L is the set of *all* words accepted by the finite state acceptor.

Definition 2.3. These acceptors come in two flavours; deterministic and nondeterministic.

- *Deterministic:* Every state has exactly one transition for each possible input.
- *Nondeterministic:* Given an input and a state, there are multiple states the machine can move to next. This would mean the transition map is not a function (hence the reason why we call it a transition *map*).

There are two remarkable results:

Theorem 2.1. *From every nondeterministic finite state acceptor M a deterministic finite state acceptor M' can be constructed such that M and M' accept the same language.* ■

Theorem 2.2. *A language L over an alphabet A is a regular language $\Leftrightarrow$ L is recognised by a deterministic finite state acceptor with input alphabet A.* ■

In this way we can consider the regular languages to be the ones recognised by finite state acceptors.

Example 2.2. We can build a deterministic finite state acceptor recognising the regular language $L = \{0^m 1^n : m, n \in \mathbb{N}\}$. Set $S = \{i, s_1, s_2, s_3\}$ where i is the initial state, $F = \{i, s_1, s_2\}$, $A = \{0, 1\}$ and we can draw the following diagram to illustrate the transition function:

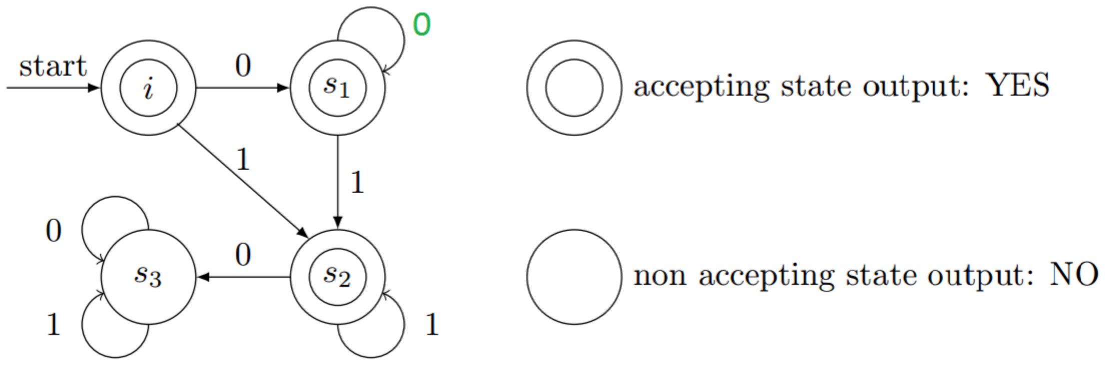

Figure 1: Source: *Online notes, MA2C03 Lecture 23.*

◊

Finally, we can tackle regular languages as an output of a regular grammar:

Definition 2.4. A context-free grammar $(V, A, < s >, P)$ is called a *regular* grammar if every production rule is one of three forms:

(1) $< A > \to b < B >$

(2) $< A > \to \epsilon$

(3) $< A > \to b$

where $< A >, < B >$ are nonterminals and b is a terminal.
A regular grammar is in *normal form* if all its production rules are of type (1) and (2).

Being in normal form isn't particularly special, just convenient to work with. It can be shown that every regular grammar can be 'converted' to normal form, i.e. a regular grammar in normal form can be constructed that outputs the same language.

Example 2.3. There is a very big difference between $\{0^m 1^n : m, n \in \mathbb{N}\}$ and $\{0^n 1^n : n \in \mathbb{N}\}$. The former is regular, as shown in *Example 2.2*, however the latter is **not**. Informally, we can think of a reason why - if $\{0^n 1^n : n \in \mathbb{N}\}$ was regular, then it would be recognised by a finite state acceptor. Given a string $00 \ldots 011 \ldots 1$ in this language, the FSA would have to remember how many 0's there were and compare that to the number of 1's. As the strings $00 \ldots 011 \ldots 1$ can be arbitrarily large, this FSA would have to remember that it saw an arbitrarily large number of 0's, which a *finite* state acceptor cannot do. Thus $\{0^n 1^n : n \in \mathbb{N}\}$ is not regular.
This informal proof can be formalised by using the Myhill-Nerode Theorem. $\Diamond$

Similarly it can be shown by construction there is a finite state acceptor recognising the same language that a regular grammar outputs. We can then add to *Theorem 2.2:*

Theorem 2.3. *A language L over an alphabet A is a regular language $\Leftrightarrow$ L is recognised by a deterministic finite state acceptor with input alphabet A $\Leftrightarrow$ L is generated by a regular grammar.* ■