Chapter 7

Finite State Machine based Text Generation

Once connected components are recognized by recognition process, next step is to replace this set of labels with appropriate Unicode values and create a computer file which then can be opened and edited in text editor. In this chapter we describe the process that we use to carry out text generation.

7.1 Mathematical Preliminaries

As mentioned above we analyze the set of recognized symbols and group them so that the resultant group represents a valid consonant-vowel combination. We can model this process with the help of a *Finite State Automaton* or *Finite State Machine*. Here, we will introduce the concept and in the later sections we will show how it is used for text generation in Gujarati OCR.

7.1.1 Finite Automaton / Finite-State Machine [27]

Finite automaton or *finite-state machine* can be thought of as a severely restricted model of an actual computer. The finite automaton shares with a real computer the fact that it has a "Central Processing Unit" of fixed, finite capacity. It receives its input as a string, delivered to it on an input tape. It delivers no output at all, except an indication of whether or not the input is considered acceptable. What makes the finite automaton such a restricted model of real computers is the complete absence of memory outside its fixed central processor.

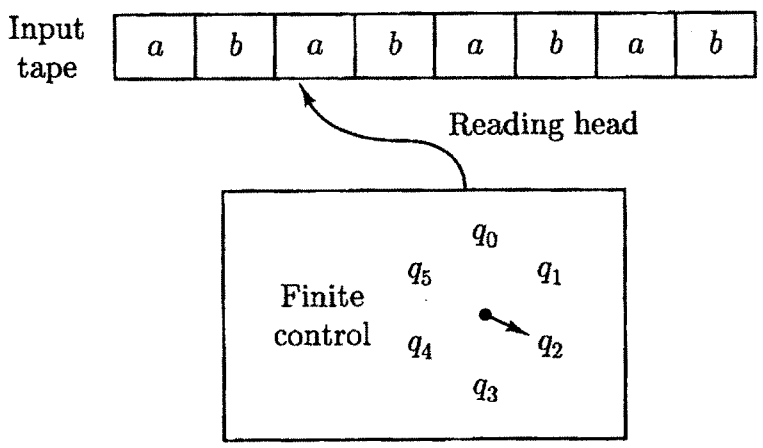


Figure 7.1: Finite State Automata - Schematic Diagram

As mentioned above strings are fed into the device by means of an *input tape*, which is divided into squares, with one symbol inscribed in each tape square (see Fig. 7.1). The main part of the machine itself is a "black box" with innards that can be, at any specified moment, in one of the finite number of distinct internal *states*. This black box – called the *finite control* – can sense what symbol is written at any position on the input tape by means of a movable *reading head*. Initially, the reading head is placed at the left most square of the tape and the finite control is set in a designated *initial state*. At regular intervals the automaton reads one symbol from the input tape and then enters a new state *that depends only on the current state and the symbol just read* – this is why we shall call this device a *deterministic* finite automaton. After reading an input symbol, the reading head moves one square to the right on the input tape so that on the next move it will read the symbol in the next tape square. This process is repeated again and again; a symbol is read, the reading head moves to the right, and the state of the finite control changes. Eventually the reading head reaches the end of the input string. The automaton then indicates its approval or disapproval of what it has read by the state it is in at the end: if it winds up in one of the *final states* the input string is considered to be *accepted.*

Deterministic Finite Automaton

Mathematically, a *deterministic finite automaton* is a quintuple $M = (K, \Sigma, \delta, s, F)$ where,

K is a finite set of *states* ,
Σ is an alphabet,
$s \in K$ is the *initial state*
$F \subseteq K$ is the set of *final states*, and

δ, the *transition function*, is a function from $K \times \Sigma$ to K

The rules according to which the automaton M picks its next state are encoded into the transition function. Thus if M is in state $q \in \Sigma$, then $\delta(q, a) \in K$ is the uniquely determined state to which K passes.

Having formalized the basic structure of a deterministic finite automaton, we must next render into mathematical terms the notion of a *computation* by an automaton on an input string. this will be, roughly speaking, a sequence of *configurations* that represent the status of the machine (the finite control, reading head, and input tape) at successive moments. But since a deterministic finite automaton is not allowed to move its reading head back into the part of the input string that has already been read, the portion of the string to the left of the reading head cannot influence the future operation of the machine. Thus, a configuration is determined by the current state and the unread part of the string being processed. In other words, a *configuration of a deterministic finite automaton* $(K, \Sigma, \delta, s, F)$ is any element of $K \times \Sigma^*$.

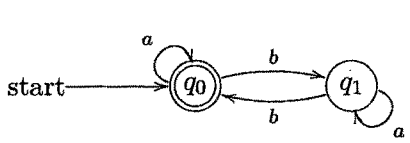
Example 7.1.1 *Let M be the deterministic finite automaton $(K, \Sigma, \delta, s, F)$ where,*

$$K = \{q_0, q_1\},$$
$$\Sigma = \{a, b\},$$
$$s = q_0,$$
$$F = \{q_0\},$$

and δ is the function tabulated below.

q	σ	$\delta(q, \sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

The tabular representation of the transition function used in this example is not the clearest description of a machine. We generally use a more convenient graphical representation called the *state diagram*, as given below :



Thus the *state diagram* is a directed graph, with certain additional information incorporated into the picture. States are represented by nodes, and there is an arrow labeled with a from node q to q' whenever $\delta(q, a) = q'$. Final states are indicated by double circles, and the initial state is shown by an arrow originating from word *start*. State Diagram

7.2 Text Generation

We use finite state automaton to model the text generation process. It is completed in two stages.

1. Glyph cluster generation
2. Unicode Substitution

At the first stage we analyze the recognized glyphs and group glyph codes of a consonant-vowel(CV) combination. Instead of consonant it could be glyph(s) corresponding to a consonant cluster, also known as conjunct. Then at the second stage of processing these glyph clusters are analyzed to substitute corresponding Unicode.

7.2.1 Glyph Cluster Generation

Glyph cluster is a set of glyphs corresponding to a valid C*(CV) combination. It is important to note here that in case of independent vowels like આ,એ,ઔ,ઐ,ઓ etc. glyph cluster refers to set comprising of અ and connected components ◌ા, ◌ૌ etc.. We use the coordinates of the connected component of a word and the zone information to accomplish this. Glyph Cluster

Essentially, in this process we sort all the middle zone connected components in the order of the column coordinate of their top-left corner and create clusters of these components. It is known that some of the alphabet/conjuncts of Gujarati give rise to more than one connected component. For example, ગ, શ, ઙ. Connected components of such alphabet / conjunct are to be collected and kept in same cluster. Then connected component in the upper and lower zones are analyzed and they are added to appropriate cluster of middle zone depending on the column coordinate of the center of their bounding boxes.

For creating clusters from middle zone glyphs, we divide all the possible glyphs of the middle zone in two subsets viz. the set of glyphs that can come as the

first glyph in the glyph cluster, say b and set of other glyphs, say o.

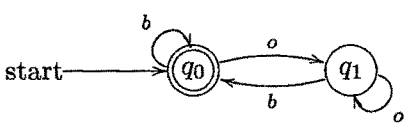
$U = \{u/u$ is a possible connected component (unit of recognition)$\}$

$B = \{b/b =$ consonant, conjunct, independent vowel, first part of$\}$

consonant / conjunct made up of more than one connected component$\}$

$O = \{o/o \in U \sim B\}$

The finite state automaton for this process of creating glyph clusters from middle zone glyphs can be given as follows :



As mentioned above next step is to assign the connected components extracted from the upper and lower zones of the word to appropriate cluster of middle zone glyphs. The steps involved are as follows :

1. Compute the bounding box coordinates for the glyph clusters of middel zone glyph as follows :
   Let,
   $(lr_1, lc_1), (lr_2, lc_2), \ldots, (lr_n, lc_n)$ be the coordinates of the top left corners and
   $(rr_1, rc_1), (rr_2, rc_2), \ldots, (rr_n, rc_n)$ be the coordinates of the bottom right corners of the bounding boxes of the glyphs constituting a glyph cluster.

   The coordinates of the top left and bottom right corners of the bounding box of the glyph cluster, say $(gclr, gclc)$ and $(gcrr, grrc)$ are given by

   $$gclr = \min_i lr_i \qquad gclc = \min_i lc_i \tag{7.1}$$

   $$gcrr = \max_i rr_i \qquad gcrc = \max_i rc_i \tag{7.2}$$

2. Compute center of the bounding box of the glyph in upper /lower zone

3. Assign glyph to the i^{th} glyph cluster, say g_i if the column coordinate of the center of its bounding box, cy of the glyphs lies between the column coordinates of top left and bottom right corners of g_i . Mathematically,

   $$cy > gclc_i \qquad \text{and} \qquad cy < gcrr_i \tag{7.3}$$

7.2.2 Unicode Substitution

Unicode is a computing industry standard for the consistent representation and handling of text expressed in most of the world's writing systems. Developed in conjunction with the Universal Character Set standard and published in book form as The Unicode Standard, the latest version of Unicode consists of a repertoire of more than 107,000 characters covering 90 scripts(including Indic scripts), a set of code charts for visual reference, an encoding methodology and set of standard character encodings, an enumeration of character properties such as upper and lower case, a set of reference data computer files, and a number of related items, such as character properties, rules for normalization, decomposition, collation, rendering, and bidirectional display order (for the correct display of text containing both right-to-left scripts, such as Arabic or Hebrew, and left-to-right scripts).

Unicode provides 128 code points for each of the Indic scripts that are accommodated in the standard. These code points are used to represent the basic set of alphabet, numbers and special symbols in that particular script.[22] gives a detailed account of amendment proposed in the code chart for Gujarati and the rendering rules in the Unicode version 3.0. The Unicode chart for Gujarati as the the latest version of the standard is given in appendix.

We use Unicode to generate text from the recognized glyphs. Mathematically, this can be looked at as a mapping which maps glyph code(s) to appropriate Unicode(s). As indicated by an 's' in the bracket, this mapping can

- map more than one glyph codes to a Unicode. For example, in case of consonants like ળ, ક્ષ, જ્ઞ this mapping maps more than one glyphs to one Unicode
- map one glyph code to one Unicode. e.g. ક,ખ,ઈ etc.
- map one glyph code to more than one Unicodes.

ઋ = ર + ◌ૃ + ઽ ; ક્ષ = ક + ◌્ + ષ ; શ્ર = શ + ◌્ + ર

In this step glyph clusters generated as per the process mentioned earlier are analyzed and appropriate Unicode(s) are replaced.

7.3 Special Cases

The result of text generation is shown in the next chapter showing the result of complete OCR system. Here, it is important to note that glyph ા can result

from various consonants, independent vowels and dependent vowel modifiers and needs to be handled carefully. More importantly, this glyph can be starting glyph when it has resulted from ર and decision about its glyph cluster becomes challenging when it is preceded by a consonant or conjunct having this glyph. In that case, its glyph cluster can not be determined by its own location. We have to analyze the glyph code of the glyphs before and above it. Hence we have not shown that in the finite state machine based model shown earlier to make glyph clusters.

Further, it may be recalled that we are not distinguishing between /pa/ (૫) and Gujarati numeral 5 (૫) at the classification level and hence, we always substitute Unicode for consonant /pa/ in such cases. For the same reason we substitute Unicode for /ra/ when we find glyph code corresponding to the fused class representing /ra/ and numeral 2. It is important it is also possible that due to various reasons like merges, split or noise we may get an unexpected connected component or a glyph may be missing altogether. In such cases, it may not be impossible to replace appropriate Unicode string.

7.4 Conclusions

The process of text generation can be modeled with a finite state machine. Here we have presented a model for using finite state machine for modeling one of the two subtasks of text generation viz. glyph cluster creation.

It is documented that taking decision about the glyph cluster for ા is somewhat complex than other cases.It is also possible that some of the glyph clusters may have insufficient glyphs to generate the text or sometimes additional glyphs.