# 3. Finite state automata

1984 Cannon extended Dehn's solution to the word problem to cocompact discrete groups of hyperbolic isometries ([Ca1]). Perhaps the main idea of Cannon's paper is that one can "see" the Cayley graphs of such groups. What does it mean to see the Cayley graph of an infinite group? For example, the Cayley graph of $(\mathbf{Z}, S)$ with $S = \{1\}$ is simply the real line with a vertex at each integer. When drawing a picture of $\Gamma_S(\mathbf{Z})$, we draw only the two-ball, say, and then a trailing line of dots $\cdots$. By this we really mean to repeat the picture of the two-ball in our heads off to infinity, thinking also of the linear recursion $n \mapsto n + 1$. To "see" the Cayley graph should mean to have a picture of some finite ball around the origin and some finite machine which tells us how to piece copies of this ball together out to infinity. Cannon suggested as an open problem that one might "Formalize the notion that a Cayley graph can be described by linear recursion, and devise efficient algorithms for working out that recursion for many examples." The idea is that if such a linear recursion exists, which should happen whenever there is some pattern in the Cayley graph, then we can build a picture of what the group looks like, and from this picture we can construct algorithms to do computations in the group, such as solving the word problem.

The next layer of foundation was provided by Thurston, who gave a formal definition of the "linear recursion" Cannon spoke of. Thurston did this by using finite state automata (FSA for short), the simplest type of machines which have been studied thoroughly by computer scientists for nearly forty years. It seems interesting that Gilman was independently exploring the use of finite state automata for normal forms in groups (see, e.g. [Gi]), although with no (explicit) discussion of geometry. The details of the basic theory of automatic groups were worked out at Warwick by Epstein, Holt and Paterson. The use of finite state automata is partly motivated by their success in both the theory and applications of computer science; most word-processors (including 'vi') construct finite state automata for tasks such as word searches, and many compilers use FSA during lexical and syntactical analysis. In order to understand automatic groups we'll first need to have some understanding of finite state automata.

## 3. Finite state automata

Given some finite set of letters $\mathscr{A} = \{a_1, \ldots, a_n\}$, we want to pick out a nice subset of the set $\mathscr{A}^*$ of all words in the letters $a_i$ (one can view $\mathscr{A}^*$ as the free monoid generated by the elements of $\mathscr{A}$). A subset $L \subseteq \mathscr{A}^*$ is called a *language*. Informally, a *finite state automaton* $W$ over $\mathscr{A}$ is a finite directed

graph with vertices called *states*, written as small circles; a special vertex called the *start state* of $W$, with the letter '$s$' (for 'start') written inside it; and directed edges connecting the vertices, each edge labelled with a letter from $\mathscr{A}$. For any given label, each vertex can have at most one edge directed out of it with this fixed label. Finally, we pick a subset $Y$ of states which we call *accept states*, and draw the vertices of accept states as double circles. States not in $Y$ are referred to as *fail states*. Examples of some finite state automata are given in figure 2.

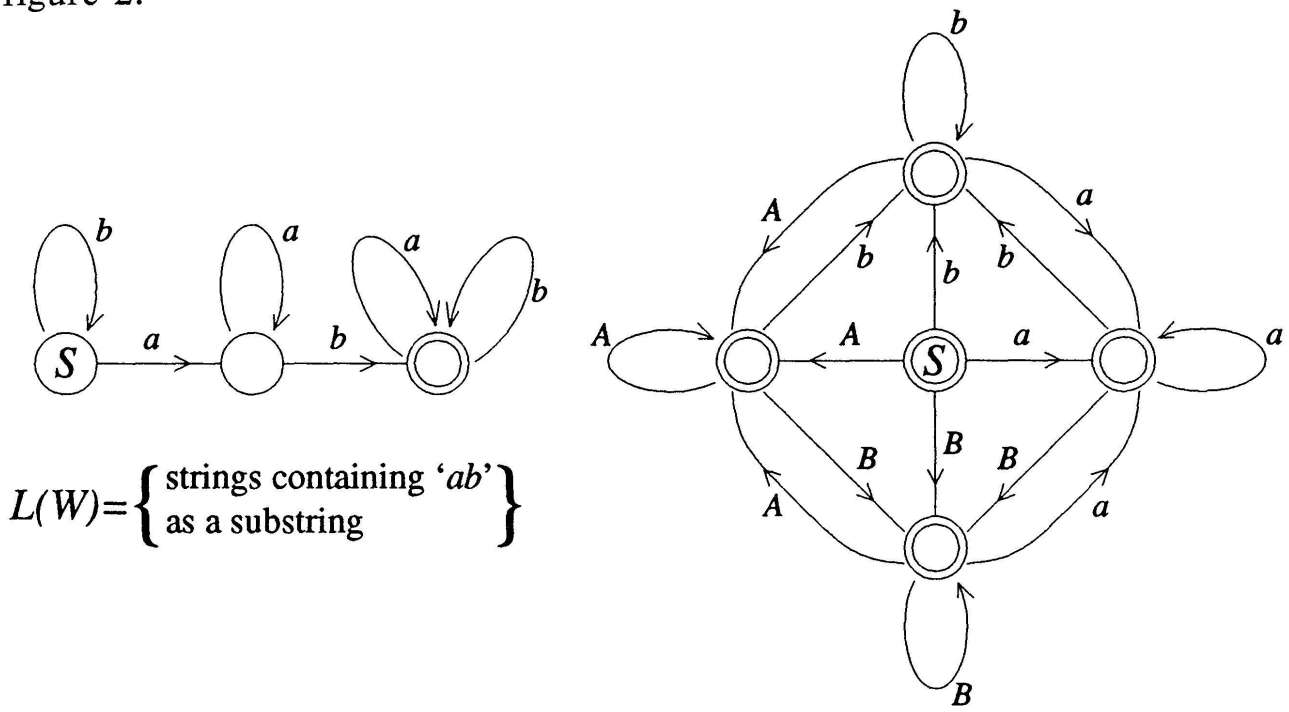


FIGURE 2

Some finite state automata and their accepted languages

A finite state automaton $W$ gives a language over $\mathscr{A}$ as follows: If $w = w_0 w_1 \ldots w_n$ is a word with each $w_i \in \mathscr{A}$, then beginning at the start state $s$, move from state to state by reading along the edge labelled $w_0$, then along $w_1, \ldots$, then along $w_n$. If at any time the current state is $v$, and the next letter which should be read is $w_i$ but there is no edge directed out of $v$ labelled $w_i$, then $w$ is not accepted by $W$. If after reading the word $w$ the current state is an accept state, then $w$ is accepted by $W$, otherwise $w$ is not accepted by $W$. The set of accepted words $L = L(W)$ is said to be the *language accepted by* $W$. Note that if the start state $s$ is an accept state then the empty word is an element of $L$. A language which is accepted by a finite state automaton is called a *regular language.*

Regular languages are the simplest, most important languages studied by computer scientists. They are very special — most languages (i.e., subsets of $\mathscr{A}^*$) are not regular. An example of a language over the alphabet $\mathscr{A} = \{a, b\}$ which is not regular is the set $\{a^n b^n : n \in \mathbf{Z}\}$; this follows immediately from the well-known "pumping lemma" of computer science (see [HU]). Note that $\{a^n b^m : n, m \in \mathbf{Z}\}$ is regular. The reason why $\{a^n b^n : n \in \mathbf{Z}\}$ is not regular is that a finite state automaton has no memory, and so cannot know exactly how many $b$'s to accept after having accepted $n$ $a$'s.

There are other ways to define regular languages via certain grammatical operations and other machines similar to the automata described above (see [Ep1], [E *et al.*] or [HU]). Different (though equivalent) definitions of regular languages are useful in different situations, but for us the above definition will suffice.

Before giving the definition of automatic groups we will need the notion of a *two-variable padded language*. Given an alphabet $\mathscr{A}$, we can add a padding symbol $\$ \notin \mathscr{A}$ to form the alphabet $\mathscr{A} \cup \{\$\}$, and we can consider a finite state automaton $W$ as above, but this time with labels in $(\mathscr{A} \cup \$) \times (\mathscr{A} \cup \$) \setminus (\$, \$)$. Given a pair of words $(u, v) \in \mathscr{A}^* \times \mathscr{A}^*$, say $u = u_1 \cdots u_n, v = v_1 \cdots v_m$ with $m \leqslant n$, we pad $v$ with the symbol $\$$ so that the resulting words have equal length. We will say that $(u, v)$ is accepted by $W$ if we can read off the edges $(u_1, v_1), \ldots, (u_m, v_m), (u_{m+1}, \$), \ldots, (u_n, \$)$ and end up at an accept state of $W$. The set of accepted pairs $(u, v)$ is said to be *regular over the (padded) alphabet* $\mathscr{A}$. The point of padding is that pairs of words can be read at equal speeds, even if the words have different lengths.

## 4. Automatic groups: definitions and examples

The definition of automatic group involves only finite state automata. We will later show this to be equivalent to a more geometric, and perhaps easier to understand, condition.

Let $G$ be a group with finite generating set $\mathscr{A} = \{a_1, \ldots, a_n\}$ such that $\mathscr{A}$ actually generates $G$ as a monoid. $\mathscr{A}$ is most often chosen as $\mathscr{A} = S \cup S^{-1}$, where $S$ is a finite set of (group) generators for $G$ and $S^{-1}$ is the set of inverses of the elements of $S$. Notice that there is a natural map from $\mathscr{A}^*$, the free monoid on $\mathscr{A}$, to the group $G$ which takes a word to the group element which it represents; we will denote this map by $w \mapsto \bar{w}$. $G$ is an *automatic group* if the following conditions hold: