ON AN APPLICATION OF LIST PROCESSING TECHNIQUES TO A RECOGNITION PROBLEM

By

Seiiti HUZINO

(Received Sept. 26, 1974)

1. Introduction

List processing techniques are now one of central techniques in computer science, in particular, in the field of artificial intelligence, e.g. theorem-proving by computer ([1], [2]). The object of the paper is to extend applicability of list processing techniques to the field of automata theory. The field contains various problems which seem to need proper aids of the techniques. Now we shall show a simple example which presents an evidence that list processing techniques can be used effectively. The problem is one of recognition of strings by a given finite automaton which seems to be a representative problem for showing usefulness of the techniques to that field. Before we enter into our main problem, we shall introduce and prepare the concept of a finite automaton and list processing techniques.

2. Finite automaton and its list representation

A finite automaton $\mathfrak{A}$ over an alphabet Σ is defined by a system $\mathfrak{A}=(S, M, s_0, F)$, where S is a non-empty finite set called a state set, M is a function from $S \times \Sigma$ into S , called a state-transition function, s_0 is an element in S , an initial state of the system, and F is a subset in S , called a final state set. The function M can be extended naturally to a function, written by M^* , from $S \times \Sigma^*$ into S , that is, M^* is defined recursively as follows:

$$M^*(s, \lambda) = s \quad (s \in S),$$

$$M^*(s, wa) = M(M^*(s, w), a) \quad (s \in S, w \in \Sigma^*, a \in \Sigma),$$

where λ is an empty string (of length zero). The function M^* divides the set of

all strings over Σ into two sets, i.e., a set of string w which satisfies the statement that $M^*(s_0, w)$ belongs to F , and its complement. The set $\{w | w \in \Sigma^* \text{ and } M^*(s_0, w) \in F\}$ is uniquely determined by a given finite automaton, so is called a set of strings recognized by a finite automaton $\mathfrak{A}$ and is written by $T(\mathfrak{A})$. The problem whether a given string w belongs to $T(\mathfrak{A})$ for a finite automaton $\mathfrak{A}$ or not is decidable. In the paper we shall concretely construct an algorithm of the recognition problem by using list processing techniques. As a representative of list processing language here we shall take LISP, in particular LISP 1.5, originated by J. McCarthy in order to study problems in the field of artificial intelligence ([2]).

First of all we need a list representation of a given finite automaton, which shows all mechanisms of the system. Let Σ be an alphabet $\{a_1, a_2, \dots, a_n\}$ and S a state set $\{s_0, s_1, \dots, s_m\}$. We use atomic symbols in LISP $A1, A2, \dots, An$, and $S0, S1, \dots, Sm$ as representations of elements in Σ and S , respectively. Let d be a list representation of both M and F which is defined as a binary tree structure $d = (d_0 d_1 \dots d_m)$, where d_i ($i=0, 1, 2, \dots, m$) is written as follows:

$$d_i = ((Si \cdot b) (A1 \cdot \overline{M(s_i, a_1)}) (A2 \cdot \overline{M(s_i, a_2)}) \dots (An \cdot \overline{M(s_i, a_n)}))$$

Here b is taken from one of atomic symbols ACCEPT or NOTACCEPT according to a condition $s_i \in F$, and $\overline{M(s_i, a_j)}$ ($i=0, 1, 2, \dots, m, j=1, 2, \dots, n$) shows a representation of state $M(s_i, a_j)$. Clearly we can show that the list d contains informations of both the mechanism of the system and the designation of final states F of the system. The couple $(Si \cdot b)$ is determined by the information F . And the sequence of couples $(A1 \cdot \overline{M(s_i, a_1)}) \dots (An \cdot \overline{M(s_i, a_n)})$ faithfully represents the behavior of the system at the present state s_i , so we can use the list d as information of a given finite automaton.

3. Recognition of strings by a given finite automaton

We shall prove the following theorem.

THEOREM. *There exists a LISP function recognize $[d; \bar{s}; \bar{w}]$ such that for any finite automaton $\mathfrak{A} = (S, M, s_0, F)$*

$$\text{recognize } [d; S0; \bar{w}] = \begin{cases} \text{ACCEPT} & \text{if } M^*(s_0, w) \in F, \\ \text{NOTACCEPT} & \text{if } M^*(s_0, w) \notin F, \end{cases}$$

where d is the above defined list representation of the automaton $\mathfrak{A}$, $w (= a_{i_1} a_{i_2} \dots$

a_{i_k}) is any string in Σ^* and $\bar{w}$ is a list $(\bar{a}_{i_1} \bar{a}_{i_2} \cdots \bar{a}_{i_k})$, the symbols $\bar{a}_{i_j}$ being atomic symbols of a_{i_j} , respectively. If w is an empty string, then we set $\bar{w} = \text{NIL}$.

PROOF. We recursively construct the above mentioned function *recognize* $[d; \bar{s}; \bar{w}]$. Let us prepare a function *findstate* $[e; \bar{a}]$ whose operation is to find a next state obtained by applying the function M when present input symbol is a . Here e is one of forms $((A1 \cdot \overline{M(s_i, a_1)}) \cdots (An \cdot \overline{M(s_i, a_n)}))$ ($i=0, 1, 2, \dots, m$). The function is written as follows:

$$\begin{aligned} & \text{findstate } [e; \bar{a}] \\ &= [\text{equal } [\text{caar}[e]; \bar{a}] \longrightarrow \text{cdar}[e]; \\ & \quad T \longrightarrow \text{findstate } [\text{cdr}[e]; \bar{a}]] \end{aligned}$$

Accordingly *findstate* $[\text{cdar}[d]; \bar{a}]$ is used for constructing a function *nextstate* $[d; \bar{s}; \bar{a}]$ the role of which is to find a state $M(s, a)$. Notice that d represents the mechanism of the system $\mathfrak{U}$ and *cdar* $[d]$ gives the first list of the form e . The function *nextstate* $[d; \bar{s}; \bar{a}]$ is easily constructed as follows:

$$\begin{aligned} & \text{nextstate } [d; \bar{s}; \bar{a}] \\ &= [\text{equal } [\text{caaar}[d]; \bar{s}] \longrightarrow \text{findstate } [\text{cdar}[d]; \bar{a}]; \\ & \quad T \longrightarrow \text{nextstate } [\text{cdr}[d]; \bar{s}; \bar{a}]]. \end{aligned}$$

Next we shall add a function *final* $[d; \bar{s}]$ whose operation is to give ACCEPT when $s \in F$ and NOTACCEPT when $s \notin F$. The function is recursively constructed:

$$\begin{aligned} & \text{final } [d; \bar{s}] \\ &= [\text{equal } [\text{caaar}[d]; \bar{s}] \longrightarrow \text{cdaar}[d]; \\ & \quad T \longrightarrow \text{final } [\text{cdr}[d]; \bar{s}]], \end{aligned}$$

because *cdaar* $[d]$ shows one of symbols ACCEPT or NOTACCEPT according to the condition $s \in F$. Finally, by combining these functions, we get the desired function:

$$\begin{aligned} & \text{recognize } [d; \bar{s}; \bar{w}] \\ &= [\text{equal } [\bar{w}; \text{NIL}] \longrightarrow \text{final } [d; \bar{s}]; \\ & \quad T \longrightarrow \text{recognize } [d; \text{nextstate } [d; \bar{s}; \text{car}[\bar{w}]]; \text{cdr}[\bar{w}]]] \end{aligned}$$

The theorem is proved.

4. An example

Let us consider the following finite automaton (Fig. 1). Here $\Sigma = \{a, b\}$, $S = \{s_0, s_1, s_2\}$, $F = \{s_1\}$ and M is represented as the matrix (Fig. 2).

Fig. 1.

M	a	b
s_0	s_1	s_2
s_1	s_2	s_0
s_2	s_2	s_2

Fig. 2.

The list representation of the automaton is a tree structure as shown in Fig. 3.

d :

Fig. 3.

The string $w = aba$ is accepted by the automaton as is clearly seen. Let us use our *recognize*-function as a list processing technique of the recognition problem. Then $\bar{w}$ is a list (ABA) . We calculate as follows:

$$\begin{aligned}
 & \text{recognize } [d; S0; (ABA)] \\
 &= [\text{equal } [(ABA); NIL] \longrightarrow \text{final } [d; S0]; \\
 & \quad T \longrightarrow \text{recognize } [d; \text{nextstate } [d; S0; A]; (BA)] \\
 &= \text{recognize } [d; \text{nextstate } [d; S0; A]; (BA)]
 \end{aligned}$$

Now

$$\begin{aligned}
 & \text{nextstate } [d; S0; A] \\
 &= [\text{equal } [S0; S0] \longrightarrow \text{findstate } [((A \cdot S1) (B \cdot S2)); A]; \\
 &\quad T \longrightarrow \text{nextstate } [\text{cdr}[d]; S0; A]] \\
 &= \text{findstate } [((A \cdot S1) (B \cdot S2)); A] \\
 &\text{findstate } [((A \cdot S1) (B \cdot S2)); A] \\
 &= [\text{equal } [A; A] \longrightarrow S1; \\
 &\quad T \longrightarrow \text{findstate } [\text{cdr}[((A \cdot S1) (B \cdot S2))]; A]] \\
 &= S1
 \end{aligned}$$

Hence

$$\text{nextstate } [d; S0; A] = S1$$

So we get

$$\begin{aligned}
 & \text{recognize } [d; S0; (ABA)] \\
 &= \text{recognize } [d; S1; (BA)]
 \end{aligned}$$

By continuing these processes, we obtain the result:

$$\begin{aligned}
 & \text{recognize } [d; S0; (ABA)] \\
 &= \text{recognize } [d; S1; (BA)] \\
 &= \text{recognize } [d; S0; (A)] \\
 &= \text{recognize } [d; S1; NIL] \\
 &= \text{final } [d; S1] \\
 &= \text{ACCEPT.}
 \end{aligned}$$

5. Conclusion

In this paper we have only indicated usefulness of list processing techniques by giving a simple problem in automata theory. These techniques, however, can

be used for solving various problems in the field of automata theory with the same principle. And we have used LISP as a list processing language throughout the paper. But it seems to the author that L^6 is more better for studying automata theory than using LISP, since the language L^6 has an ability for representing complicated data structure more directly than by LISP ([3]). We shall consider about this point in another paper ([4]).

References

- [1] S. HUZINO, *List processing and linguistic analysis* (Japanese), Asakura Pub. Co., 1970, pp. 192.
- [2] J. MCCARTHY et al., *LISP 1.5 Programmer's Manual*, The M.I.T. Press, 1962, pp. 99.
- [3] K. C. KNOWLTON, *A programmer's description of L^6* , Comm. ACM., **9** (1966), 616-625.
- [4] S. HUZINO, *On an application of L^6 to the theory of automata* (to appear).