ON AN APPLICATION OF L^6 TO THE THEORY OF AUTOMATA

By

Seiiti HUZINO

(Received Sept. 26, 1974)

1. Introduction.

In [1] the author has shown that it is possible to apply a list processing technique to problems in the theory of automata by presenting some simple examples and to execute smoothly the technique by using list processing language LISP and constructing a recursive program to a problem in the field of automata theory. The method can also be applied to another problems in automata theory with the same principle. However, there is a weak point in this method, brought from the restriction that we use LISP as a list processing language. Though LISP has remarkable features for recursive programming, using LISP to solve problems in automata theory is forming a serious restriction in treating such structure as in automaton, that is, the restriction is that LISP only uses binary tree structures, one of the simplest structures, as its data structures. Accordingly the users of LISP language have firstly to consider how to represent any given informations by binary tree structures. Happily we could technically treat the structure of automaton as we have shown in the paper [1]. But there remains a sense of being somewhat artificially in the view-point of automata theory. Are there any useful list processing languages, being able to construct more naturally information structure about automaton. If not, we must creat such a new language before we proceed further investigations. We have studied various list processing languages and other symbol-manipulating languages and have found that a language L^6 ([2]), created by K. C. Knowlton of Bell Telephone Laboratories, is one of the most suitable ones in applying to the theory. In this paper we shall start from, and establish, characterizations of structures by L^6 in treating problems in automata theory and explain its effectiveness through a simple example. We shall take the same problem, i.e., a recognition problem of strings by a finite automaton as in [1], in order to compare these techniques.

2. Representation of a finite automaton by L^6 .

Consider now a finite automaton with three states shown in figure 1 in order to simplify our explanations. It seems certainly difficult to imbed, and represent, such a structure into the memory in one of the nearest form to the original state

Fig. 1.

diagram by using LISP as a processing language. In L^6 , however, we can use another form, more complicated in structure and more easier in processing than in LISP. We can proceed as follows: Consider three blocks consisting of two words (See figure 2). We store into the first word in a block a number corresponding to a state s_i , e.g., a number i to s_i ($i=0, 1, 2$)

Fig. 2.

and divide the second word into two fields, left and right. We store in the left field of the word an address of the first word of a block representing the next state to which a present state is transited by an input a , and in the right field an address of the first word of a block representing the next state to which a present state is transited by an input b . Let us assume that the above three blocks have addresses shown in figure 2. Then, from the above rules of assigning, we have the following network (figure 3), corresponding to a given finite automaton. It certainly shows that figure 3 faithfully maps figure 1 into computer memory by aids of the above rules of assigning.

Now let us try to construct procedures for realizing such network by means of L^6 -technique. See the paper [2] for the details of L^6 -programming technique. By the way there is an important point necessary for representing structures of finite automata completely. It is how to point whether a state is either an initial state or a final state by using contents of blocks. This is executed by the following technique: First we use the first three bits in the first word of a block and in

Fig. 3.

it, by octal representations of numbers, we mark a symbol

1. 0_8 when a state is an initial state,
2. 1_8 when a state is a final state,

and

3. 2_8 otherwise,

where i_8 ($i=0, 1, 2$) means a decimal number i represented by octal number system. We preserve the remaining 33 bits as a field and mark names of states in it. For example, we use strings $S0$, $S1$ and $S2$ in place of states s_0 , s_1 and s_2 , respectively. If we assume that each symbol introduced above is represented by 6 bits, then only last 12 bits in the field are used in practice. The next word in a block is divided into two fields, left and right. The left field contains a pointer corresponding to a next state transited by an input a and the right field a pointer to a next state by an input b .

The following segment of a L^6 program exactly shows the above procedure:

```

AUTOMATON THEN (1, SS, 2, 1000)
              THEN (0, DA, 0, 2) (0, DB, 3, 35)
              THEN (1, DL, 0, 17) (1, DR, 18, 35)
              THEN (T, GT, 2) (TL, GT, 2) (TLL, GT, 2)
  
```

THEN (TR, P, TLL) (TLR, P, T) (TLLL, P, TLL)
 THEN (TLLR, P, TLL)
 THEN (TA, EO, 0) (TLA, EO, 1) (TLLA, EO, 2)
 THEN (TB, EH, S0) (TLB, EH, S1) (TLLB, EH, S2)

(Explanation of the program)

- line 1: preparation for free storage
- line 2: assigning names of fields in the first word as *A* and *B*, respectively
- line 3: assigning names of fields in the second word as *L* and *R*, respectively
- lines 4–6: construction for a state diagram of a given finite automaton
- line 7: assigning designated states
- line 8: naming states

The data structure shown in figure 4 is constructed by the above segment of a program, which shows that all information in figure 1 are completely imbedded into the memory.

Fig. 4.

3. Information processing for recognition of strings by a finite automaton.

If we input a string *aba* into the finite automaton shown in figure 1, then the automaton accepts the string. Let us present symbol-manipulating technique for doing such recognition of strings in a computer. It is done by preparing data structure for the string and by using two bugs which moves in the same manner as in actual recognition process of a finite automaton. For example we construct a data structure for the string *aba* as shown in figure 5. This is in fact constructed by the following segment of a program:

Fig. 5.

```

INPUT    THEN (0, DC, 0, 17) (0, DD, 18, 35)
          THEN (I, GT, 1) (IC, EH, A) (ID, E, 0)
          THEN (I, GT, 1, ID) (IC, EH, B)
          THEN (I, GT, 1, ID) (IC, EH, A)

```

(Explanation of the program)

line 1: assigning names of fields as *C* and *D*

lines 2–3: construction for a block

Next we shall prepare a segment of a program by which we test whether a given finite automaton recognizes a given string or not. First let us prepare two bugs *B* and *C*:

```

INITIAL  THEN (B, P, T) (C, P, I)

```

Two bugs *B* and *C* are in starting positions. Then, by using these bugs *B* and *C*, we get the following main program:

```

MAIN     IF (CD, E, 0) THEN LAST
          IF (CC, EH, A) THEN (B, P, BL) (C, P, CD) MAIN

```

```

                IF (CC, EH, B) THEN (B, P, BR) (C, P, CD) MAIN
LAST      IF (CC, EH, A) THEN (B, P, BL) TEST
                IF (CC, EH, B) THEN (B, P, BR)
TEST      IF (BA, EO, 1) THEN (Z, EH, ACCEPT) NEXT
                THEN (Z, EH, NOTACCEPT)
NEXT

```

The above program accepts the string aba and content of a pointer Z shows the string ACCEPT.

A complete program is now constructed by uniting all the above segments as follows:

```

AUTOMATON  THEN (1, SS, 2, 1000)
                THEN (0, DA, 0, 2) (0, DB, 3, 35)
                THEN (1, DL, 0, 17) (1, DR, 18, 35)
                THEN (T, GT, 2) (TL, GT, 2) (TLL, GT, 2)
                THEN (TR, P, TLL) (TLR, P, T) (TLLL, P, TLL)
                THEN (TLLR, P, TLL)
                THEN (TA, EO, 0) (TLA, EO, 1) (TLLA, EO, 2)
                THEN (TB, EH, SO) (TLB, EH, S1) (TLLB, EH, S2)
INPUT      THEN (0, DC, 0, 17) (0, DD, 18, 35)
                THEN (I, GT, 1) (IC, EH, A) (ID, E, 0)
                THEN (I, GT, 1, ID) (IC, EH, B)
                THEN (I, GT, 1, ID) (IC, EH, A)
INITIAL    THEN (B, P, T) (C, P, I)
MAIN       IF (CD, E, 0) THEN LAST
                IF (CC, EH, A) THEN (B, P, BL) (C, P, CD) MAIN

```

```

                                IF (CC, EH, B)    THEN (B, P, BR) (C, P, CD) MAIN
LAST                            IF (CC, EH, A)    THEN (B, P, BL)  TEST
                                IF (CC, EH, B)    THEN (B, P, BR)
TEST                            IF (BA, EO, 1)    THEN (Z, EH, ACCEPT) NEXT
                                THEN (Z, EH, NOTACCEPT)
NEXT

```

4. Conclusion.

We have shown in this paper that L^6 -technique is more superior than LISP technique in this case. We can easily generalize the above technique in the case when a finite automaton with m states ($m < 3$) over an alphabet $\Sigma = \{a_1, a_2, \dots, a_n\}$ is given. In this case we had better to prepare m blocks consisting of $\left\lceil \frac{n}{2} \right\rceil$ words. And also the above processings are extended to the cases for recognition problems of tree automaton by slightly changing data structures for which L_6 has much freedom of selection.

References

- [1] S. HUZINO, *On an application of list processing techniques to a recognition problem*, Mem. Fac. Sci., Kyushu Univ. Ser. A. **29** (1974)
- [2] K. C. KNOWLTON, *A programmer's description of L^6* , Comm. ACM., **9** (1966), 616–625.