MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo. No. 148.

January 1968.

SUBM

A CONVERT PROGRAM FOR CONSTRUCTING THE SUBSET MACHINE DEFINED BY A TRANSITION SYSTEM

Harold V. McIntosh[*]

[*]ESCUELA SUPERIOR DE FISICA Y MATEMATICAS
INSTITUTO POLITECNICO NACIONAL
MEXICO 14 D.F., MEXICO.

ABSTRACT

SUBM is a CONVERT program, realized in the CTSS LISP of Project MAC, for constructing the subset machine with the same behaviour as a given transition system. The program interactively collects the six items defining a transition system: its state set, alphabet, transition function, initial states, accepting states and spontaneous transitions. It then computes the subset machine, producing its state set, transition function, initial state and accepting states.

A transition system is a variant of the concept of a finite state machine which very often provides a more economical and convenient notation. One of the principal uses of a transition system is as an intermediate step in the process of constructing a finite state machine which will accept precisely those words designated by a given regular expression, since there is a very straightforward process by which the transition systems corresponding to individual regular expressions may be combined to correspond to the concatination, union, and iteration of regular expressions; the three processes from which compound regular expressions are formed from simple regular expressions.

A transition system is essentially a graph. The nodes of the graph represent the _states_ of the transition system, while the directed arrows connecting the states correspond to input words, and represent the _transitions_ of the system. Arrows representing all possible words are not drawn, but only those representing individual letters or the null word. The combinations of these primitive arrows yield composite arrows.

In addition to the graph, there is specified a subset of the states called the _initial states_ as well as another subset, called the _accepting states._ The use of the transition system is to distinguish those words for which there exists at least one path from an initial state to an accepting state from those for which there does not.

A finite state machine may also be represented by a graph, but in contrast there is only one initial state, and there is exactly one arrow emerging from each node for each letter of the alphabet, and no arrows corresponding to the null word. By contrast there may be several arrows or none corresponding to a given letter emerging from a node of a transition system.

The construction of a finite state machine equivalent to a transition system consists in taking as states of the machine subsets of states of the transition system. By taking as the image of a machine state the set of all possible states of the transition system derivable by the same letter from some state of the argument set, one ensures that there will be one image and hence one arrow for each letter of the input alphabet. Of course, the image set could be the empty set.

It then remains to define the initial and accepting states appropriately so that both the transition system and its subset machine make exactly the same distinctions of input words. The initial state of the subset machine is conveniently taken as the set of initial states of the transition system, together with any states derivable from them by means of the null word. The accepting states are those subsets which contain one of the accepting states of the transition system.

Whereas the subset machine of an n-state transition system would contain 2^n states, one is really only interested in the connected component of the subset machine, and the connected component may contain considerably less than this number of states.

The formal definition of a finite state machine, or automaton, is that it is a quintuple

$$S = \langle S, \Sigma, M, a, F\rangle$$

S is the set of states, generally carrying the same name
Σ is the input alphabet
$M: S\times\Sigma \to S$ is the transition function
$a\in S$ is the initial state
$F\subset S$ is the set of accepting states.

The formal definition of a transition system is that it is a sextuple,

$$T = \langle T, \Sigma, N, A, O, P\rangle$$

T is the set of states, generally carrying the same name
Σ is the input alphabet
$N: T\times\Sigma \to 2^T$ is the transition function
$A\subset T$ is the set of initial states
$O\subset T$ is the set of accepting states
$P\subset T\times T$ is the spontaneous transition relation.

In the usage of transition systems one requires the following further definitions.

$\hat{P}$ is the reflexive and transitive closure of P.
$|s, \sigma, t|$; There is a transition from s to t mediated by σ; when $\sigma = \lambda$, $(s,t) \in \hat{P}$; otherwise when $\sigma = w\alpha$, $|s, w, k|$ <u>and</u> $q \in N(k,\alpha)$ <u>and</u> $(q,t)\in \hat{P}$, for some k and q.

The subset construction itself consists in forming the following machine, given the transition system T.

$$S = 2^T$$

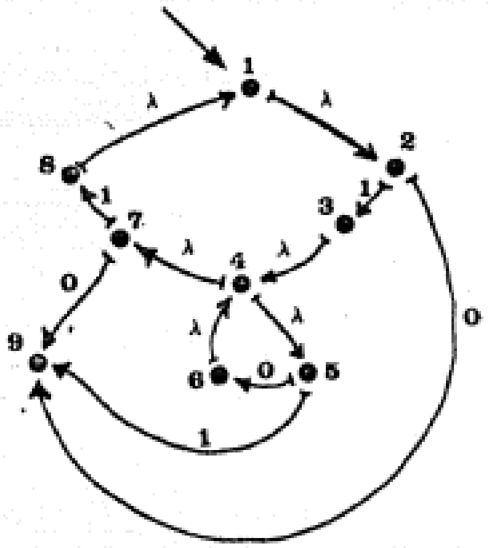
$$\Sigma = \Sigma$$

$$M(U, \sigma) = \{t \in T \mid \exists u \in U \ni |u \ \sigma \ t|\}$$

$$a = \{t \in T \mid \exists a \in A \ni (a,t) \in \hat{P}\}$$

$$F = \{U \subset 2^T \mid U \cap O \neq \phi\}$$

In order to carry out the subset construction the CONVERT program SUBM was formulated. However, on account of the LISP substrate of CONVERT, it appears in the CTSS time sharing system of Project MAC as a LISP function of no variables, (SUBM). As befits its presence in a time sharing interactive envoironment, the program gives instructions for further data to be input during the course of its execution, and in this manner the components of the transition system T are specified. The operation of the program may be illustrated by following its development during the execution of an example, which is the following transition system of nine states.



State number 1 is the only initial state, and is also the only accepting state.

The following is a transcript of the execution of this example, annotated to indicate the sequence of steps.

The gathering of data about the transition system thus follows the following course.

> Read in the alphabet
>
> Read in the state set
>
> Read in the list of initial states
>
> Build the transition table reading the transitions from one state at a time
>
> Read in the Spontaneous Transition Relation
>
> Form its Transitive Closure
>
> Read in the list of accepting states.

Thereupon the data concerning the subset machine is calculated, and output as it is determined.

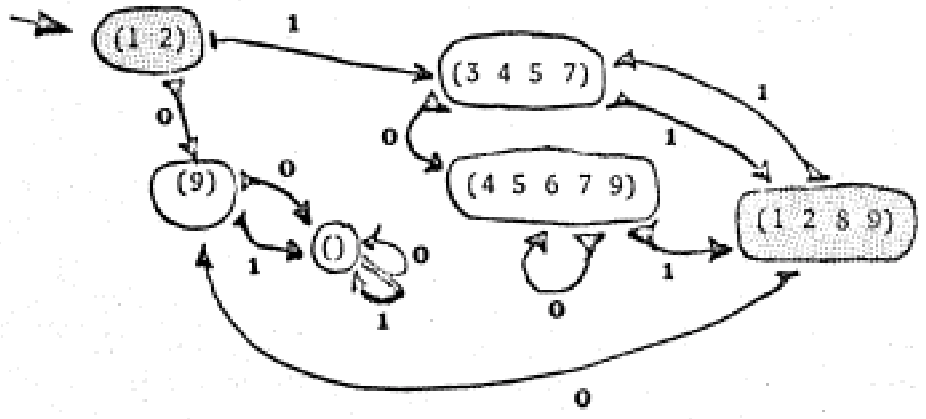
> Determine the initial state, it is the set of initial states of the transition system together with their successor states by the null word, determined from (*P*).
>
> Determine the transitions, at the same time noting the new states which arise.
>
> When all the transitions are known, the state set of the connected component of the subset machine is known and may be printed.
>
> The accepting states among them may be determined by testing their intersection with the set of accepting states of the transition system.

At this point the program enters a loop in which one may ask for specific transitions which interest him. The program is terminated by writing ST; other requests will be unceremoniously rejected.

```
r                            CTSS will now execute CONVERT
W 1620.0                     the time is a signal that CONVERT is ready
load ((subm))                load the disc file SUBM LISP
NIL                          synonymm for (), means the file is loaded
subm ()                      LISP is asked to execute (SUBM)
(ALPHABET)                   SUBM asks for the alphabet of the transition system
(0 1)                        which is a binary alphabet
(STATE SET)                  SUBM now asks for a list of the states
(1 2 3 4 5 6 7 8 9)          which are numbers, 1 through 9
(INITIAL STATES)             SUBM asks for the initial states
(1)                          there is but one, but it must be listed
(TRANSITIONS)                SUBM now asks for the function N
1                            requesting first images of state number 1
()                           of which there are none
2                            SUBM asks for the image of state number 2
((1 3) (0 9))                1 carries 2 into 3; 0 carries 2 into 9
3                                 SUBM continues to list the states, to which
()                                the response is a list whose elements are
4                                 lists; the input letter followed by the
()                                states to which that letter causes
5                                 transitions.  If there are no image
((0 6) (1 9))                     states the letter need not be mentioned
6                                 either. Thus if 1 caused transitions to
()                                states 3, 4, and 5, but 0 caused no
7                                 transitions, the response would be
((0 9) (1 8))                     ((1 3 4 5)).
8
()
9
()
((9) (8) (7 (0 9) (1 8)) (6) (5 (0 6)  (1 9)) (4) (3)
(2 (1 3) (0 9)) (1))         SUBM writes its internal representation of N
(SPONTANEOUS TRANSITIONS) SUBM now asks for the spontaneous transitions
1                            first those originating from state number 1
(2)                          of which 2 is the only such state
2                                 SUBM continues listing the states in
()                                order, and expects in each case a list of
3                                 those states to which the system may pass
(4)                               spontaneously, indicated in the transition
4                                 diagram by arrows labelled λ.  Id there are
(5 7)                             none, the response is ().
5
()
6
(4)
7
()
8
(1)
9
()
```

```
(TRANSITIVE CLOSURE)          SUBM now computes the transitive and reflexive
(9)                                closure of the spontaneous transition
(8 1 2)                            relation which it has been given, and
(7)                                prints the result, in the form of a list
(6 4 5 7)                          of those states accessible from the first
(5)                                state  of the list, by means of a chain
(4 5 7)                            of null words.
(3 4 5 7)
(2)
(1 2)
(ACCEPTING STATES)            SUBM now asks for a list of the accepting states
(1)                           of which 1 is the only one, but it must be listed
==
(SUBSET MACHINE)              We now begin the output of the description of the
==                                 subset machine
(INITIAL STATE)               First the initial state is listed; it is a subset
((1 2))
(TRANSITIONS)                 followed by the transition function M, which is
(ORIGINATING FROM (1 2))           defined as a table which has the form
((1 2) (0 (9)) (1 (3 4 5 7)))      (=== (s === (i t) ===) ===), which is
==                                 to say that it is a list of sublists, one for
(ORIGINATING FROM (3 4 5 7))       each state.  One of these sublists begins with
                                   the name of the state, and contains pairs (i t)
                                   which state that letter i causes a transition
                                   to state t.  There is one such pair for each
                                   letter of the alphabet.
((3 4 5 7) (0 (4 5 6 7 9)) (1 (1 2 8 9)))
==
(ORIGINATING FROM (1 2 8 9))
((1 2 8 9) (0 (9)) (1 (3 4 5 7)))
==
(ORIGINATING FROM (4 5 6 7 9))
((4 5 6 7 9) (0 (4 5 6 7 9)) (1 (1 2 8 9)))
==
(ORIGINATING FROM (9))
((9) (0 ()) (1 ()))
==
(ORIGINATING FROM ())
(() (0 ()) (1 ()))
==
(STATE SET)                   The states comprising the subset machine are listed
()
(9)
(4 5 6 7 9)
(1 2 8 9)
(3 4 5 7)
(1 2)
(ACCEPTING STATES)            The accepting states are listed
((1 2 8 9) (1 2))
st                            SUBM is commanded to cease operation
(=== GOODBYE ===)             to which it responds amiably
stop                          LISP is asked to cease operation
VALUE                         to which it returns its value
NIL
R 13.683+20.650               and CTSS the total elapsed time
```

With the results so obtained we may draw the transition diagram of the subset machine.



The CONVERT program itself is a fairly straightforward realization of the theoretical steps necessary for the construction. SUBM is primarily a CONVERT Program, which calculates in order the quantities of interest after having collected the necessary data. Necessary sets are treated as lists, while the transition functions are treated as tables, and also stored as lists with an appropriate structure. This structure is (=== (s === (i t) ===) ===); which means that for every state s there is a list of transitions headed by the name of the states. The entry (i t) means that M(s, i) = t, in the case of a machine. In the case of the transition system, the entry would be (i ttt), where the fragment ttt was a list of the elements comprising the set of images under i; thus N(s, i) = (ttt).

There is a problem in the representation of sets by lists that the sets so represented may be equal without the corresponding lists being equal, due to the fact that lists more properly represent ordered sets. CONVERT does not immediately recognize set equality; rather than revise CONVERT to include a pattern to test equality of sets, it is preferable to normalize the lists representing sets. This is accomplished by a double complementation, if S is the set and U a subset, the normalized form of U is (*COMP* S (*COMP* S U)). It is to be recalled that *COMP* does not change the order of the surviving elements of its argument.

The argument M of CONVERT, in the definition of SUBM, contains the following five entries.

(*READ*)	On account of CONVERT not containing the fragment analogue of =READ=, which reads one "S-Expression" from the teletype console.
NORM	A function (NORM X) which places the elements of the subset of states X in their standard order, by double complementation.
(SUST)	A fragment-valued function (SUST S L) which finds the "successors of a state" wherein S is a state, and L is a letter of the alphabet. It first of all finds all those states listed in the transition table (*TR*) as images of S under L, and then examines the spontaneous transition table (*P*), which is extended to its transitive and reflexive closure in an initial stage of the program, to see how many further states arise from the image states by spontaneous transitions.
SUSE	(SUSE X L) concatinates the successor states for each element of the set S according to the letter L; it therefore finds the"successors of a set."
ADJS	In forming the set of states of the subset machine, we do not undertake to discover all the subsets of the state set of the transition system, but rather those of its connected component, which arise from the initial state by some chain of transitions. In the process a list of subsets already found is kept, (*SS*). (ADJS X) is used to "adjoin a state" to this set. If X is new, it is adjoined to (*SS*).

The argument I of CONVERT consists of variables S, L, and X; fragments XXX and YYY.

The argument E of CONVERT is vacuous, since the program receives its data by means of =READ=.

The argument R of CONVERT is one single program. Its program variables have the following significance.

=A=	The common alphabet of the transition system and the subset machine.
=AA=	The accepting states of the subset machine.
=I=	The initial states of the transition system.
=S=	A state of the subset machine.
(*S*)	The state set of the transition system.
(*SS*)	The state set of the subset machine.

(*TR*)	The list representing the function N; that is the transition table of the transition system.
(*TT*)	The list representing the function M, that is, the transition table of the subset machine.
(*P*)	The spontaneous transition relation, which is determined by an *READ* cycle; for each state there are listed that state and its spontaneous images. This relation is promptly replaced by its transitive closure, which form it retains throughout the program.
(*F*)	The list of accepting states of the transition system.
(*W*)	A working variable used in various parts of the program.

Following the program in rough outline we see that it follows the steps seen in the example. The necessary data to define the transition system are collected. A request for the data is written on the user's console, and when it is furnished, is stored in an appropriate program variable. No test is made to see if the data furnished is plausible, which assumes that a fairly experienced person will use the program. A syntactic check could be included if necessary. Some of the data is collected step by step; rather than inputting the whole transition table at once, it is called for state by state, the program volunteering the state name so that none will be overlooked.

The transitive closure of the spontaneous transition relation is calculated by a subprogram. This program uses the program variables (II) and (I*). Initially, (*P*) is the input spontaneous transition relation, a list whose sublists consist of the different states followed by their immediate spontaneous images. The variable (I) runs through these lists, to the current one of which (II) is initially set. Thus (II) contains the immediate image of a certain state, s. For each of these immediate images, an immediate image is sought, giving second images derivable from s not by λ, but by $\lambda\lambda$. Any new states found by this process form the set (I*). The new states are adjoined to (II), and the cycle repeated with a search for third images, and so on. The process stops when no new images are found, and the next state is investigated until none are left.

```
     DEFINE ((
(SUBM (LAMBDA () (CONVERT
(QUOTE (
   (*READ*)  SKEL    ((*CONC* =READ=))
     NORM    REPT    (((X) (=COMP= (*S*) (=COMP= (*S*) X))))
    (SUST)   REPT    (((S L) (=WHEN= (*TR*) (=== (S === (L XXX) ===) ===)
                          (=ITER= J (XXX) (*SKEL* J VAR J (=WHEN= (*P*) (=== (J YYY) ===) (J YYY)  ()))))
                          ()))))
     SUSE    REPT    (((X L) (NORM (=ITER= I X (SUST I L))))))
     ADJS    REPT    (((X) (=WHEN=  (*SS*) (=== X ===) X (=WHEN= (=SETQ= (*W*) (=UNON= (X *W*))) == X))))
     ))
(QUOTE (
     S L X (XXX) (YYY)
     ))
(LIST)
(QUOTE (*0 (
(== (=PROG= ( =A= =AA= =I= =S= (*S*) (*SS*) (*TR*) (*TT*) (*P*) (*F*) (*W*) )
     (=PRNT= (=QUOT= (ALPHABET)))
     (=SETQ=  =A= =READ=)
     (=PRNT= (=QUOT= (STATE SET)))
     (=SETQ= (*S*) =READ=)
     (=PRNT= (=QUOT= (INITIAL STATES)))
     (=SETQ= =I= =READ=)
     (=PRNT= (=QUOT= (TRANSITIONS)))
     (=ITER= J (*S*) (=PROG= () (=PRNT= J) (=SETQ= (*TR*) ((J *READ*) *TR*))))
     (=PRNT= (*TR*))
     (=PRNT= (=QUOT= (SPONTANEOUS TRANSITIONS)))
     (=ITER= J (*S*) (=PROG= () (=PRNT= J) (=SETQ= (*P*) ((J *READ*) *P*))))
     (=PRNT= (=QUOT= (TRANSITIVE CLOSURE)))
     (=SETQ= (*P*) (=ITER= (I) (*P*) (=PROG= ( (II) (I*) )
               (=SETQ= (II) (I))
               1
               (=SETQ= (I*) (=COMP= (=UNON= (=ITER= J (II) (*SKEL* J VAR J (=WHEN= (*P*)
                                 (=== (J YYY) ===) (J YYY) ()))))) (II)))
               (=WHEN= (I*) () (=RETN= (II)))
               (=SETQ= (II) (II I*))
               (=GOTO= 1)
               )))
     (=ITER= I (*P*) (=PRNT= I))
     (=PRNT= (=QUOT= (ACCEPTING STATES)))
     (=SETQ= (*F*) =READ=)

     (=PRNT= ==)
     (=PRNT= (=QUOT= (SUBSET MACHINE)))
     (=PRNT= ==)

     (=PRNT= (=QUOT= (INITIAL STATE)))
     (=SETQ= (*W*) ((NORM (=ITER= J =I= (*SKEL* J VAR J (=WHEN= (*P*) (=== (J YYY) ===) (J YYY) ())))))))
     (=PRNT= (*W*))
     (=PRNT= (=QUOT= (TRANSITIONS)))
     1
     (=WHEN= (*W*) () (=GOTO= 2))
     (=WHEN= (*W*) (X ===) (=SETQ= =S= X))
     (=PRNT= ((*QUOT* (ORIGINATING FROM)) =S=))
     (=SETQ= (*SS*) (=S= *SS*))
     (=WHEN= (*W*) (== XXX) (=SETQ= (*W*) (XXX)))
(=SETQ= (*TT*) ((=PRNT= (=S= (*ITER* I =A= (I (ADJS (SUSE =S= I))))))) *TT*))
(=PRNT= ==)
(=GOTO= 1)
2
(=PRNT= (=QUOT= (STATE SET)))
(=ITER= I (*SS*) (=PRNT= I))
(=PRNT=  (=QUOT= (ACCEPTING STATES)))
(=SETQ= =AA= (=PRNT= (=ITER= I (*SS*) (*WHEN* (=INTS= I (*F*)) () () (I)))))
3
(=REPT= =READ= *1 (
      ((S L) (=PRNT= ((SUST S L))))
      (ST      (=RETN= (=== GOODBYE ===)))
      (== (=PRNT= (PTUI ===== =SAME= =====)))
      ))
(=GOTO= 3)

))
)))
)))
))
```

REFERENCES

Subset Construction:

Michael A. Harrison, INTRODUCTION TO SWITCHING AND AUTOMATA THEORY, New York: McGraw-Hill Book Company, 1965, Chapter 13.

CONVERT:

Adolfo Guzman and Harold V. McIntosh, "CONVERT," Communications of the Association for Computing Machinery 9 604-615 (1966).

Harold V. McIntosh and Adolfo Guzman, "A MISCELLANEY OF CONVERT PROGRAMMING," Project MAC Artificial Intelligence Group Memo 130 (April 1967).