An implementation of algorithms to define and manipulate finite automata and regular expressions

by

Adnan Abdul Kadir Maroof

A thesis submitted for the degree of Master of Science

Department of Computational Science

University of St.Andrews

St.Andrews

September 1984

Declaration

I declare that this thesis has been composed by myself and that the work that it describes has been done by myself. The work has not been submitted in any previous application for a higher degree. The research has been performed since my admission as a research student. October 1982 for the degree of Master of Science.

Adnan Abdul Kadir Maroof

I hereby declare that the conditions of the Ordinance and Regulations for the degree of Master of science (M.Sc.) at the University of St.Andrews have been fulfilled by the candidate, Adnan Abdul Kadir Maroof.

Professor A.J.Cole

Acknowledgements

I wish to thank Professor A.J.Cole for his continuous guidance and help in preparing this thesis and for his kindness at difficult times, and to the Iraqi government for supporting me financially during my stay in St.Andrews.

Also I thank my wife Jenan Abdul Wahid for her support during this work and for Omar who was born during the final year.

Finally I wish to dedicate this work to my mother and my late father Abdul Kadir Maroof.

Contents

Chapter 1

Finite automata

Introduction

The main purpose of the work described in this chapter is to enable the user to define and modify deterministic and non-deterministic finite automata .The first step is to allow the user to define states and transitions for both deterministic and non-deterministic finite state automata and to subsequently modify them . It is possible at this stage to systematically rename final and other states .

Functions are provided to convert non-deterministic finite automata to deterministic finite automata and also to trim and minimize automata . The minimization process produces a minimum state automaton . We will abbreviate the expressions **deterministic finite automata** and **non deterministic finite automata** to DFA and NDFA respectively .

1.1 Deterministic finite automata (DFA)

A DFA consist of a finite set of states and a set of transitions or moves from state to state that occur on given input symbols.The input symbols are chosen from a given finite alphabet ; the DFA consists of an initial state and a set of final states which is a subset of the whole set of states of the DFA.

Formally a DFA [1] is denoted by a 5-tuple

$(Q, \Sigma, q0, F, \delta)$

where :-

(i) Q is the finite set of states.

(ii) Σ is the input alphabet.

(iii) q0 is the initial state.

(iv) F is a finite set of final states ; F is a subset of Q ; i.e : ($F \subseteq Q$).

(v) δ is the transition function or move of the DFA ; i.e ($\delta : Q \times \Sigma \rightarrow Q$).

Suppose the automata is in state q reading a string w following by an input symbol a ; it can go to one state i.e ($\delta(q,a) = p$); it cannot go to another state in addition to p by a.

To extend the function δ to map $Q \times \Sigma^* \rightarrow Q$. We let (Σ^*) denote the set of all finite-length strings of symbols from (Σ). Including in (Σ^*) is (ϵ), the empty string. We define formally

(1) $\delta(q , \epsilon) = q$

(2) For all x in (Σ^*) and a in (Σ),

$\hat{\delta}(q,xa) = \delta(\hat{\delta}(q,x),a)$.

The DFA is said to accept the string if there is a sequence of moves that leads from the initial state to one of the final states .

Suppose $\hat{\delta}$ is a sequence of moves ;

and q0 is the initial state ;

and w is a string in Σ^*

and

$\hat{\delta}(q0,w) \cap F \neq \phi$

where ϕ is the empty set .Then w is accepted by the DFA.

1.2 Non deterministic finite automata (NDFA)

The NDFA consists of a finite set of states and a finite set of moves either from a state to a given state or from a state to many states that occur on a given finite input symbol . The input symbols are chosen from a given finite alphabet .

Formally a NDFA [1] is denoted by a 5-tuple $(Q , \Sigma , q0 , F , \delta)$ where :-

$Q , \Sigma , q0 , F$ have the same meaning as for a DFA; δ is a transition function $Q \times \Sigma \rightarrow 2^{Q}$ i.e : $(\delta : Q \times \Sigma \rightarrow 2^{Q})$; where 2^{Q} is the power set of states Q . δ may be a partial function.

Thus the automata in state q0 reading an input symbol a can go to many states.

For example $\delta (q0 , a) = \{ p , q , r , \ldots\ldots\ldots \}$.

Suppose $\hat{\delta}$ is a function defined such that if the automata is in state q reading the string w followed by an input symbol a ,the automata will be in state p iff one possible state the automata can be in after reading w is b and from b it can go to p upon reading a.

Hence :-

$\hat{\delta} (q , wa) = \{ p \mid$ for some state b in $\hat{\delta} (q , w)$,

p is in δ (b , a) } .

1.3 Data structures

The following three data structures are used

1.3.a Structures

1.3.b Linked lists

1.3.c Files

1.3.a Structures

The structures are as follows

(i) The structure named **box** defines a record consisting of a string named **name**, an integer named **number** and two pointers named **down** and **next** respectively.

That is **structure** box(**string** name;**int** number;**pntr** down,next).

(ii) The structure named **rec** defines a record consisting of a string named **node** and two pointers named **down.** and **right** respectively.

That is **structure** rec(**string** node;**pntr** down.,right).

(iii) The structure named **recl** defines a record consisting of a string named **state,** an integer named **number.** and a pointer named **right..i.e**

structure recl(**string** state;**int** number.;**pntr** right.).

(iv) The structure named **boxl** defines a record consisting of a string named **numberl** .

That is **structure** boxl(**string** numberl).

1.3.b Linked lists

In this work various procedures are used to create the linked list in which a description of the automata is

held . The main two procedures which create this list are as follows :-

(1) The procedure named **machine** links a list M of records each of which consists of the name of the state and two pointers, one pointing to the **right** to link all records sequentially and the other pointer named **down.** .

That is **structure** rec(**string** node;**pntr** down.,right).

The list E of records pointed at by **down.** each of which consists of the name of the state, an input number and two pointers **next** and **down** respectively. **next** refers to the next record in M and **down** refers to the next record in E. This will be discussed further in chapter 2.

(2) The procedure named **alt.node** links the list E of records. In addition to linking down E , each record points to the next state in M. The pointer **next** does this.i.e

structure box(**string** name;**int** number;**pntr** down,next).

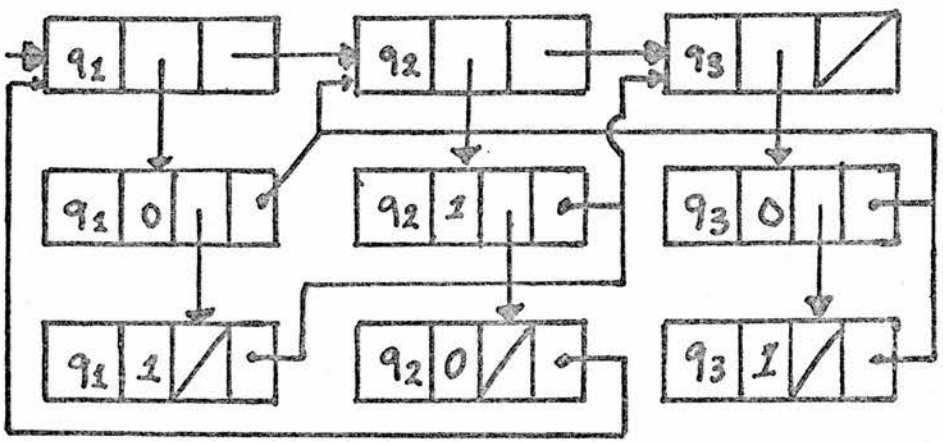
Example

Suppose the DFA consists of one non final state **q1** and two final states **q2** , **q3** where **q1** is the initial state of the automaton.

Suppose that **q1** goes to **q2** and **q3** by **0** and **1** respectively, **q2** goes to **q1** and **q3** by **0** and **1** respectively and **q3** goes to **q2** by two input numbers **0** and **1** respectively.

Then the states and the moves of this automaton are held in the structure as illustrated in **fig 1.1.**

fig 1.1



1.3.c Files

Files in this work are used to enable the user to perform operations and changes on both NDFA and DFA . Description of such automata can be saved and retrieved when required. For example , the system can rename and trim a DFA and the user has the option of saving the trim automata if he wishes (see 1.5 for the definition of a trim automata) .

1.4 Renamed-Automata

This is simply the original DFA, but with states renamed . The system changes the non final states to {a0 , a1 , a2 ... } and the final states to { f0 , f1 , f2 , ... } and the result will be a renamed-automata.

1.5 Trim-Automata

For any automaton, the trim automaton is obtained by removing all states which are not vertices in some successful path from the initial state to one of the final states. Hence by removing the inaccessible states from the automata, we obtain the trim automata.

The formal algorithm used to obtain the trim automata is as follows :

By definition [2] any automata A having any of the following properties are equivalent :-

(i) A is trim.

(ii) Every state in A is a vertex in some successful path.

(iii) Every path in A is a segment of some successful path.

It is obvious that for any automata A a trim automata A_t is obtained as follows :-

(a) Determine the paths which lead from the initial state to one of the final states.

(b) Find the states of each sub-path which is a segment in some successful path.

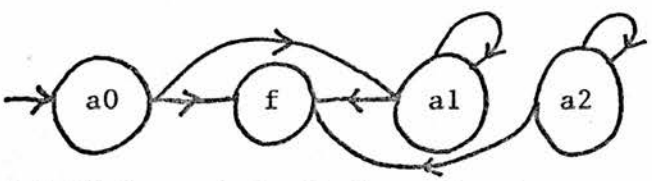
(c) Remove all states which do not appear in some successful path from the initial state to one of the final states.

The above parts of the algorithm will be more fully discussed in chapter 4.

Example :-

Suppose the automaton A consist of three non final states **a0 , a1 , a2** and one final state **f** where **a0** is the start state of **A .** Suppose that **a0** goes to **a1** and **f** by given input symbols , **a1** goes to **f** and itself by given input symbols , **a2** goes to **f** and itself by given input symbols ; then the successful path which leads from the initial state to one of the final states is **a0 , a1 , f** and the only states of A_t are **a0 , a1 , f .** a2 is not a state of A_t because it is not a vertex in any successful path .

Diagram



1.6 Minimum state finite automata

During the minimization process we have to start with the final set of states **i.e (f0 , f1 , f2 , ...)** and build a table of sets **(f0 , f1 , f2 , ...)** S^{-1} with $S \in \Sigma^*$; this can be done as follows :-

(a) Reverse all the moves of the trim automata ; if we have $\delta(q0,a) = q1$ the system will systematically reverse this transition to become $\delta(q1,a) = q0$.

(b) The system will start with the set of final states and build a table of sets **(f0 , f1 , f2 , ...)** S^{-1} . This is done recursively by calling the corresponding procedure (see chapter 5).

(c) The system checks the set of states in the table of sets (f0 , f1 , f2 , ...) S^{-1} to find which pairs of non final states are equivalent and never get separated in the table of sets (f0 , f1 , f2 , ...) S^{-1} ; equivalent pairs of states are merged.

(d) The system checks whether the set of final states appear together in the table of sets (f0 , f1 , f2 , ...)S^{-1}. If so the system will merge them.

(e) If d fails the system checks in the table of sets (f0 , f1 , f2 , ...) S^{-1} to find the pairs of states which are equivalent and never get separated in the table , i.e $f0 \in (f0 , f1)\ S^{-1} \Leftrightarrow f1 \in (f0 , f1)\ S^{-1}$; any such pairs of states are merged.

(f) If d and e fails the system checks in the table of (f0 , f1 , f2 , ...)S^{-1} and merges the set of final states which do not have a back pointer i.e $\delta(f0 , a) = f0$.

(g) If d and e and f fails the system will leave the set of final states without merging.

The whole algorithm is more fully discussed in chapter 5.

Example :-

Suppose the trim automaton is as follows :-

a0

1	a0
0	a1

a1

1	f0
0	a1

f0

1	f0
0	f1

f1

1	f0
0	f1

To minimize this automaton we have to start with the set of final states { f0 , f1 } and build a table of (f0 , f1)S^{-1} as follows :-

(1) In reversing the transitions of the automata the result is as follows :-

a0

1	a0

a1

0	a0
0	a1

f0

1	a1
1	f0
1	f1

f1

0	f0
0	f1

(2) By constructing a table of sets (f0 , f1)S^{-1} recursively the resulting automaton is as follows :-

f0,f1

0	f0,f1

	1	a1,f0,f1
a1,f0,f1		
	0	a0,a1,f0,f1
	1	a1,f0,f1
a0,a1,f0,f1		
	0	a0,a1,f0,f1
	1	a0,a1,f0,f1

Note that a sequence of states such as a0,a1,f0,f1 is now regarded as the name of a new state.

(3) We observe that states f0 , f1 never get separated in the above table ; whenever one appears on the right-hand side of an equation then so does the other. This means that $f0 \in (f0 , f1)S^{-1} \Leftrightarrow f1 \in (f0 , f1)S^{-1}$; it follows that the states f0 and f1 are equivalent and should be merged. An inspection of the table shows that this is the only such pair. Performing this merger gives the minimum state finite automata as follows:-

a0		
	1	a0
	0	a1
a1		
	1	f0,f1
	0	a1
f0,f1		
	1	f0,f1
	0	f0,f1

Chapter 2

Some data structures and algorithms associated with DFA and NDFA

Introduction

The following three data structures are used :

(1) structures

(2) linked list

(3) Files

(see 1.3 for the definition of the data structures).

The purpose of this chapter is to introduce the algorithms used for defining states and transitions for both DFA and NDFA . Functions are provided to modify deterministic and non-deterministic finite automata and also to find whether or not a particular string is acceptable. The structure in which the states and the moves of the automaton are held is illustrated in fig 2.3.

The algorithms

All of the algorithms , 2.1 to 2.7 relate to the transition $\delta(p,x) = q$ where p , q are states and x is an integer.

2.1 procedure machine (**string** s ; **pntr** r -> **pntr**)

This procedure has two parameters,namely a string s being the name of a state p of the automaton and a pointer r to point at a record containing s and two pointers one pointing to the right to link all records sequentially and the other initially pointing to **nil**. It produces a result of type pntr (pointer) to point at the whole list.

The idea of this procedure is to build a list M of records each of which consists of a string s and two pointers one pointing to the right to link all records sequentially and the other initially pointing to nil ,i.e :

(structure rec (string node ; pntr down.,right)).

M is used to hold the states of the automaton.

The procedure starts by testing to see if the pointer r is equal to nil . If it is we assign r to the structure named rec which consists of the string s and two pointers each of which is pointing to nil .

In order to insert a new record at the end of the list M, the end record of the list M must be found. This is accomplished by chaining through the list M until a record with a pointer pb pointing to nil is found and the new record is inserted there. This process continues until all records are linked. The procedure returns a pointer to the list M.

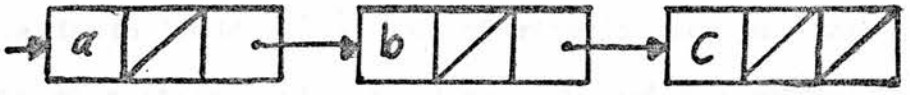
Example

Suppose the automaton A consists of three states a , b , c where a is the initial state of the automaton.

Suppose that a goes to itself and b by 0 , b goes to a , c by 0 , 1 respectively and c goes to itself and a by 0 , 1 respectively.

Then the states of this automaton are held in the list as illustrated in fig 2.1.

fig 2.1



2.2 procedure find.node (string s2 ; pntr r -> pntr)
This procedure has two parameters,namely a string s2 being the name of a state p of the automaton and a pointer r and produces a result of type pntr (pointer).
The idea of this procedure is to find a record containing s2 in the list pointed at by r .

This procedure initialises the pointer named p2 to r and declares a name done to be equal to false. p2 is tested to see if it is equal to nil or the required record has already been found.
If this is not so then the procedure loops, testing to see if each successive record is the required one.
The procedure either returns a pointer nil or a pointer to the required record.

2.3 procedure alt.node(string s1; int j1; pntr r)
This procedure has three parameters , namely a string s1

being the name of a state p of the automaton,an integer jl being an input number x and a pointer r to point at a record containing sl , jl and two pointers one pointing down to link all records in ascending order and the other initially pointing to nil .

The idea is to build a list E of records each of which consists of a string sl ,an integer jl and two pointers one pointing down to link all records in ascending order and the other initially pointing to nil , i.e :

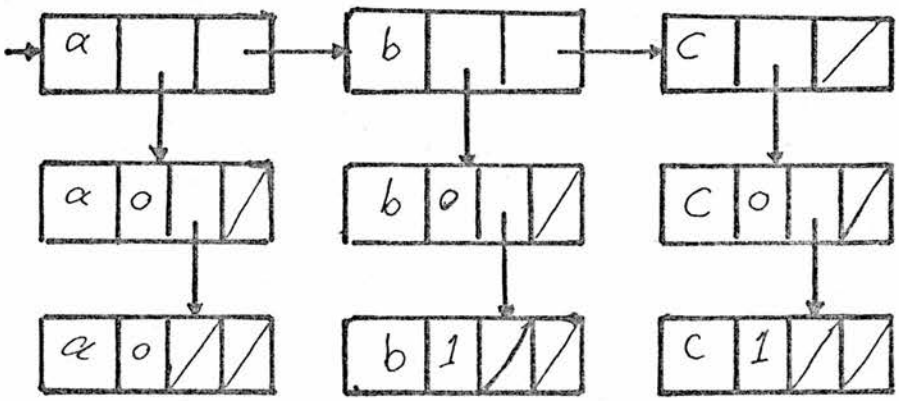
structure box(string name; int number; pntr down,next).

E is used to hold the states and an input number of the automaton.

The procedure starts by finding the required state in the list M and testing to see if the pointer p(down.) is equal to nil . If it is we assign p(down.) to the structure named box which consists of the string sl ,an integer jl and two pointers each of which is pointing to nil . In order to sort all records in ascending order, the record containing the smallest input number must be found. This is accomplished by chaining through the list E until a record with the smallest number is found, which is interchanged with the first record in the list. This interchange places the record with the smallest number in the first position of the list E . This process of searching for the record with the next smallest number and placing it in its proper position continues until all records have been sorted in ascending order.

In the example the states of the automaton corresponding to given input symbols are held in the structure as illustrated in fig 2.2 .

fig 2.2



2.4 procedure find(string s3; int j; pntr r -> pntr)

This procedure has three parameters,namely a string s3 being the name of a state p of the automaton, an integer j being an input number x and a pointer r and produces a result of type pntr (pointer).

The idea of this procedure is to find a record containing s3 and j in the list pointed at by r .

The procedure starts by finding the required state in the list M. It then assigns a pointer named w to p3(down) and false to the boolean variable search.

w is tested to see if it is equal to nil or the required record has already been found. If this is not so then the procedure loops,testing to see if each successive record is

the required one. The procedure either returns a pointer **nil** or a pointer to the required record.

2.5 procedure linked(string s4; int j1; string s5; pntr r)

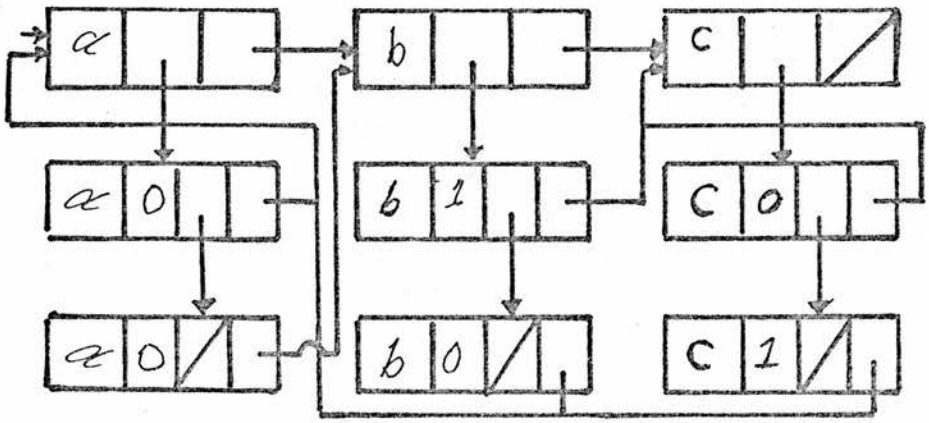
This procedure has four parameters namely,a string **s4** being the name **p** of a state of the automaton together with an integer **j1** being the corresponding input number **x** , **s5** consisting of the new state **q** of the automaton and a pointer **r** to point at a list **M** .

The idea of this procedure is to link each record in the list E with the required record in the list **M**. The pointer **next** does this.

The procedure starts by finding the required next state in **M**. In order to point to the next state in **M**, the record containing the required state, input number and pointer **next** pointing to **nil** must be found. This is accomplished by chaining through the list E until a record containing the required state, input number and a pointer **next** pointing to **nil** is found. The pointer **next** is set to the required next state in **M**.

In the example the moves of the automaton are held in the structure as illustrated in **fig 2.3.**

fig 2.3



2.6 procedure del.node (string s)

This procedure has one parameter namely, s which represent the name of a state p .

The idea is to delete the record containing the required state in the list M .

The procedure starts by checking to see if the state to be deleted is the first state of the list M, and if it is, then the second state in the list becomes the new first state. Otherwise it chains through the list until a record containing the required state is found and changes the link field of the predecessor record of the record containing the required state to point to its successor record.

2.7 procedure delete.line (string s ;int j,string sl; pntr r)

This procedure has four parameters namely,a string s being the name p of a state of the automaton together with an

integer **j** being the corresponding input number **x** , **sl** consisting of the new state **q** of the automaton and a pointer **r** to point at a list **M.**

The idea of this procedure is to delete the record containing **s** and **j** in the list **E** which points by a pointer named **next** to the record containing **sl** in the list **M.**

The procedure starts by finding the required next state in **M.** In order to point next to **nil,** the record containing the required state, input number,and pointer **next** pointing to the required next state in **M** must be found. This is accomplished by chaining through the list **E** until a record containing the required state, input number and pointer **next** pointing to the required next state in **M** is found and a pointer **next** is pointed at **nil.**

2.8 procedure linked.file

The idea is to read from a given file in which the description of the states of the automaton is held and create the list **M** of records.

The procedure successively reads the next state from the given file and produce a list **M** of records, (see **2.1** for the description of **machine).**

2.9 procedure automata.file(**pntr** r)

This procedure has one parameter, namely a pointer **r** to point at the whole structure in which the states and the moves of the automaton are held, (see **fig 2.3** for the description of the structure in which the states and the moves of the automaton are held).

The idea is to read from a given file in which the description of the states and the moves of the automaton are held and produces the whole structure with a pointer r pointing to it.

The procedure successively reads the moves from the given file and produces the whole structure of the automaton.

2.10 procedure symbol(**string** s)

This procedure has one parameter,namely a string s being the given input symbols. The idea is to create a list M1 of records each of which consists of one of the sequence of states which may occur in response to the given sequence of input symbols together with two pointers one pointing to the right to link all records sequentially and the other pointing to **nil.**

The procedure starts with the first record in the list M and the first input symbol in the given sequence of input symbols and successively searches down through each list from the record containing one of the sequence of states looking for the record containing such state and input symbol and inserts the successor state at the end of the list **M1.**

2.11 procedure acceptance (**string** s)

This procedure has one parameter, namely a string s being the given input symbols.

The idea is to find whether or not a string s is acceptable.

The procedure checks the last record in the list M1 to see

if it contains one of the final states. If it does, the given input string is acceptable. Otherwise the given input string is unacceptable.

Chapter 3

Conversion of
a non-deterministic finite automata
to a deterministic finite automata

Introduction

Suppose $N = (Q, \Sigma, \delta, q0, F)$ be a NDFA (see 1.1 and 1.2 for a formal definition of a DFA and NDFA). We define a DFA $D = (Q', \Sigma', \delta', q0', F')$ as follows.

The states of D are all the subsets of the set Q . That is elements of $Q' = 2^{Q}$ and called the power set of Q. F' is the set of all states in Q' containing a final state in N.

Suppose $\{qi, qj, \ldots, qt\}$ is a given set of states with $i<j<\ldots<t$ we refer to this set of states by the unique name $qi, qj, \ldots, qt$. Then a new set of names of states is defined recursively as follows.

(1) q0 is the name of $\{q0\}$.

(2) if q is a name and it is of the form

$q = qi, qj, \ldots, qt$ where $\{qi, qj, \ldots, qt\} \subseteq Q$

and

$i<j<\ldots<t$.

Then

(1) $\delta'(q0, a) = q$

iff

$\delta(q0, a) = \{q\}$

(2) if $p = pi, pj, \ldots, pt$

and $r = rs, rt, \ldots, ru$

where

$$i<j<t$$

and

$$s<t<u$$

then

$$\delta'(p,a) = r$$

iff

$$\delta(\{p\},a) = \{r\}$$

Note that the name of a state in the DFA corresponding to a given NDFA is built up as a concatenation of states of the NDFA separated by commas.

The description of an algorithm to find a DFA corresponding to a given NDFA is as follows.

(a) The system will start with the initial state of the NDFA , looking for all the moves of the automata which may occur from one input and concatenating all the corresponding successor states.

Example

Suppose $\delta(q0,0) = q0$

and $\delta(q0,0) = q1$

then $\delta(q0,0) = \{q0,q1\}$

and the new state is q0,q1 .

(b) The system continues separating each concatenation of states into its constituent states and looking for all the moves of the automata which may occur from one input and concatenating all successor states.

It is possible at this stage to systematically rename final

and other states (see 1.4 for the definition of renamed automata).

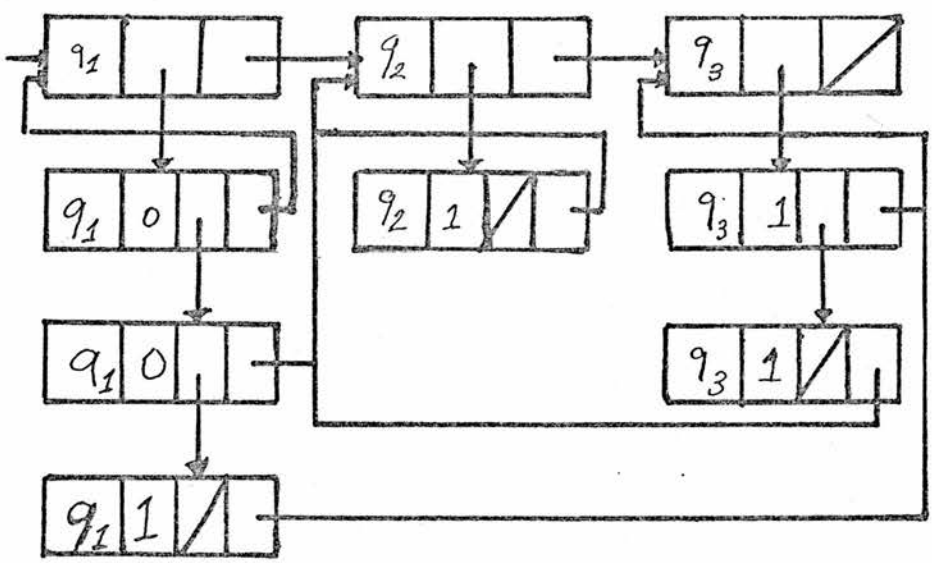
The algorithm

example

Suppose the NDFA consists of two non final states q1 , q3 and one final state q2 where q1 is the initial state of the automaton.

Suppose that q1 goes to itself, q2 and q3 by 0 , 0 and 1 respectively, q2 goes to itself by 1 and q3 goes to q2 and itself by 1. Then the states and the moves of this automaton are held in the structure as illustrated in fig 3.1.

fig 3.1



3.1 procedure concatenate(**string** s;**int** x;**pntr** r ->**string**)

This procedure has three parameters,namely a string **s** being the name of a state of the NDFA, an integer **x** being the input number and a pointer **r** to a list M of records in which the states of the NDFA are held (see 1.3.b for the description of the linked lists).

It produces a result of type **string** which is the concatenation of names of states.

The idea of this procedure is to look for all the moves of the automaton which may occur as a consequence of a particular one input and to concatenate all the corresponding successor states.

The procedure starts by finding the required state in M and searching down through the sublist from the record containing **s** looking for all the moves which may occur from the given input **x** and concatenating all the corresponding successor states.The procedure either returns an empty string or the concatenation of names of states.

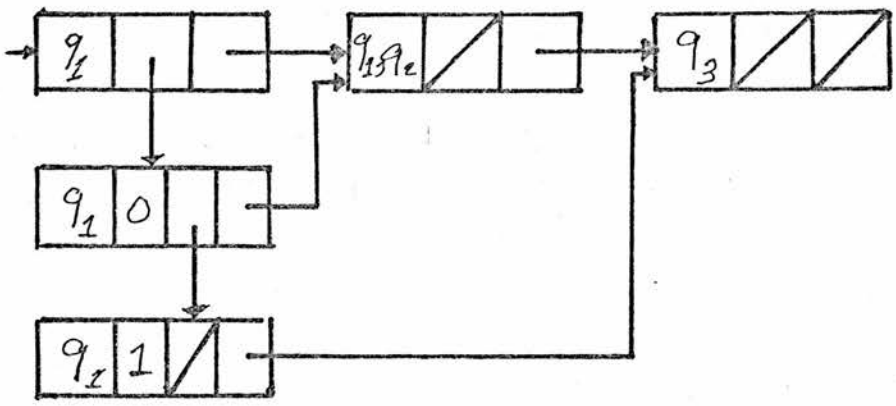
3.2 procedure finite

The idea of this procedure is to look for all the moves from the initial state of the automaton which may occur for one input and to concatenate all the corresponding successor states.

The procedure starts with the initial state ,in the example **q1** , and searches down through the sublist from the record containing the initial state looking for all the moves which may occur for one input and concatenates all the correspond-

ing successor states. In the example the procedure concatenates q1 with q2 and separates them by a comma.
The procedure produces a list D of records each of which consists of the concatenation of names of states and two pointers one pointing to the right to link all records sequentially and the other pointing down to a list of records each of which consists of the name of states together with the corresponding input numbers.
In the example dm points at the structure shown in fig 3.2.

fig 3.2



3.3 procedure d.f.a

The idea of this procedure is to separate each concatenation of states into its constituent states and to look for all the moves of the automaton which may occur from one input

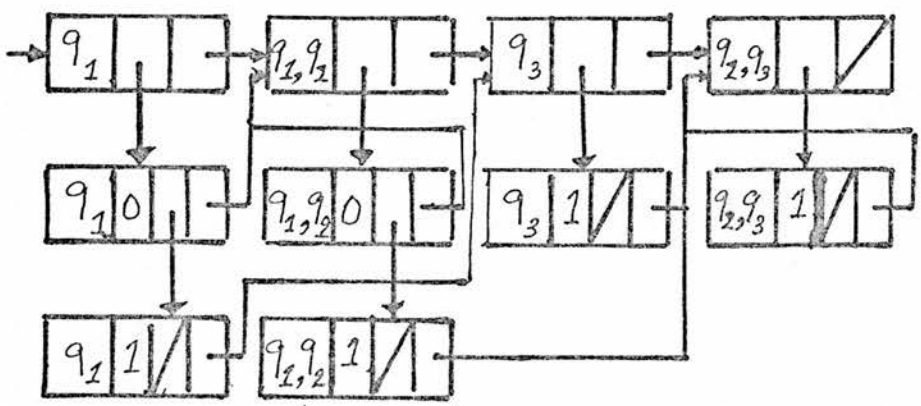
and finally concatenating all successor states.

The procedure starts with the second state in the list D, in the example q1,q2 separating this name into its constituent states and searches down through each list from the record containing one of the constituent states looking for all the moves which may occur from one input and concatenating all the corresponding successor states. This process continues successively through each list down the record containing one of the constituent states.

The procedure produces a list D1 of records each of which consists of the concatenation of names of states and two pointers one pointing to the right to link all records sequentially and the other pointer pointing down to a list of records each of which consists of the name of states together with the corresponding input numbers.

In the example the resulting automaton which is a DFA is held in the structure pointed at by a pointer named dm as illustrated in fig 3.3 .

fig 3.3



3.4 procedure f.automata

This procedure finds the finite set of final states of the DFA .

The procedure successively searches each concatenation of the names of states, looking to see if it contains any number of final states.

It produces a list **D2** of records each of which consists of the concatenation of names of states which contains a number of final states together with two pointers one pointing to the right to link all records sequentially and the other pointer pointing to **nil.** In the example a pointer named **dfinal** points at the list D2 as illustrated in **fig 3.4.**

fig 3.4

3.5 procedure s.automata

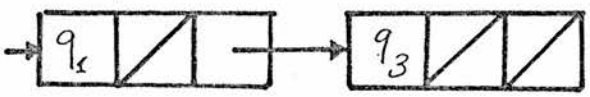
The idea of this procedure is to create a list **D3** of records each of which consists of the concatenation of names of non final states of the DFA together with two pointers one pointing to the right to link all records sequentially and the other pointer pointing to **nil.**

The procedure successively inserts all states which do not

appear in the list D2 at the end of the list D3.

In the example a pointer named **dstart** points at the list D3 as illustrated in **fig 3.5.**

fig 3.5



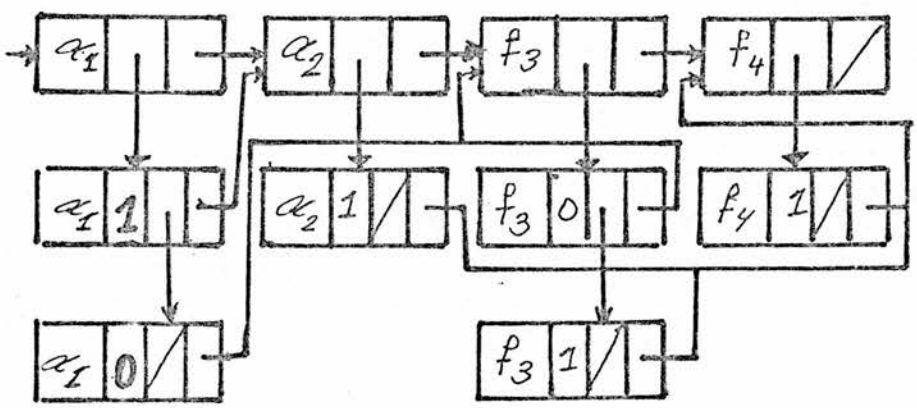
3.6 procedure renamed

The idea of this procedure is to change the names of the non final states to (**ai , aj , ... , ak**) respectively and the names of the set of all final states to (**fl , fm , ... , fn**) respectively where **i<j<...<k<l<m<...<n .**

The procedure starts with the initial state of the automaton and successively renames the names of the non final states and the names of the final states.

In the example the resulting automaton which is a renamed automaton is held in the structure pointed at by a pointer named **dm** as illustrated in **fig 3.6.**

fig 3.6



Chapter 4

Trim automata

Introduction

For any automaton **A** the corresponding trim automaton [2] is obtained by removing all states which are not vertices in some successful path from the initial state to one of the final states.
(see 1.5 for a formal definition of a trim automaton and a description of an algorithm to find a trim automaton).

The algorithm

4.1 procedure lengthen **(string** s **)**
This procedure has one parameter namely **s** which has a value being the concatenation of names of states.
The idea is to separate the string **s** into its constituent states and to create a list **C** of records each of which consists of one state and two pointers , one pointing to the right to link all records sequentially and the other pointing to **nil** .

Example

if **s** is exactly equal to **a1,a2** then the procedure produces a list **C** of two records. The first record consists of the string **a1** and two pointers one pointing to the right to link the first record with the second record and the other pointing to **nil.** The second record consists of the string **a2** and two pointers which in this case are both **nil** .

The procedure starts by initialising two integer variables **i , j** to **l .**

It then successively selects the separate states,in the example **a1** and **a2** , by looking for the next separator which is either a comma or the end of the string. The procedure produces a list **C** of records which is pointed at by a pointer named **second .**

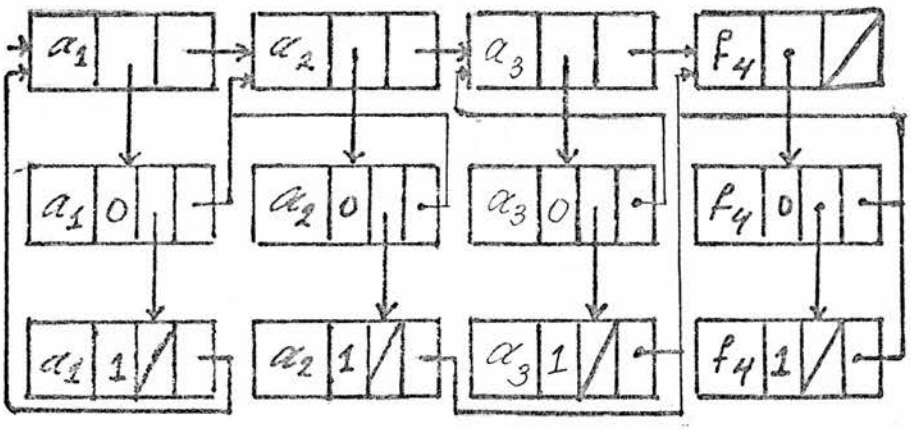
4.2 procedure trim

The idea of this procedure is to find all paths which lead from the initial state to one of the final states. This will be done by starting with the initial state of the automaton and concatenating the name of this state with the names of successive states until one of the final states is reached. The moves of the automata are held in the structure as illustrated in **fig 4.1 .**

Example

Suppose the automaton **A** consists of three non final states **a1 , a2 , a3** and one final state **f4** and the moves of this automaton are held in the structure as illustrated in **fig 4.1 .**

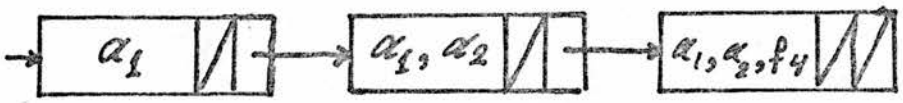
fig 4.1



The procedure produces a list **C1** of three records.The first record consists of the string **a1** and two pointers one pointing to the right to link the first record with the second record and the other pointing to **nil**
i.e (**structure** rec(**string** node ; **pntr** down.,right)).
The second record consists of the string **a1,a2** and two pointers one pointing to the right to link the second record with the third record and the other pointing to **nil.** The third record consists of the string **a1,a2,f4** and two pointers which in this case are both **nil.**

The procedure starts with the initial state, in the example **a1,** and searches down through the sublist from the record containing the initial state looking for all states in the records pointed at by **next** and concatenates the name of the initial state with the names of each one of those next states.In the example the procedure concatenates **a1** with **a2** and separates them by a comma and concatenates **a1** with **a2** and **f4** and separates them by commas. This process continues recursively through each list down the record containing one of the successive states. The procedure concatenates the name of each one of the successive states with the names of the previous states.
The procedure produces a list **C1** of records which is pointed at by a pointer named **tr** . In the example **tr** points at the list shown in **fig 4.2** .

fig 4.2



4.3 procedure find.f **(string** s **-> bool)**

This procedure has one parameter namely **s** which represents the concatenation of the names of states and produces a result of type boolean.

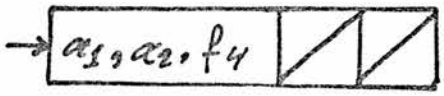
The procedure searches **s** , looking to see if it contains an **f** . It returns **true** if it does and **false** otherwise.

4.4 procedure trim.automata

The idea of this procedure is to create a list **C2** of records each of which consists of any string which contains any number of final states together with two pointers, one pointing to the right to link all records sequentially and the other pointing to **nil.**

The procedure successively checks each record in the list **C1** looking for strings containing any **f's** and produces a list **C2** of such records pointed at by a pointer named **head.** In the example **head** points at the list shown in **fig 4.3.**

fig 4.3



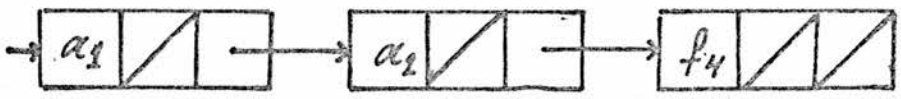
4.5 procedure trim.m

The idea of this procedure is to create a list C3 of records each of which consists of a string which is one of the states in some successful path from the initial state to one of the final states and two pointers one pointing to the right to link all records sequentially and the other pointing to nil ,that is :

structure rec (string node ; pntr down.,right).

The procedure successively separates each concatenation of states in the list C2 by calling lengthen (see 4.1 for the description of lengthen procedure) and produces a list C3 of records pointed at by a pointer named second. In the example second points at the list shown in fig 4.4.

fig 4.4

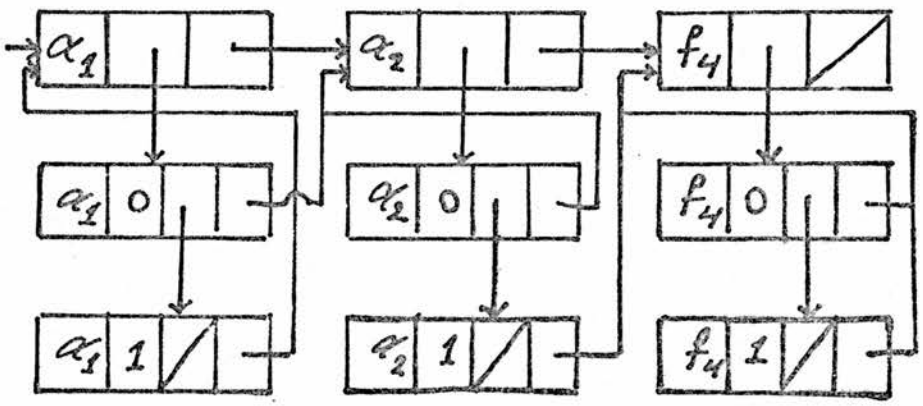


4.6 procedure trim.machine

The idea of this procedure is to remove all states in the list M which do not appear in the list C3 . The procedure successively removes all states which do not appear in the list C3.

In the example the resulting automaton which is a trim automaton is held in the structure pointed at by a pointer named dm as illustrated in fig 4.5 .

fig 4.5



Chapter 5

Minimum state finite automata

Introduction

For any DFA the corresponding minimum state finite automaton [2] is obtained by starting with the final set of states i.e (f0 , f1 ,...) and building a table of set $(f0 , f1 , \ldots) S^{-1}$ with $S \in \Sigma^*$ (see 1.6 for a description of an algorithm to find a minimum state finite automaton).

The algorithm

5.1 procedure reverse

The idea of this procedure is to reverse all the moves of the trim automaton.

It reverses the transition $\delta(p,x) = q$ to become $\delta(q,x) = p$ where p , q are states and x is an integer. It produces a list E1 of records each of which consists of the name q of a state of the automaton and two pointers one pointing to the right to link all records sequentially and the other pointer pointing down to another list of records each of which consists of the name q of a state of the automaton together with the corresponding input number x and two pointers one pointing to the next state p and the other pointing down to link all records in ascending order.

This procedure starts with the initial state of the automaton and searches down through the sublist from the record

containing the initial state. In order to reverse the moves of this state,all the successor states must be found.

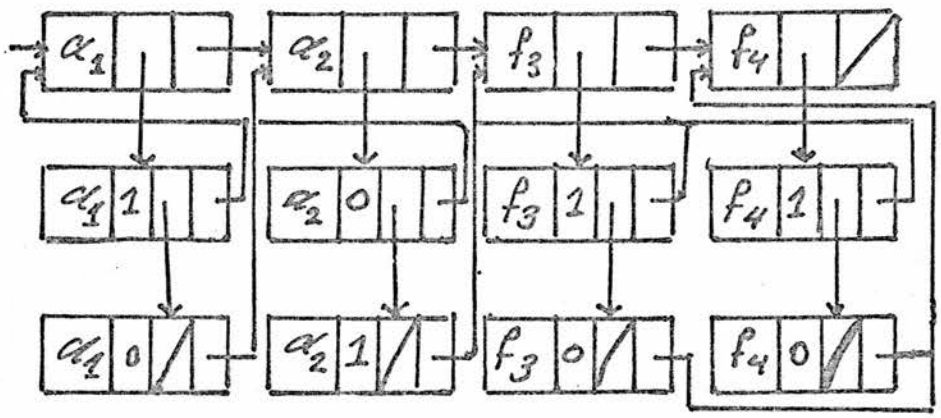
This is accomplished by chaining through the sublist until a pointer next pointing to the next state is found and the new move becomes $\delta(q,x) = p$

This process continues successively until all the moves of the automaton are reversed.

Example

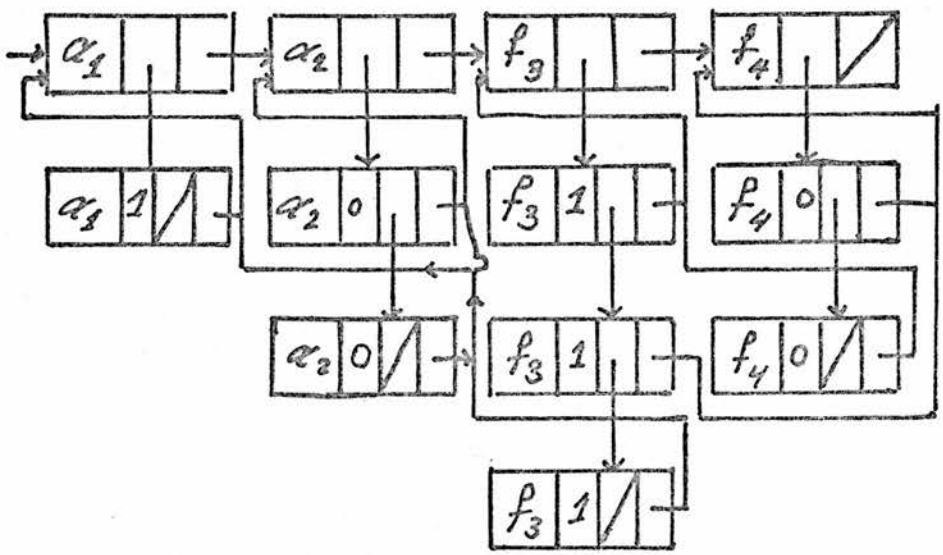
Suppose the trim automaton consists of two non final states **a1** , **a2** and two final states **f3** , **f4** where **a1** is the initial state of the automaton and the moves of this automaton are held in the structure pointed at by **dm** as illustrated in **fig 5.1.**

fig 5.1



Then by reversing all the moves of the automaton the resulting automaton is held in the structure pointed at by a pointer named **first** as illustrated in **fig 5.2.**

fig 5.2



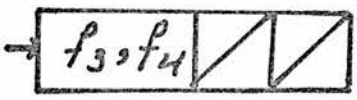
5.2 procedure reduce

This procedure concatenates the set of final states of the list E1 and produces a list E2 of one record which consists of the concatenation of the set of final states together with two pointers which are initially pointing to **nil.**

This procedure successively checks each record in the list E1 looking for strings containing any f's and concatenates the names of states in such records.

In the example a pointer named mn points at the list E2 as illustrated in **fig 5.3.**

fig 5.3



5.3 procedure reduction

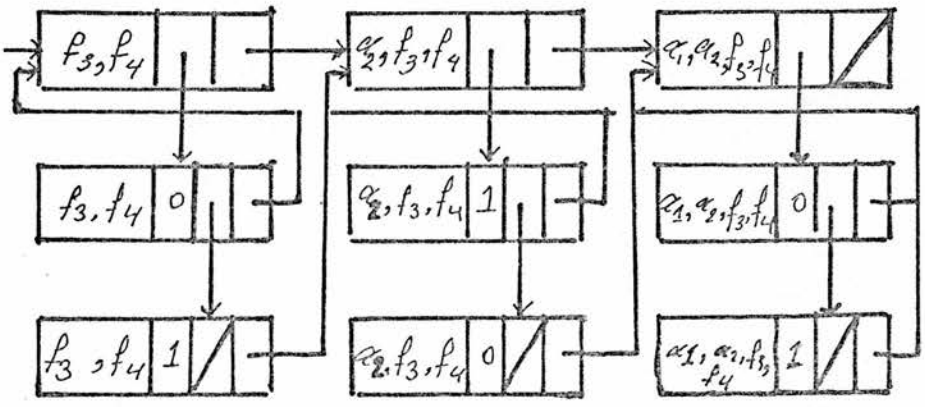
This procedure separates each concatenation of states into its constituent states and looks for all the moves of the automaton which may occur from one input and concatenates all successor states.

The procedure starts with the first state in the list E2 in the example f3,f4 separating this name into its constituent states and searches down through each list from the record containing one of the constituent states looking for all the moves which may occur from one input and concatenating all the corresponding successor states. This process continues recursively through each list down the record containing one of the constituent states.

The procedure produces a list E3 of records each of which consists of the concatenation of names of states and two pointers one pointing to the right to link all records sequentially and the other pointer pointing down to a list of records each of which consists of the name of states together with the corresponding input numbers.

In the example the resulting concatenations and moves are held in the structure shown in fig 5.4

fig 5.4



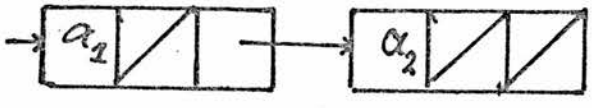
5.4 procedure min.m

This procedure creates a list E4 of records each of which consists of the non final states of the automaton and two pointers one pointing to the right to link all records sequentially and the other pointing to nil.

The procedure successively checks each record in the list E1 looking for strings which do not contain any f's and produces a list of such records.

In the example a pointer named third points at the list shown in fig 5.5.

fig 5.5



5.5 procedure minimum.machine

This procedure finds which pairs of states of the list E4 are equivalent and never get separated in the list E3.

The procedure successively checks each pair of states of the list E4 in the list E3 to find which pairs of states are equivalent and never get separated; equivalent pairs of states are concatenated and separated by commas.

In the example there are no such pairs of states.

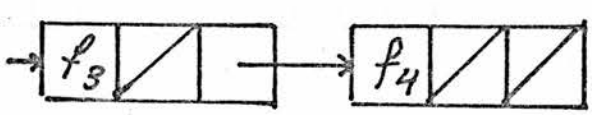
5.6 procedure min.final

The idea of this procedure is to create a list E5 of records each of which consists of the name of a final state of the automaton and two pointers one pointing to the right to link all records sequentially and the other pointing to **nil.**

The procedure successively checks each record in the list E1 looking for strings containing any f's and produces a list E5 of such records pointed at by a pointer named **start.**

In the example **start** points at the list shown in **fig 5.6.**

fig 5.6

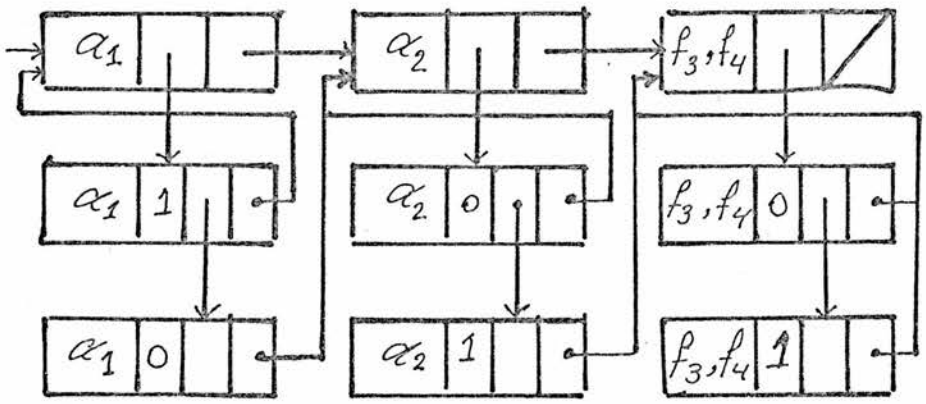


5.7 procedure m.machine

The idea of this procedure is to check whether the set of states of the list **E5** appears together in the list **E3.** The procedure successively checks each record in the list **E3** looking for strings containing the concatenation of the names of the whole set of final states. It concatenates them if such a string is found and it will leave them with out concatenation otherwise.

In the example in fig 5.1 the procedure concatenates f3 with f4 and separates them by a comma. Performing this concatenation gives the minimum state finite automaton as illustrated in **fig 5.7** .

fig 5.7



5.8 procedure m.automata

The idea of this procedure is to find the pairs of states of the list E5 which are equivalent and never get separated in the list E3.

The procedure successively checks each record in the list E3 looking for strings containing such pairs of states. It concatenates such pairs of states if it does and it will leave them without concatenation otherwise.

5.9 procedure back

This procedure checks each record in the list E5 to see if it contains a back pointer.

This is done successively by searching down through the sub-list from each record containing one of the final states looking for all the moves which may start and end with the same final state and produces a list of records each of which consists of such states and two pointers one pointing to the right to link all records sequentially and the other pointing to nil.

In the example there are no such pairs.

Chapter 6

Regular expressions

Introduction

The languages accepted by finite automata are called regular expressions [1],[3],[4].

Definition

Given two sets of words R and S from (Σ^*) where (Σ) is a finite set of symbols denote:

(1) $R + S = \{x | x \in R \text{ or } x \in S\}$

is the set theoretical union.

(2) $R.S = RS = \{xy | x \in R , y \in S\}$

is the multiplication(concatenation).

(3) $R^* = \epsilon + \{x | x$ is obtained by multiplying (concatenating) a finite number of words of R$\}$

Notice that R^* denotes the set of words obtained as the union of all words in R^i for $i = 0 , 1 , 2 , \ldots$, where $R^0 = \epsilon$ and

$$R^i = (\ldots((RR)R)\ldots)R$$
$$\underbrace{\qquad\qquad}_{i \text{ times}}$$

Therefore,

$$R^* = \epsilon + R + R^2 + \ldots$$

Example

let $\Sigma = \{a , b\}$

$R = \{a^2 , ab\}$

$S = \{aba , ab , ba\}$

then

$R + S = \{a^2 , ab , aba , ba\}$

RS = {aba , ab , aba , ababa , abab , aba}

R^* = {ϵ , a , ab , a , ab , aba , abab , a ,...}.

Let (Σ) be an alphabet. The regular expressions over (Σ) and the set that they denote are defined recursively as follows.

(1) ϕ is a regular expression and denotes the empty set.

(2) ϵ is a regular expression and denotes the empty string.

(3) For each a in (Σ), a is a regular expression and denotes the set {a}.

(4) If r and s are regular expressions denoting the language R and S respectively, then

(a) (r+s) is a regular expression denotes the set R U S.

(b) (rs) is a regular expression denotes the set RS.

(c) (r^*) is a regular expression denotes the set R^*.

Let a , b and c be regular expressions then the basic algebraic properties of the regular expressions are as follows.

1. a + b = b + a

2. $\phi^* = \epsilon$

3. a + (b + c) = (a + b) + c

4. a(bc) = (ab)c

5. a(b + c) = ab + ac

6. (a + b)c = ac + bc

7. $a\epsilon = \epsilon a = a$

8. $\phi a = a \phi = \phi$

9. $a^* = a + a^*$

10. $(a^*)^* = a^*$

11. $a + a = a$

12. $a + \phi = a$

Let R_{ij}^k denote the set of all strings that take the automaton from state qi to state qj without going through any state numbered higher than k. Since i or j may be greater than k and there is no state numbered greater than n, R_{ij}^n denotes all strings that take qi to qj.

We can define R_{ij}^k recursively :

$$R_{ij}^k = R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1},$$

$$R_{ij}^0 = \begin{cases} \{a \mid \delta(qi,a) = qj\} \text{ if } i \neq j, \\ \{a \mid \delta(qi,a) = qj\} \cup \{\epsilon\} \text{ if } i = j. \end{cases}$$

The definition of R_{ij}^k above means that the inputs that causes the automaton to go from qi to qj without passing through a state higher than qk are either

(1) in R_{ij}^{k-1} (that is, they never pass through a state higher than qk); or

(2) Composed of a string in R_{ik}^{k-1} (which takes the automaton to qk for the first time) followed by zero or more strings in R_{kk}^{k-1} (which takes the automaton from qk back to qk without passing through qk or a higher-numbered state)

followed by a string in R_{kj}^{k-1} (which takes the automaton from state qk to qj).

For i, j, and k, there exist a regular expression r_{ij}^{k} denoting the language R_{ij}^{k}.

Suppose k = 0

R_{ij}^{0} is a finite set of strings each of which is either (ϵ) or a single symbol.

Thus r_{ij}^{0} = a1 + a2 + ...+ ap if $i \neq j$

or

$$r_{ij}^{0} = a1 + a2 + ...+ ap + \epsilon \text{ if } i = j$$

where {a1 , a2 ,...,ap} is the set of all symbols a such that $\delta(qi,a) = qj$.

If there are no such a's then

ϕ if $i \neq j$ or

ϵ if i = j serves as r_{ij}^{0}.

For r_{ij}^{k} we may select the regular expression

$(r_{ik}^{k-1})(r_{kk}^{k-1*})(r_{kj}^{k-1}) + r_{ij}^{k-1}$,

The language $L(M) = \bigcup_{qj \text{ in } F} R_{1j}^{n}$

Since R_{1j}^{n} denotes the labels of all paths from q1 to qj.

Thus L(M) is denoted by the regular expression

$r_{1j1}^{n} + r_{1j2}^{n} + ... + r_{1jp}^{n}$,

where

F = {qj1, qj2, ..., qjp}

The purpose of this chapter is to provide tools to simplify regular expressions, formed by the above method.

Note that in forming

$r_{ij}^{k} = (r_{ik}^{k-1})(r_{kk}^{k-1*})(r_{kj}^{k-1}) + r_{ij}^{k-1}$,

it may be assumed that

(r_{ik}^{k-1}), $(r_{kk}^{k-1})^*$, (r_{kj}^{k-1}) and (r_{ij}^{k-1})

are all in simplified form.

It is therefore only necessary to be able to simplify concatenations and unions of reduced regular expressions.

During the reduction process, we assume that ri,rj is equivalent to rij.

Example

Suppose the automata A consists of the initial state q1 and two final states q2 , q3 respectively.

Suppose q1 goes to q2 and q3 with 0 , 1 respectively, q2 goes to q1 and q3 with 0 , 1 respectively and q3 goes to q2 with 0 , 1.

Then the system will simplify the regular expressions.

For example, the regular expression for r_{22}^{1} is given by

$$r_{22}^{1} = r_{21}^{0} (r_{11}^{0})^* (r_{12}^{0}) + r_{22}^{0}$$

$$= 0(\$)^*0+\$$$

where $ is the empty string.

The system will reduce the above expression to 00+$.

Similarly,

$$r_{13}^{2} = r_{12}^{1} (r_{22}^{1})^* r_{23}^{1} + r_{13}^{1}$$

$$= 0(00+\$)^*(1+01)+ 1.$$

The system will reduce the above expression to 0*1.

The algorithm

Some of the following algorithms relate to the transition $\delta(p,x) = q$ where p , q are states of the automaton and x is an integer. Others relate to R_{ij}^{k} which is the set of

all strings that take **qi** to **qj** without passing through a state higher than **qk.**

6.1 procedure regular

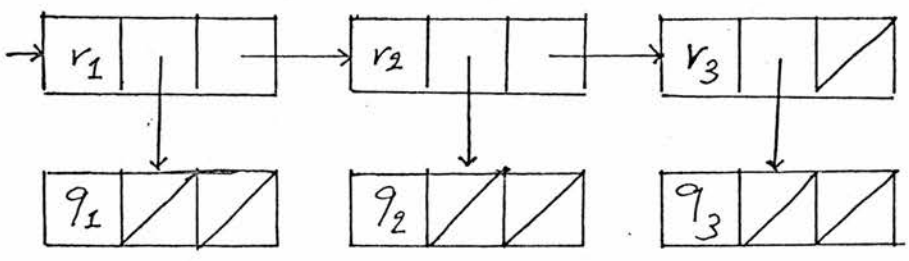
The idea is to change the names of the states of the automaton to **(ri , rj ,...rk)** respectively where **i < j <...< k** and to create a list **Gl** of records each of which consists of the new name of the state of the automaton and two pointers one pointing to the right to link all records sequentially and the other pointing down to another record which consists of the old name of the state of the automaton and two pointers which are both **nil** i.e

structure rec(**string** node; **pntr** down.,right)

The procedure starts with the initial state of the automaton, in the example **ql,** and successively renames the names of the successive states.

In the example the procedure produces a list **Gl** of three records shown in **fig 6.1.**

fig 6.1



6.2 procedure regular.ex(string s)

This procedure has one parameter namely a string s being the name of a state of the automaton to form a list of all possible pairs of states p,q.

The idea is to concatenate s with each name of the states of the automaton.

The procedure starts with the first record in the list G1 and successively concatenates each name of the states of the automaton with s and separates them by a comma.

It produces a list of records each of which consists of the concatenation of the names of the states together with two pointers one pointing to the right to link all records sequentially and the other initially pointing to nil. i.e:

structure rec(string node; pntr down.,right).

The resulting list is used for the calculation of the regular expression

$$r_{i,j}^{k} = (r_{i,k}^{k-1})(r_{k,k}^{k-1})^{*}(r_{k,j}^{k-1}) + (r_{i,j}^{k-1}).$$

6.3 procedure regular.exp

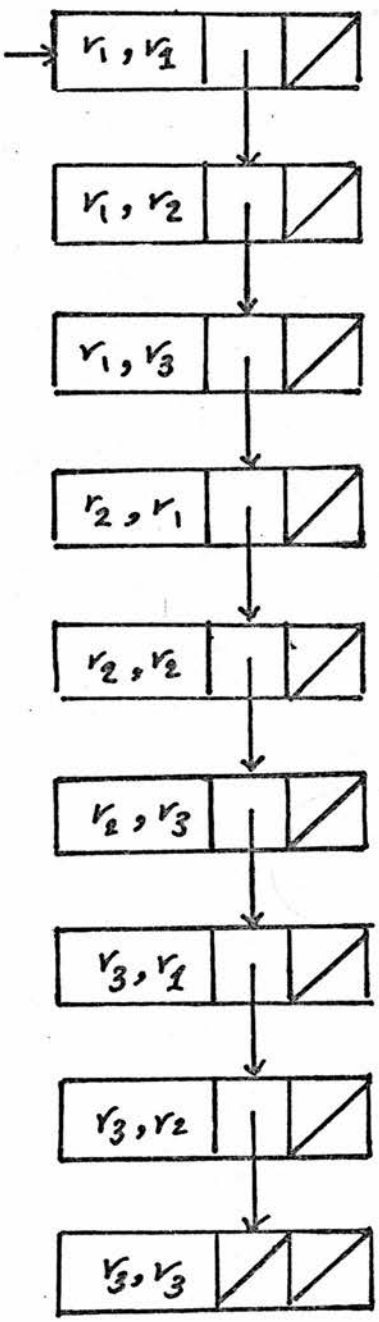
This procedure calls the procedure regular.ex recursively to form a list G2 of all possible pairs of states p,q.

Each element of this list is a structure of the form

structure rec(string node; pntr down.,right).

In the example, the procedure produces a list G2 of records shown in fig 6.2.

fig 6.2



6.4 procedure lang(string s,sl; pntr r -> string)

This procedure has three parameters namely, a string **s** being the name **p** of the state, a string **sl** being the name **q** of the state and a pointer **r** to point at a list **E** of records (see 2.3 for the description of the list **E**).

The idea is to search through the list **E** looking for records containing a pointer **next** pointing at the record containing **sl** and returns a string which is one of the following forms.

(1) The concatenation of the corresponding input numbers which are separated by plus signs to calculate

$r^{o}_{ij} = a1 + a2 + \ldots + ap$ if $i \neq j$

or

$r^{o}_{ij} = a1 + a2 + \ldots + ap + \in$ if $i = j$

where

{a1, a2, ..., ap}

is the set of all symbols **a**.

(2) an empty set if there are no such **a's** and

$i \neq j$.

(3) The empty string $ if there are no such **a's** and

$i = j$.

The procedure starts with the first record in the list **E** and successively checks each record in the list **E** looking for a pointer **next** pointing to the record containing **sl**. If such records are found, it concatenates the corresponding input numbers and separates them by plus signs. Otherwise it either returns an empty set if **s** is not equal to **sl** or

$ if s is equal to s1.

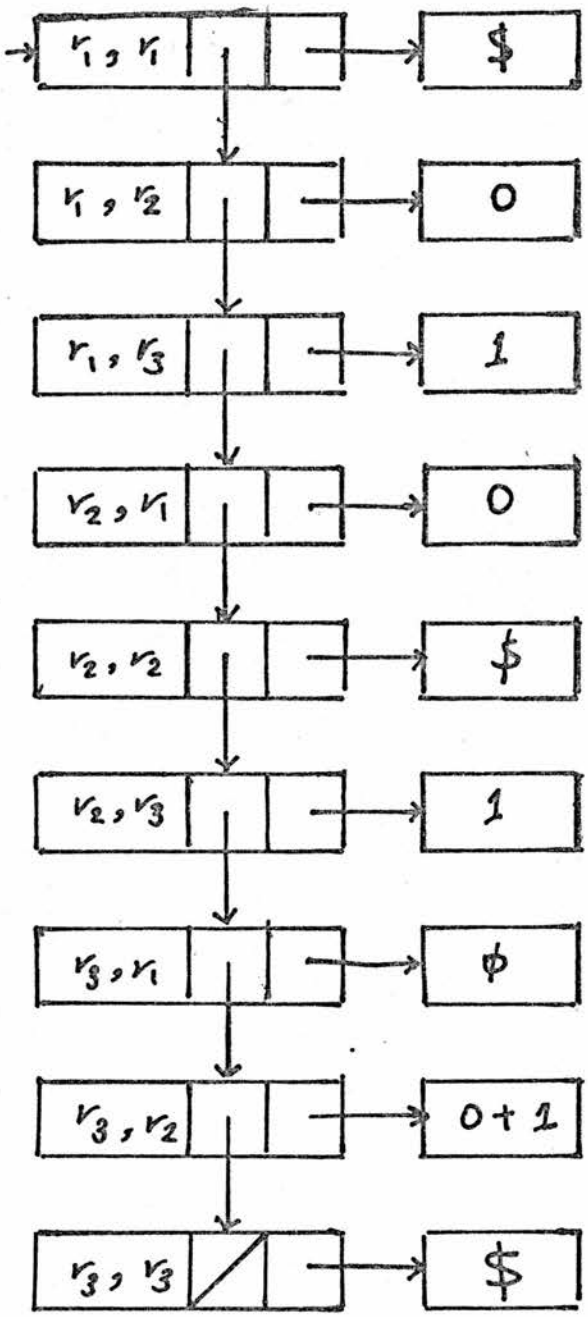
6.5 procedure reg.exp

This procedure calls the procedure lang recursively to form a record containing the resulting string which is a structure of the form

structure box1(string number1)

In the example the procedure produces a list G3 of records shown in fig 6.3.

fig 6.3



6.6 procedure reg.lang

The idea is to create a temporary list of records to hold the information of the original list for further reduction corresponding to the value of k, each of which consists of the concatenation of the names of the states of the automaton together with two pointers one pointing to the right to link all records sequentially and the other initially pointing to nil. i.e:

structure rec(string node; pntr down.,right)

The procedure starts with the first record in the list G3 and successively copies each concatenation of the names of the states from the list G3 and creates a temporary list of records with a pointer first pointing to it.

6.7 procedure reg.language(string s1,s2,s3,s4 -> string)

This procedure has four parameters. s1, s2, s3, s4 being the strings r_{ik}^{k-1}, r_{kk}^{k-1}, r_{kj}^{k-1}, r_{ij}^{k-1} respectively and returns a result of type string which is s1s2*s3+s4 .

The aim is to calculate $r_{ij}^{k} = (r_{ik}^{k-1})(r_{kk}^{k-1})^{*}(r_{kj}^{k-1})+r_{ij}^{k-1}$.

The procedure concatenates s1, s2, *, s3, +, s4 respectively.

6.8 procedure regular.expression(int x)

This procedure has one parameter namely x being the integer k.

The idea is to calculate r_{ij}^{k} and produce a temporary list of records each of which consists of the concatenation of the names of the states together with two pointers one

pointing to the right to link all records sequentially and the other pointer named **down.** pointing to the record containing the new possible string i.e

structure rec(**string** node;**pntr** down.,right).

The procedure starts with the first record in the list G3 and calls the procedure **reg.lang** recursively to form a record containing the new possible string which is a structure of the form

structure box1(**string** number1).

6.9 **procedure** proof(**int** x)

This procedure has one parameter namely **x** being an integer k.

The idea is to calculate the language $L(M) = \bigcup_{qj \in F} R1j^{n}$ where $R1j^{n}$ denotes the labels of all paths from ql to qj, L(M) is denoted by the regular expression

$r1j1^{n} + r1j2^{n} + \ldots + r1jp^{n}$,

n denotes the number of states of the automaton and

F = {qj1, qj2, ..., gjp}

The procedure recursively calculates the regular expression $r1j1^{n} + r1j2^{n} + \ldots + r1jp^{n}$ and produces the language L(M).

The following algorithms are used for the reduction of a string.

(1) **any.string** :- reduces **x + x to x**

(2) **tabulate** :- reduces (xx)*(y)+(y) to (xx)*(y)

(3) **proc** :- reduces (x+y)* + $ to (x+y)*

(4) table :- reduces a(x)*(x)+a to a(x)*

(5) stake :- reduces aa*b + b to a*b

(6) factor :- reduces (xx)*(yx+y) to (xx)* (x+$)y and to x*y.

(7) fact :- reduces a(x)*+a to a(x)*

(8) rame :- reduces (x)(x)*(x)+(x) to (x)(x)*

(9) ram :- reduces (x+$)(x)* to (x)*

(10) word :- reduces (x+$)* to x*

(11) chart :- reduces $*x to x

and x$ to x

Note that $ is the empty string.

As mentioned above it is only necessary to apply these reductions to the concatenation or union of two reduced regular expressions.

In the example the resulting regular expression is as follows.

	0
r1,r1	$
r1,r2	0
r1,r3	1
r2,r1	0
r2,r2	$
r2,r3	1

r3,r1	ϕ
r3,r2	0+1
r3,r3	$

1

r1,r1	$
r1,r2	0
r1,r3	1
r2,r1	0
r2,r2	(00+$)
r2,r3	(01+1)
r3,r1	ϕ
r3,r2	(0+1)
r3,r3	$

2

r1,r1	(00)*
r1,r2	0(00)*
r1,r3	0*1
r2,r1	(00)*0
r2,r2	(00)*
r2,r3	0*1
r3,r1	(0+1)(00)*0
r3,r2	(0+1)(00)*
r3,r3	(0+1)0*1+$

The language L(M) which is denoted by the regular expression is as follows :-

((0+1))(0(0+1)+1)*(0(0+1)+1)+((0+1))

This follows from the definition on page 62 and simplification.

In the example the minimum state finite automaton is as follows.

q1

0	q2,q3
1	q2,q3

q2,q3

0	q1
1	q2,q3

The regular expression of the minimum state finite automaton is as follows :-

0

r1,r1	$
r1,r2	0+1
r2,r1	0
r2,r2	1

1

r1,r1	$
r1,r2	(0+1)
r2,r1	0
r2,r2	0(0+1)+1

The language L(M) which is denoted by the regular expression is as follows :-

((0+1))(0(0+1)+1)*(0(0+1)+1)+((0+1))

The language L(M) which is denoted by the regular expression of the equivalent NDFA and trim automaton is the same as above. The language L(M) which is denoted by the regular expression of the minimum state finite automaton is usually simpler than others.

Chapter 7

Graphics

Introduction

The main purpose of the work described in this chapter is to enable the user to display an automaton on the screen.

Functions are provided to display any automaton on the screen and to interact with it.

7.1 Data structures

The following three data structures are used.

7.1.1 Structures

The structures are as follows

(i) The structure named **box** defines a record consisting of a string named state,an integer named number, a string named node and a pointer named next that is

structure box(**string** state; **int** number; **string** node; **pntr** next).

(ii) The structure named rec defines a record consisting of a string named vertex, two integers named xco and yco respectively and a pointer named right i.e:

structure rec(**string** vertex; **int** xco,yco; **pntr** right).

(iii) The structure named record defines a record consisting of a string named state1, an integer named number1 and three strings, namely state2, lane and back respectively and a pointer named nex i.e:

structure record(**string** state1; **int** number1; **string**

state2,lane,back; pntr nex).

7.1.2 Linked lists

In this work three procedures are used to create the linked list in which a description of the automaton and the corresponding coordinates numbers are held. See 7.2.1, 7.2.2, 7.2.3 for the description of those procedures).

7.1.3 Files

Files are used to retrieve the automaton when required. For example, the system can display any automaton and the user has the option of saving the corresponding coordinates numbers if he wishes.

7.2 The algorithms

Some of the following algorithms relate to the transition $\delta(p,x) = q$ where **p** , **q** are states and **x** is an integer and the others relate to the following notations, namely the letter **l** represents the straight line from one circle to another, the letter **a** represents an arc from one circle to another, the letter **u** represents an arc to be drawn above the circle, the letter **d** represents an arc to be drawn beneath the circle, and the letter **s** represents either a straight line or an arc from circle to circle.

7.2.1 procedure automata(**string** s1,s2; **int** j; **pntr** r **-> pntr**)

This procedure has four parameters, namely a string **s1** being the name of a state **p** of the automaton, a string **s2** being the name of a state **q** of the automaton, an integer

j being an input number x and a pointer r to point at a record containing s1 , s2 , j together with a pointer next to link all records sequentially and produces a result of type pntr (pointer) to point at the whole list.

The idea of this procedure is to build a list A of records each of which consists of s1 , s2 , j together with a pointer to link all records sequentially i.e:

structure box(**string** state; **int** number; **string** node; **pntr** next).

A is used to hold the states and the moves of the automaton.

The idea of inserting a new record at the end of the list A is the same as in the description of **machine** (see 2.1 for the description).

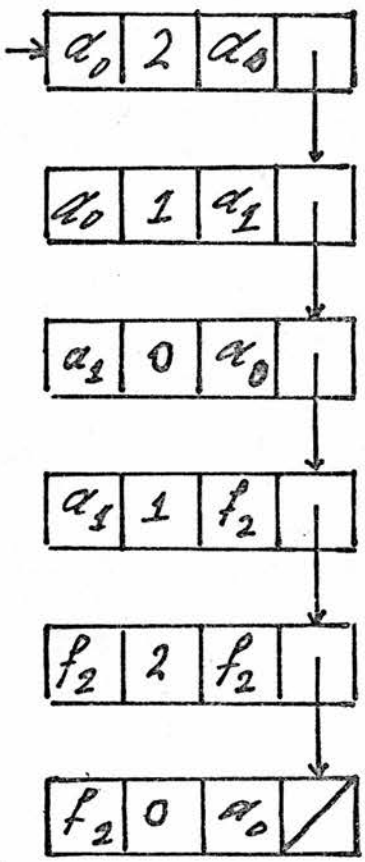
Example

Suppose the automaton A consists of two non final states a0 , a1 respectively and one final state f2 where a0 is the initial state of the automaton.

Suppose that a0 goes to itself and a1 by 2 , 1 respectively, a1 goes to a0 and f2 by 0 and 1 respectively and f2 goes to itself and a0 by 2 , 0 respectively.

Then the states and the moves of the automaton are held in the structure as illustrated in fig 7.1.

fig 7.1



7.2.2 procedure machine(**string** s; **int** i,j)

This procedure has three parameters, namely a string **s** being the name of a state **p** of the automaton and two integers **i** , **j** being the x-coordinate and y-coordinate of a state **p**.

This procedure builds a list **Al** of records each of which

consists of **s , i , j** together with one pointer pointing to the right to link all records in ascending order i.e:

structure rec(**string** vertex; **int** xco,yco; pntr right)

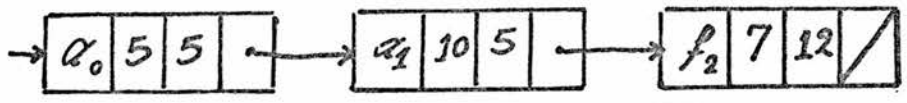
Al is used to hold the states of the automaton together with the corresponding coordinates.

The procedure starts by testing to see if the pointer **first** is equal to **nil.** If it is we assign **first** to the structure named rec which consists of the string **s,** two integers **i, j** respectively and one pointer which is in this case **nil.** In order to sort all records in ascending order, the record containing the smallest name of state of the automaton must be found. This is accomplished by chaining through the list **Al** until a record with the smallest name of a state is found, which is interchanged with the first record in the list. This interchange places the record with the smallest name of a state in the first position of the list **Al.** This process of searching for the record with the next smallest name of a state and placing it in its proper position continues until all the records have been sorted in ascending order.

In the example, suppose the given coordinates for **a0** is **(5,5),** the given coordinates for **a1** is **(10,5)** and the given coordinates of **f2** is **(7,12).**

Then the list in which the states of the automaton and the corresponding coordinates are held in the structure as illustrated in **fig 7.2**

fig 7.2



7.2.3 procedure collect(string s; int j; string s1,s2,s3)

This procedure has five parameters, namely a string s being a name of a state p of the automaton, an integer j being an input number x, a string s1 being a name of a state q of the automaton, a string s2 being the name of a line which is either l or a and a string s3 being the name of the back arc which is either u or d.

The idea of this procedure is to build a list A2 of records each of which consists of s , j , s1 , s2 , s3 together with a pointer nex which pointing to the next to link all records sequentially i.e:

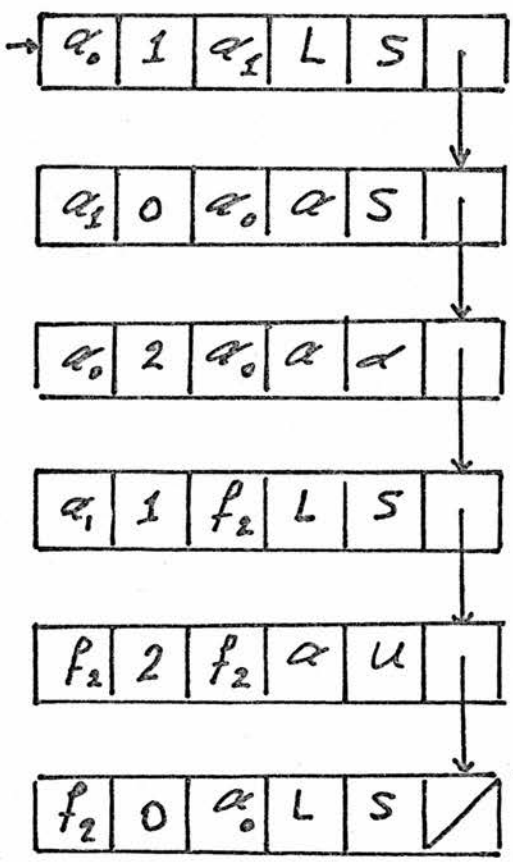
structure record(string state1; int number1; string state2,lane,back; pntr nex).

A2 is used to hold the states and the moves of the automaton together with the corresponding names of the lines which leads either from one circle to another circle or from a circle to itself.

The method of inserting a new record at the end of the list A2 is the same as in the description of machine (see 2.1 for the description).

In the example, suppose the given line from a0 to a1 is 1, the line from a1 to a0 is a, the line from a0 to itself is d the line from a1 to f2 is 1, the line from f2 to itself is u and the line from f2 to a0 is 1. Then the list in which this informations are held is illustrated in fig 7.3

fig 7.3



7.2.4 procedure non.blank (-> string)

This procedure reads the next non blank string from the given file and produces a result of type **string.**

The procedure successively reads all blanks and produces a string with no blanks preceding it.

7.2.5 procedure check

This procedure reads from a given file in which the description of the states and the moves of the automaton are held and creates a list **A** of records (see 7.2.1 for the description of automata).

The procedure successively reads the non blank string and produces a list **A** of records.

7.2.6 procedure c(**int** j,x -> **pic**)

This procedure has two parameters, namely an integer **j** being a given number to determine the value of the angle **theta** which is **j/32** and an integer **x** being the x-coordinate of the point **p** which is **(x,0)** and it produce a result of type **pic** (picture) which is a circle.

The procedure starts by rotating the point **p** through the angle **theta** about the origin in clockwise direction until the curve of the circle is completed.The procedure returns a picture of the circle.

7.2.7 procedure circle (-> pic)

This procedure produces a result of type **pic.**

The idea of this procedure is to draw a new circle inside the old circle to give the description of each final state

of the automaton.

The procedure starts by rotating the point (0.75,0) through the angle 360/32 about the origin in a clockwise direction until the curve of the circle is completed. The procedure returns a picture of the circle.

7.2.8 procedure draw.start

This procedure draws an arrow to the circle in which the name of the initial state of the automaton is placed.

The procedure starts by finding the first record of the list **A1** in which the name of the initial state of the automaton is held and draws an arrow to the circle in which the name of the initial state of the automaton is placed.

7.2.9 procedure draw.circle

The idea of this procedure is to draw a circle for each state in the list **A1.**

This procedure successively draws the circles by shifting **c(360,1)** by the given coordinates number and placing the name of a particular state inside each circle.

The procedure produces a number of circles with the corresponding name of state inside each circle.

In the example, it draws a circles for **a0 , a1 , f2** as shown in **fig 7.4.**

7.2.10 procedure input(**string** s,s1 -> **string**)

This procedure has two parameters, namely a string **s** being the name of a state **p** and a string **s1** being the name of a state **q** and produces a result of type **string** which is a concatenation of the input numbers.

The procedure successively checks each record in the list **A** to see if it contains **s** and **s1** and concatenates all the input numbers of the records containing such states and separates them by commas.

Example

Suppose $\delta(q,0) = p$

and $\delta(q,1) = p$

then the resulting string is 0,1.

7.2.11 **procedure** equal.co(**int** x1,y1,x2,y2,j,k; **string** s,s1,s2 -> **pic**)

This procedure has nine parameters. (x1,y1) and (x2,y2) are the coordinates of the centers of the two circles, an integer **j** representing either **1** or **0**, an integer **k** representing either **2** or **0**, a string **s** being a name of a state **p**, a string **s1** being a name of a state **q**, a string **s2** representing either a letter or a digit.

The idea is to draw a straight line between the two circles with **x1** = **x2** and to produce a result of type **pic**.

The procedure starts by designating a straight line on a plane, starting from the point (j,0) and ending with the point (x,0) where x represents the distance between the arcs of the two circles minus **k** plus **j**. It rotates the line by -90 and tests to see if **y1** is less than **y2**. If it is, it shifts the line by the coordinates x1,y1 and places the concatenation of the input numbers above the line. Otherwise it tests to see if **y1** is greater than **y2** . If it is, it shifts the line by the coordinates x2,y2 and places

the concatenation of the input numbers above the line.

The procedure returns a picture of the line.

7.2.12 procedure not.equal(**int** x1,y1,x2,y2,j,k; **string** s,s1,s2 -> **pic**)

This procedure has parameters as **equal.co** (see 7.2.11).

The idea is to draw a straight line between the two circles with **x1** ~= **x2** producing a result of type **pic.**

The procedure starts by designating a straight line on a plane, starting from the point (j,0) and ending with the point (x,0) where x represents the distance between the arcs of the two circles minus k plus j. It rotates the line by - **arctan** (dy/dx)X(180/pi) and tests to see if **x1** is less than **x2.** If it is, it shifts the line by the coordinates **x1,y1** and places the concatenation of the input numbers above the line. Otherwise it tests to see if **x1** is greater than **x2.** If it is, it shifts the line by the coordinates **x2,y2** and places the concatenation of the input numbers above the line.

The procedure returns a picture of the line.

7.2.13 procedure bak.arc(-> **pic**)

The idea is to draw an arc in a clockwise direction and produce a result of type **pic.**

The procedure starts by designating a point (1,0) on a plane and continuously rotating it through the angle **arctan**(0.50/5)X(180/pi) until the shape of the arc is completed.

The procedure returns a picture of the arc.

7.2.14 procedure bak(-> **pic**)

The idea is to draw an arc in anticlockwise direction and produces a result of type **pic.**

The procedure starts by designating a point (1,0) on a plane and continuously rotating it through the angle **arctan(-0.50/5)X(180/pi)** until the shape of the arc is completed.

The procedure returns a picture of the arc.

7.2.15 procedure equal.arc(**int** x1,y1,x2,y2,j,k; **string** s,s1,s2 -> **pic**)

This procedure has parameters as **equal.co** (see 7.2.11).

The aim is to draw an arc between the two circles with **x1** = **x2** and produce a result of type **pic.**

The procedure starts by calculating **d-j** where **d** represents the distance between the two circles and assigning (0.50 X d + 1) to the integer variable **q.** It tests to see if **y1** is less than **y2.** If it is, it scales **bak.arc** by (d/2 + k , 0), rotating the arc by -90 and shifts it by the coordinates **x1,y1** and places the concatenation of the input numbers above the arc. Otherwise it tests to see if **y1** is greater than **y2.** If it is, it scales **bak** by (d/2 ,1), rotates it by -90 and shifts it by the coordinates **x2,y2** and places the concatenation of the input numbers above the arc.

The procedure returns a picture of the arc.

7.2.16 procedure draw.arc(**int** x1,y1,x2,y2,j,k; **string** s,s1,s2 -> **pic**)

This procedure has parameters as **equal.co** (see 7.2.11).

The idea is to draw an arc between the two circles with **x1** ~= **x2** and produces a result of type **pic.**

The procedure starts by calculating **d-j** where **d** represents the distance between the two circles and assigns **0.50 X d + 1** to the integer variable **q.** It tests to see if **x1** is greater than **x2.** If it is, it scales **bak** by **(d/2 , 1),** shifts the arc by **(d/2 + k,0),** rotates it by the angle **-arctan(dy/dx)X(180/pi)** and shifts it by the coordinates **(x2,y2)** and places the concatenation of the input numbers above the arc. Otherwise it tests to see if **x1** is less than **x2.** If it is, it scales **bak.arc** by **(d/2 ,1),** shifts the arc by **(d/2 + k ,0),** rotates it by the angle **-arctan(dy/dx)X(180/pi)** and shifts it by the coordinates **(x1,y1)** and places the concatenation of the input numbers above the arc.

The procedure returns a picture of the arc.

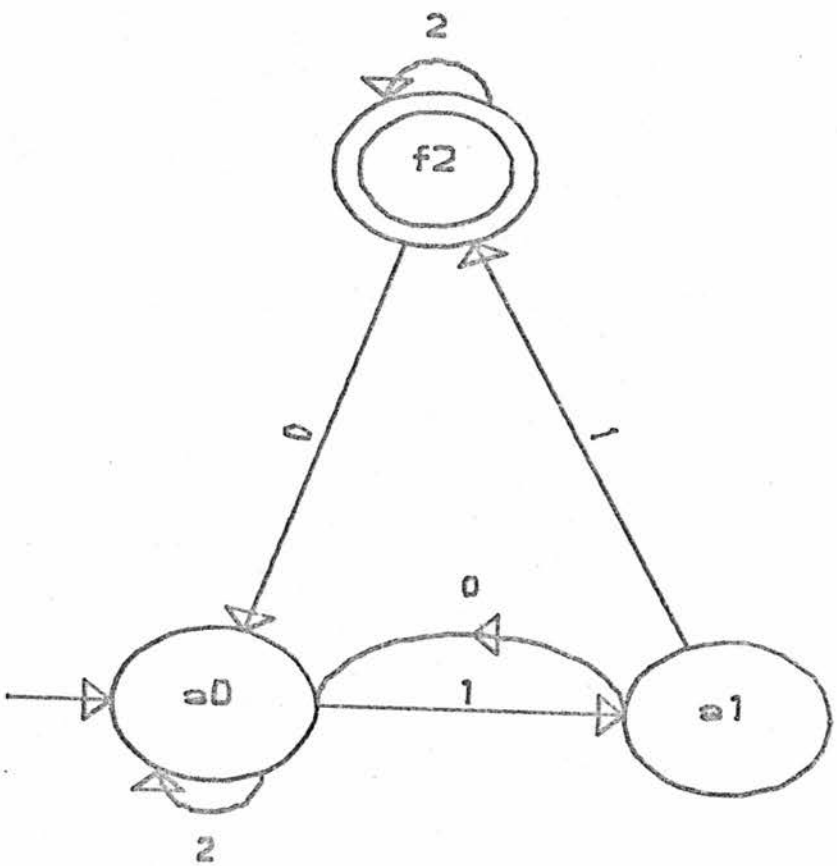
7.2.17 procedure draw.edge

The idea is to draw the lines between given circles.

The procedure successively checks each record in the list **A2** and draws a line according to the given information in each record.

In the example, the resulting diagram is shown in **fig 7.4.**

fig 7.4



Chapter 8

Mealy automaton

Introduction

The main purpose of the work described in this chapter is to enable the user to define and modify Mealy automata.

The first step is to allow the user to define states, transitions and output functions for a Mealy automaton and to subsequently modify it.

Functions are provided to trim and convert a Mealy automaton to a Moore automaton.

A Mealy automaton consists of a finite set of states together with a set of transitions or moves from state to state that occur on given input symbols. The input symbols are chosen from a given finite input alphabet. The output function g(q,a) = b produce output b in moving from state q on input a. The output symbols are chosen from a given finite output alphabet; a Mealy automaton has an initial state.

Formally a Mealy automaton [5] is denoted by a six tuple

M = (Q,S,R,f,g,q0)

where :-

(i) Q is the finite set of states.

(ii) S is the finite input alphabet.

(iii) R is the finite output alphabet.

(iv) f is the transition function or move of the Mealy automaton, it is a map from Q X S into Q

i.e (f: Q X S -> Q).

(v) g is the output function, it is a map from Q X S into R

i.e (g: Q X S -> R).

(vi) $q0 \in Q$ is the initial state.

The output of M in response to the input ala2...an is g(q0,al)g(ql,a2)...g(qn-l,an) where q0,ql,...,qn is the sequence of states such that f(qi-l,ai) = qi for $1 \leq i \leq n$.

8.1 Data structures

The following two data structures are used

8.1.1 Structures

The structures are as follows :-

(i) The structure named box defines a record consisting of a string named name and two pointers named down. and right respectively, i.e :-

structure box(string name; pntr down.,right)

(ii) The structure named rec defines a record consisting of a string named state, two integers named input and out respectively and two pointers named down and next respectively, i.e :-

structure rec(string state; int input,out; pntr down,next)

8.1.2 Linked lists

Various procedures are used to create the linked list in which a description of a Mealy automaton is held, (see 8.2 for the description of the algorithms).

8.2 The algorithms

Most of the following algorithms relate to the transition function **f(p,xl) = q** and the output function **g(p,xl) = yl** where **p , q** are states of a Mealy automaton and **xl , yl** are elements of the input alphabet and the output alphabet respectively.

8.2.1 procedure mealy(**string** s; **pntr** r -> **pntr**)

This procedure has two parameters, namely a string **s** being the name of a state **p** of the automaton and a pointer **r** to point at a record containing **s** and two pointers one pointing to the right to link all records sequentially and the other initially pointing to **nil.** It produces a result of type **pntr** to point at the whole list.

The idea is to build a list **L** of records each of which consists of a string **s** and two pointers one pointing to the right to link all records sequentially and the other initially pointing to **nil** i.e :-

structure box(**string** name; **pntr** down.,right)

L is used to hold the states of the automaton.

The idea of inserting a new record at the end of the list **L** is the same as in the description of **machine** (see 2.1 for the description of machine).

Example

Suppose a Mealy automaton consists of four states **q0 , q1 , q2 , q3** where **q0** is the initial state of the automaton.

Suppose the transition function is as follows :-

f(q0,0) = q2

f(q0,1) = q1

f(q1,0) = q0

f(q3,0) = q2

f(q3,1) = q1

and the output function is as follows

g(q0,0) = 0

g(q0,1) = 0

g(q1,0) = 0

g(q3,0) = 1

g(q3,1) = 1

Then the states of Mealy automaton are held in the list L as shown in fig 8.1.

fig 8.1

8.2.2 procedure move(string s; int x,y; pntr r)

This procedure has four parameters, namely a string **s** being the name of a state **p** of a Mealy automaton, an integer **x** being an input number **x1** an integer **y** being an output number **y1** and a pointer **r** to point at a record containing **s, x , y** and two pointers one pointing down to link all records in ascending order and the other initially pointing

to nil.

The idea is to build a list L1 of records each of which consists of a string s, an integer x, an integer y and two pointers one pointing down to link all records in ascending order and the other initially pointing to nil. i.e:-

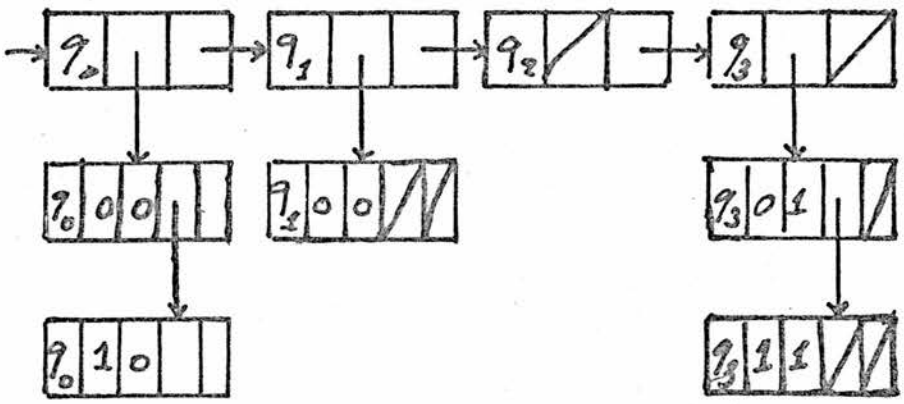
structure rec(string state; int input,out; pntr down,next.)

L1 is used to hold the states, input numbers and output numbers of a Mealy automaton.

The idea of building a list L1 of records is the same as in alt.node, (see 2.3 for the description).

In the example, the states of the automaton corresponding to the given input and output symbols are held in the structure as illustrated in fig 8.2.

fig 8.2



8.2.3 procedure mapping(string s; int x,y; string sl; pntr r)

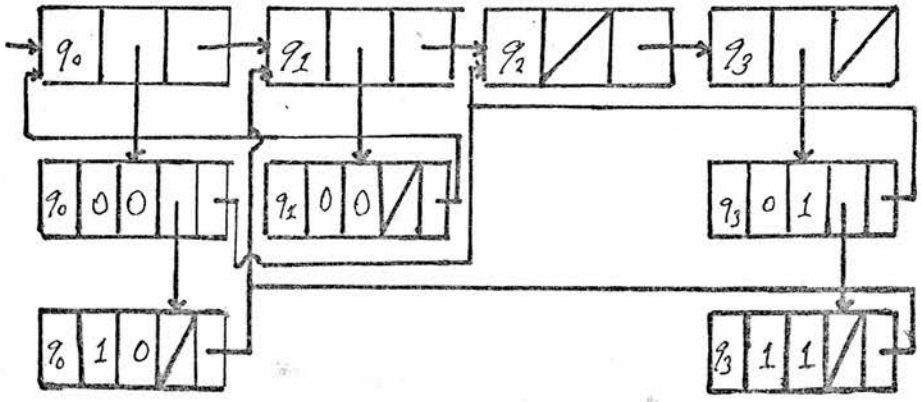
This procedure has five parameters, namely a string s being the name p of a state of a Mealy automaton, an integer x being the corresponding input number xl, an integer y being the corresponding output number yl, a string sl consisting of the new state q of the automaton and a pointer r to point at a list L.

The idea is to link each record in the list Ll with the required record in the list L. The pointer next does this.

The idea of linking each record in the list Ll with the required record in the list L is the same as in linked (see 2.5 for the description).

In the example the transition functions and the output functions are held in the structure as illustrated in fig 8.3.

fig 8.3



8.2.4 **procedure** symbol (**string** s)

This procedure has one parameter namely, a string **s** being the given sequence of input symbols.

The idea is to create a list **L2** of records each of which consists of one of the sequence of states which may occur in response to the given sequence of input symbols together with two pointers one pointing to the right to link all records sequentially and the other pointing to **nil.** i.e:

structure box(**string** name; **pntr** down.,right)

The procedure starts with the first record in the list **L** and the first input symbol in the given sequence of input symbols. It successively searches down through each list from the record containing one of the sequence of states looking for the record containing the required state and input symbol. If such a record is found, it inserts the successor state at the end of the list **L2.** Otherwise the given sequence of input symbols is unacceptable.

8.2.5 **procedure** out.string

The idea is to find the output sequence of a Mealy automaton in response to the given sequence of input symbols and produces the output sequence.

The procedure starts with the first record in the list **L2** and successively concatenates the output number of such states and separates them by commas.

8.3 Trim Mealy

The description of an algorithm to find a trim Mealy is as follows :-

(1) The system will remove any state which does not have a pointer pointing to it.

(2) The system will remove all states which do not have any pointer pointing to other states.

The algorithm

8.3.1 procedure trim

The idea is to start with the initial state of the automaton and to create a list L3 of records each of which consists of one of the states which has a pointer next pointing to it together with two pointers one pointing to the right to link all records sequentially and the other pointing to **nil** i.e:

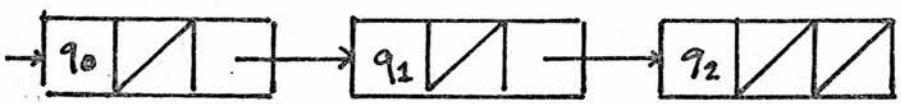
structure box(**string** name ; **pntr** down.,right)

The procedure starts with the initial state of the list L, in the example **q0**, and searches down through the sublist from the record containing the initial state looking for all states in the records pointed at by next and inserts each record containing such a state at the end of the list **L3.** In the example, the procedure inserts a record contains **q1** and a record contains **q2** successively at the end of the list **L3.** This process continues successively through each list down the record containing one of the successive states.

The procedure produces a list L3 of records which is

pointed at by a pointer named **second.** In the example **second** points at the list shown in **fig 8.4.**

fig 8.4



8.3.2 **procedure** trim.mealy

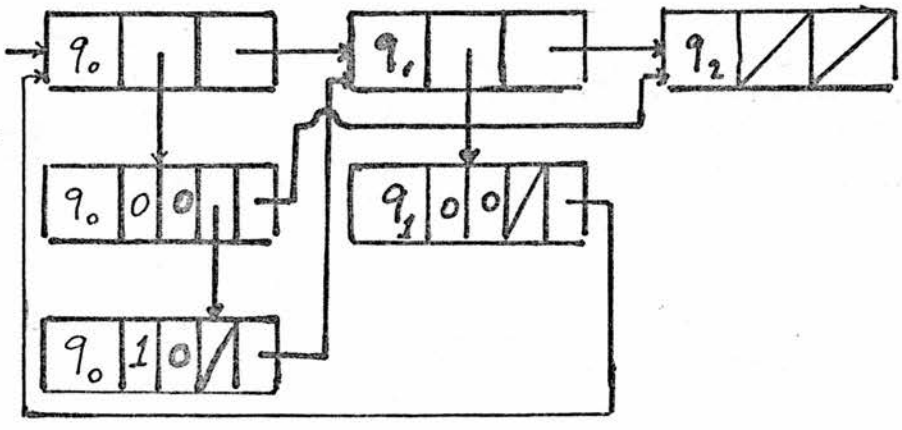
The idea is to remove all states in the list L which do not appear in the list L3 and to create a list L4 of records each of which consists of one of the reachable states from the initial state together with two pointers one pointing to the right to link all records sequentially and the other pointing to a list L1 i.e :

structure box(**string** name ; **pntr** down.,right).

The procedure successively removes all states which do not appear in the list L3 and produces a list L4 of records which is pointed at by a pointer named **head.**

In the example **head** points at the structure shown in **fig 8.5.**

fig 8.5



8.3.3 **procedure** next.move

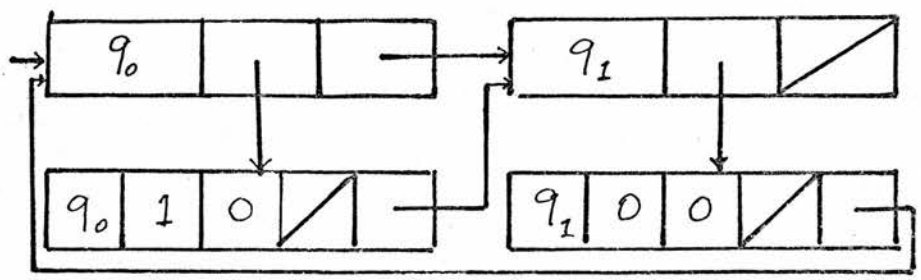
The idea is to remove all states in the list L4 which do not have a pointer pointing to other states and creates a list L5 of records each of which consists of one of the states of the trim automaton together with two pointers one pointing to the right to link all records sequentially and the other pointing to **nil.** i.e:

structure box(**string** name; **pntr** down.,right).

The procedure successively removes all records containing a pointer **down.** pointing to **nil.** In the example the procedure removes the record containing q2 and the resulting automaton which is a trim mealy automaton is held in the

structure as shown in fig 8.6.

fig 8.6



Chapter 9

Moore automaton

Introduction

The main purpose of the work described in this chapter is to enable the user to define and modify Moore automata.

The first step is to allow the user to define states, transitions and output functions for Moore automaton and subsequently to modify it.

Functions are provided to trim and to convert Moore automata to Mealy automata.

Algorithms to convert a Mealy automaton to the corresponding Moore automaton and vice versa are also described.

A Moore automaton consists of a finite set of states, a set of transitions or moves from state to state that occur on given input symbols. The input symbols are chosen from a given finite input alphabet. There is an output function from state to a given output symbol. The output symbols are chosen from a given finite output alphabet. A Moore automaton has an initial state.

Formally a Moore automaton [5] is denoted by six tuple

M = (Q,S,R,f,g,q0)

where

(i) Q is a finite set of states.

(ii) S is a finite input alphabet.

(iii) R is a finite output alphabet.

(iv) f is the transition function or move of the Moore

automaton i.e:

f : Q X S -> Q

(v) g is the output function i.e:

g : Q -> R

(vi) q0 ∈ Q is the initial state.

The output of M in response to the input a1a2...an is g(q0)g(q1)...g(qn) where

q0,q1,...,qn is the sequence of states such that

f(qi-1,ai) = ai for 1<= i <= n.

9.1 Data structures

The following two data structures are used.

9.1.1 structures

The structures are as follows :-

(i) The structure named **box** defines a record consisting of a string named **name** and two pointers named **down** and **right** respectively i.e:-

structure box(**string** name;**pntr** down,right)

(ii) The structure named **rec** defines a record consisting of an integer named **number**, an integer named **out** and two pointers named **down.** and **next** respectively i.e:

structure rec(**int** number,out; **pntr** down.,next)

9.1.2 Linked lists

Various procedures are used to create the linked list in which the description of the Moore automaton is held (see 9.2 for the description of the algorithms).

9.2 The algorithms

Most of the following algorithms relate to the transition

function

f(p,x1) = q and the output function

g(p) = y1 where p,q are states of moore automaton and x1, y1 are the input alphabet and the output alphabet respectively.

9.2.1 procedure moore(**string** s; **pntr** r -> **pntr**)

This procedure has two parameters, namely a string s being the name of a state p of the automaton and a pointer r to point at a record containing s and two pointers one pointing to the right to link all records sequentially and the other initially pointing to **nil** and produces a result of type **pntr** to point at the whole list.

The idea is to build a list R of records each of which consists of a string s and two pointers one pointing to the right to link all records sequentially and the other initially pointing to **nil** i.e:

structure box(**string** name; **pntr** down,right)

R is used to hold the states of the automaton.

The method of inserting a new record at the end of the list R is the same as in the description of **machine** see 2.1 for the description.

Example

Suppose a Moore automaton consists of four states

[q0,0], [q1,0], [q2,1], [q3,1] where [q0,0] is the initial state of the automaton.

Suppose the transition function is as follows:-

f([q0,0],0) = [q2,1]

f([q0,0],1) = [q1,0]

f([q1,0],0) = [q0,0]

f([q3,1],0) = [q2,1]

f([q3,1],1) = [q1,0]

and the output function is as follows

g(q0) = 0

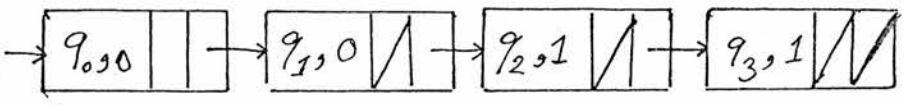
g(q1) = 0

g(q2) = 1

g(q3) = 1

Then the states of Moore automaton are held in the list R as shown in fig 9.1.

fig 9.1



9.2.2 procedure move(string s; int x,y; pntr r)

This procedure has three parameters, namely a string s being the name of a state p of a Moore automaton, an integer x being an input number x1, an integer y being an output number y1 and a pointer r to point at a record containing x, y and two pointers one pointing down to link all records in ascending order and the other initially pointing to nil. The idea is to build a list R1 of records each of which

consists of an integer **x**, an integer **y** and two pointers one pointing down to link all records in ascending order and the other initially pointing to **nil** i.e:

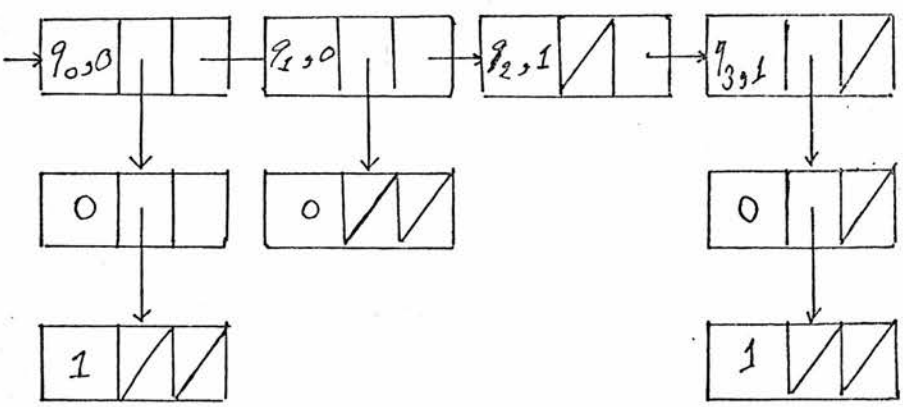
structure rec(**int** number,out; pntr down.,next)

R1 is used to hold the input numbers of the Moore automaton.

The method of building a list **R1** of records is the same as in **alt.node** (see 2.3 for the description).

In the example, the states of the automaton corresponding to the given input symbols are held in the structure as illustrated in **fig 9.2.**

fig 9.2



9.2.3 procedure mapping(**string** s; **int** x; **string** s1; pntr r)

This procedure has four parameters, namely a string **s** being the name **p** of a state of Moore automaton, an integer **x**

being the corresponding input number **xl**, a string **sl** consisting of the new state **q** of the automaton and a pointer **r** to point at a list **R**.

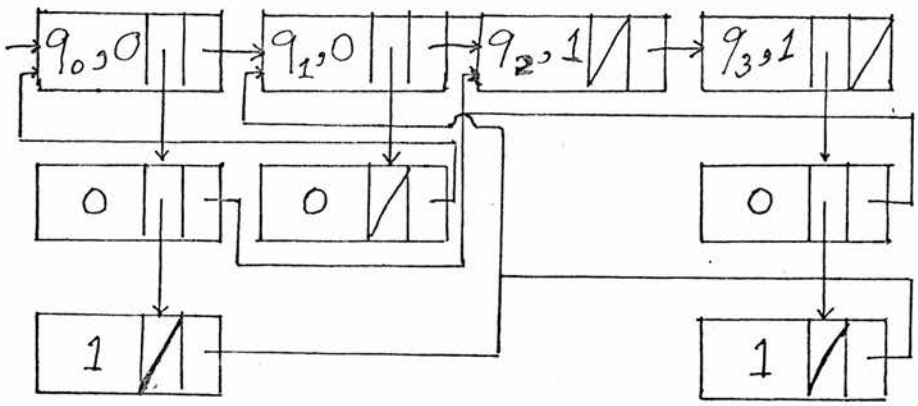
The idea is to link each record in the list **Rl** with the required record in the list **R**.

The pointer next does this.

The method of linking each record in the list **Rl** with the required record in the list **R** is the same as in **linked** (see **2.5** for the description).

In the example, the moves of Moore automaton are held in the structure as illustrated in **fig 9.3.**

fig 9.3

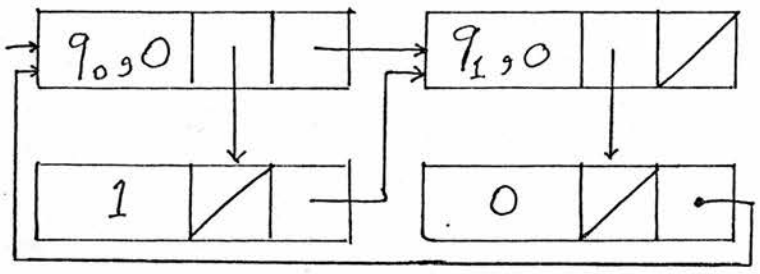


9.3 Trim moore

The description of an algorithm to find a trim Moore automaton is the same as in the description of a trim mealy automaton (see 8.3 for the description).

In the example, the resulting automaton which is a trim Moore automaton is held in the structure shown in fig 9.4.

fig 9.4



9.4 Conversion of a Moore automaton to a Mealy automaton

Introduction

Suppose M =(Q,S,R,f,g,q0) is a Moore automaton (see above for a formal definition of a Moore automaton); then we define a Mealy automaton

M1 = (Q,S,f,g1,q0)

as follows.

g1(q,a) = g(f(q,a))

for all states q and input symbols a.

Then M and M1 enter the same sequence of states on the same input symbols and with each transition or move M1 emits the output symbol that M associates with the state entered.

The description of an algorithm to find a Mealy automaton corresponding to a given Moore automaton is as follows.

The system starts with the initial state of a Moore automaton and associates each name of the states of a Moore automaton with the corresponding name of the state of a Mealy automaton and the associated output symbol.

The algorithm

9.4.1 procedure mealy

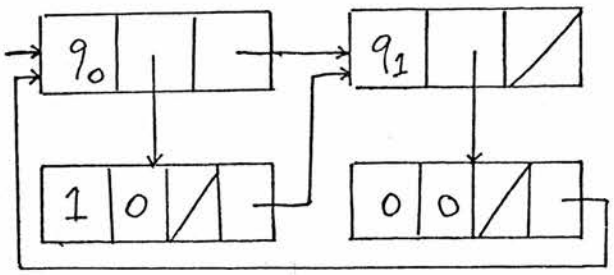
The idea is to create a list R2 of records each of which consists of the name of the state of a Mealy automaton and two pointers one pointing to the right to link all records sequentially and the other pointing down to the list R1 of

records i.e:

structure box(**string** name;**pntr** down,right)

The procedure successively separates each name of the states of a Moore automaton into the corresponding name of the state of a Mealy automaton and the associated output symbol. In the example the resulting automaton which is a Mealy automaton is held in the structure shown in **fig 9.5.**

fig 9.5



9.5 Conversion of a Mealy automaton to a Moore automaton

Introduction

Suppose M = (Q,S,R,f,g,q0) be a Mealy automaton (see chapter 8 for a formal definition of a Mealy automaton).

{a1,a2,...,an} is a finite input alphabet and {b0,b1,...,bn} is a finite output alphabet.

Suppose M1 = (QXR,S,R,f1,g1,[q0,b0]) is a Moore automaton as defined above.

The description of an algorithm to find a Moore automaton corresponding to a given Mealy automaton is as follows.

The system will start with the initial state of a Mealy automaton looking for all the moves of the automaton and concatenating each state with the corresponding output number.

The algorithm

9.5.1 procedure more(**string** s; **pntr** r)

This procedure has two parameters namely, a string **s** being the name of the state of a Moore automaton and a pointer **r** to a list **L1**.

The idea is to look for each move of a Mealy automaton and to concatenate the successor state with the corresponding output number.

The procedure successively checks each record in the list pointed at by a pointer r to see if it contains a pointer

next pointing to the successor record. If such a record is found, the procedure concatenates the successor state with the corresponding output number.

9.5.2 procedure moore

The idea is to create a list L6 of records each of which consists of the state of a Mealy automaton together with the corresponding output number and two pointers one pointing to the right to link all records sequentially and the other pointing to the list L1 i.e:

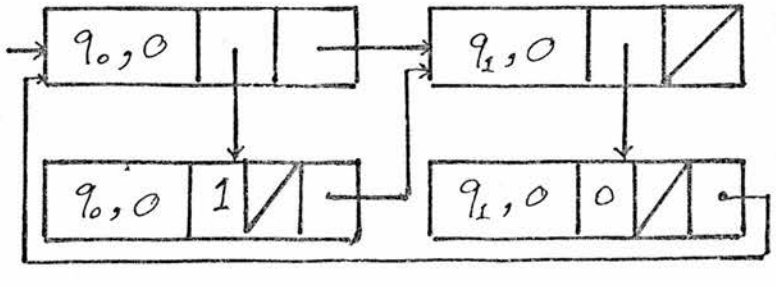
structure box(**string** name ;**pntr** down.,right)

The procedure starts with the first state in the list L5, in the example in chapter 8 q0, and searches down through the sublist from the record containing the initial state for all the moves and concatenates the initial state with the corresponding output number. In the example in chapter 8, the procedure concatenates q0 with 0 and separates them by a comma.

This process continues successively through each list down the record containing one of the successive states.

In the example in chapter 8, the resulting automaton which is a Moore automaton is held in the structure as illustrated in fig 9.6.

fig 9.6



Chapter 10

Conclusions

In this thesis we have discussed DFA, the equivalence of DFA's and NDFA's, trim automata, minimum state finite automata, regular expressions, graphics, Mealy automata, trim Mealy automata, Moore automata, trim Moore automata, the equivalence of a Moore automaton and a Mealy automaton, the equivalence of a Mealy automaton and a Moore automaton and the data structures used for defining states and transitions for the automata.

It is possible to use a tree instead of a simple linked list for holding the states and the transitions of the automaton, but the technique which is used for holding the automaton is simpler than using trees and the corresponding modification is less complicated.

To complete the discussion of the regular expressions, it is necessary to construct an automaton for the given regular expression which is closed under

1- union ;

2- concatenation ;

3- kleene star ;

4- complementation ;

5- intersection ;

This problem could well form the basis for future research.

Concerning the minimum state finite automaton, it is possible [1] to construct a table with an entry for each pair of states as follows. An X is placed in the table each time we

discover a pair of states that cannot be equivalent.

Initially an X is placed in each entry corresponding to one final state and one non final state. Next for each pair of states p and q that are not distinguishable, we consider the pairs of states $r = \delta(p,a)$ and $s = \delta(q,a)$ for each input symbol a.

If (r,s) in the table has an X, an X is also placed at the entry (p,q). Otherwise, the pair (p,q) is placed on a list associated with (r,s). At some future time, if the (r,s) entry receives an X, then each pair on the list associated with (r,s) also receives an X.

In displaying an automaton on the screen, it is possible to write more sophisticated algorithms for erasing a line or an arc from the screen without affecting the rest of the diagram including crossover points. Also it is possible to save the old information and to make a comparison between the new picture and the old picture giving the user the option of saving the better one.

The problem of drawing the " best " picture of a given finite state automaton automatically is difficult. Various algorithms assist in drawing such pictures but none of them are complete.

It would also be possible to write two algorithms for displaying a Moore automaton and a Mealy automaton on the screen and to interact with them.

References

[1] Hopcroft, J.E & Ullman, J.D. (1979). Introduction to automata theory, languages, and computation, 13-35.

[2] Eilenberg, S. (1974). Automata, languages, and machines, vol A, 23-53.

[3] Ginzburg, A. (1968). Algebraic theory of automata, 63-94.

[4] Aho, A.V & Ullman, J.D (1972). The theory of parsing, transition and compiling, vol 1, 104-106.

[5] Denning, P.J & Dennis, J.B & Qualitz, J.E (1978). Machine languages, and computation, 88-136.