T H E   U N I V E R S I T Y   O F   M I C H I G A N

COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Department of Philosophy

Final Report

APPLICATION OF LOGIC TO THE DESIGN OF COMPUTING MACHINES

A. W. Burks
H. Wang
J. Holland

UMRI Project 2512

under contract with:

UNITED STATES AIR FORCE
AIR RESEARCH AND DEVELOPMENT COMMAND
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
CONTRACT NO. AF 18(603)-72
BALTIMORE, MARYLAND

administered by:

THE UNIVERSITY OF MICHIGAN RESEARCH INSTITUTE   ANN ARBOR

August 1959

engn

UMR0958

PREFACE

This final report has two parts. In the first, the results obtained on fixed automata are summarized. (The detailed account of these results is to be found in the technical report, The Logic of Automata, by Arthur W. Burks and Hao Wang.) Results obtained on growing automata are summarized in the second part.

TABLE OF CONTENTS

PART I

"THE LOGIC OF AUTOMATA"*—SUMMARY

Arthur W. Burks and Hao Wang

*AFOSR TN-56-539, ASTIA AD-110-358, also published in the _Journal of the Association for Computing Machinery_, _4_, No. 2, 193-218, 279-297.

Classes of automata are distinguished: fixed and growing, deterministic and probabilistic (Section 2.1).* Attention is then focussed on finite automata. A finite automaton is a fixed finite structure with a fixed finite number of input junctions and a fixed finite number of internal junctions such that (1) each junction is capable of two states, (2) the states of the input junctions at every moment are arbitrary, (3) the states of the internal junction at time zero are distinguished, (4) the internal state (i.e., the states of the internal junctions) at the time t+1 is completely determined by the states of all junctions at time t and the input junctions at time t+1, according to an arbitrary pre-assigned law (which is embodied in the given structure). An abstract automaton is obtained from an automaton by allowing an arbitrary initial internal state (Section 2.1).

Let there be M possible input states (M possible combinations of activations or non-activations on the input wires). Designate by I the class of input states. Let there be N possible internal states and designate by S the class of internal states. Let there be P output states and designate by O the class of output states. The values of M and N and P are not necessarily powers of two since the structure or use of the automaton may prohibit certain states from occurring (Section 2.2).

An Automaton is in general characterized by two arbitrary effective transformations (τ and λ) from pairs of integers to integers. These integers are drawn from finite sets {I}, {S}, and {O}. {S} contains a distinguished integer S_O (the initial internal state of the automaton). The transformations are given by

$$S(0) = S_O$$

$$S(t+1) = [I(t), S(t)]$$

$$O(t) = [I(t), S(t)]$$

(If we omit the condition $S(0) = S_O$, we obtain an abstract automaton.) (Section 2.2.)

We can produce a table of MxN pairs $\ll I,S$, $S' >$ such that, if $< I,S >$ is part of the state at t, then S' is part of the state at t+1. We shall call this set the complete table of the given automaton. Each complete table is a definition of the function τ. In a similar way we can construct an output table for the automaton, each now being of the form $\ll I,S >, O >$. Such a table defines the function λ. The function τ and the complete table involve a time shift, while the function λ and the output table do not. For an investigation of the behavior of an automaton through time the complete table is basic, the output table derivative (Section 2.2).

*Section references are to the complete version of The Logic of Automata.

Two abstract automata are equivalent just in case they have the same complete table. (If the complete tables are the same, then for the same initial internal state and the same input functions, the initial complete states in the two automata are the same, and the same complete state at any moment plus the same input functions always yield the same complete state at the next moment. Section 2.2.)

In the case of automata with predetermined initial internal states (i.e., nonabstract automata), identical complete tables are sufficient but not necessary to show the equivalence of any two automata. This is possible because it can happen that, for every pair $\ll I,S>, S'>$ which occurs in one table but not in the other, we can never arrive at the internal state S from the distinguished initial internal state S_0, and hence can never have the complete state $< I,S>$, no matter how we choose the input functions. To secure a necessary and sufficient condition, it suffices to determine all the internal states which the given initial internal state can yield when combined with arbitrary input words, and then to repeat this process with each of the internal states thus found: if two complete tables coincide insofar as all the pairs occurring in these determinations are concerned, the two automata are equivalent. Since there are only finitely many pairs in each complete table, the process of determination will repeat itself in a finite time.

To describe the procedure more exactly, one can think of a "tree" with the chosen initial internal state S_0 as the trunk. From the trunk M branches are grown, one for each possible input word, with the corresponding internal state at the next moment of time at the end of the branch. Whenever in the construction we come to an S that is already on the tree, we stop; otherwise we grow M branches on it as before. This process is continued as long as some new internal state is introduced at every height. Since there are altogether only M _a priori_ possible internal states, the number of distinct branch levels of the tree cannot exceed M. Such a tree will be called the _admissibility tree_ of the automaton.

If we collect together the ordered pairs which represent branches of the admissibility tree, we obtain a subset of the complete table which we shall call the _characterizing table_ of the automaton. Two automata are equivalent if they have the same characterizing table. Since there is an effective method of deciding whether two automata have the same characterizing table, we have a decision procedure for testing whether two automata are equivalent (Section 2.2).

If the words of the characterizing table are put in a binary form and the digits related to the junctions of the automaton, it can be decided whether or not two junctions behave the same (Section 2.2).

Automata can be represented by diagrams called automata nets, which show the internal structure of automata. These nets are composed of _switching elements_ which realize truth functions, and _delay elements_ which delay an input one unit of time. _Well-formed nets_ are constructed by the following rules:

(1) A switching element or delay element is well-formed.

(2) If N_1 and N_2 are disjoint well-formed nets, then

(a) the juxtaposition of N_1 and N_2 is well-formed:
(b) the result of joining junctions of N_1 to input junctions of N_2 is well-formed;
(c) the result of joining input junctions of N_1 is well-formed; and
(d) the result of joining junctions of N_1 to junctions of N_1 which are delay element input wires is well-formed (Section 2.3).

A normal-form net is organized as follows. It has a direct-transition switch, fed by the net inputs and the delay outputs, and driving the delay inputs. It has an output switch, fed by the delay outputs and the net inputs, and not driving any delay elements. Given a sufficiently rich set of switching elements, we can always obtain the normal form net for any well-formed net. We can label the input wires to a normal form net and let I (the input state) be the concatenation of the negated (for nonactivated) and unnegated (for activated) symbols for the separate input wires. In similar fashion we obtain designation for delay inputs, delay outputs, and direct-transition switch. We let the initial concatenation of states of delay output wires be S_0 (the initial state of the automaton). The remaining elements of S are the other concatenations of states of the delay output wires determined by the net and its inputs. From the states of the net we can construct a complete table. By use of the admissibility tree we can construct the characterizing table. Therefore, given a well-formed net, we can construct a complete table, a characterizing, and an output table describing its behavior. By considering the binary codings, if we are given a complete table, characterizing table, and an output table, we can construct a well-formed net realizing these tables (Section 2.3). The information in a characterizing table can be arranged in a direct-transition matrix. The rows and the columns of the matrix are labeled with the internal states of the automaton; the matrix entries themselves are the input states which will take the automaton from one internal state to another. When there is no input which will directly take the automaton from a certain internal state to another particular internal state the matrix entry "$\emptyset$" is to appear (Section 3.1).

An abstract automaton is backwards deterministic if and only if for each finite sequence I(0), I(1), ..., I(t), S(t+1), there is a unique sequence S(0), S(1), ..., S(2) satisfying the complete table. A direct transition matrix is backwards deterministic if and only if for each I and S there is at most one other internal state such that I and this other internal state directly produce S. If a direct-transition matrix is backwards deterministic, the associated abstract automata is backwards deterministic (Section 3.3).

Fixed, deterministic nets can be analyzed in terms of cycles. A sequence of junctions A_1, A_2, ..., A_n, A_1 (possibly with repetitions) constitutes a cycle if and only if each A_j is an input to an element whose output is A_k, where $k \equiv j+1$ modulo n. Thus a junction occurs in a cycle if it is possible to start at that junction, proceed forward (in the direction of the element wire arrows) through switching elements and delay elements, and ultimately return to the junction. A junction which does not occur in a cycle has _degree_ zero, as does an input-independent junction. The degree of a noninput-independent junction which occurs in a cycle is the maximum number of distinct delay elements it is possible to pass through by traveling around cycles in which the junction occurs. The _degree of a net_ is the maximum of the degrees of its junctions (Section 4.1).

The close correspondence between switching nets and the theory of truth functions has been noted earlier. If to the theory of truth functions we add a δ operator (which when applied to the input value of a delay element gives the output value of that delay element) and also allow the system to include expressions like $t = a$, $t > a$, $(t-a) \equiv c \bmod b$ (where t is a variable and a, b, and c are integers) we can describe any periodic function. This system is called the _extended theory_ of truth-functions. For every junction of a net of degree zero, we can effectively construct a formula of the extended theory of truth functions which describe the behavior of the junction as an explicit function of the behavior of the inputs (Section 4.2).

For explication of all the foregoing remarks, including many examples of the processes discussed (and also consideration of and comment on numerous other automata problems), the reader is referred to the original technical report.

PART II

"A UNIVERSAL COMPUTER CAPABLE OF EXECUTING AN ARBITRARY NUMBER OF SUB-PROGRAMS SIMULTANEOUSLY"

John Holland

ABSTRACT

The paper describes a universal computer capable of simultaneously executing an arbitrary number of sub-programs, the number of such sub-programs varying as a function of time under program control or as directed by input to the computer. Three features of the computer are: (1) the structure of the computer is a 2-dimensional modular (or iterative) network; (2) each sub-program is spatially organized, thus facilitating the simulation of "highly parallel" systems having many points or parts; and (3) the computer's structure and behavior can, with simple generalizations, be formulated so as to make it a useful abstract tool for investigating problems in automata theory.

SUMMARY

The main purpose of this paper is to describe a univeral computer capable of simultaneously executing an arbitrary number of sub-programs under program control. The formulation is intended as an abstract prototype which, if current component research is successful, could lead to a practical computer.

The computer can be considered to be composed of modules arranged in a 2-dimensional rectangular grid; the computer is homogeneous (or iterative) in the sense that each of the modules can be represented by the same fixed logical network. The modules are synchronously timed and time for the computer can be considered to be divided into discrete units or time steps, $t = 0, 1, 2, \ldots$.

Basically each module consists of a binary storage register together with associated circuitry and some auxiliary registers. At each time-step a module may be either active or inactive. An active module, in effect, interprets the number in its storage register as an instruction and proceeds to execute it. There is no restriction (other than the size of the computer) on the number of active modules at any given time. Ordinarily, if a module $M(i,j)$ at coordinates (i,j) is active at time-step t, then at time-step, $t+1$, $M(i,j)$ returns to inactive status and its successor, one of the 4 neighbors $M(i+1,j)$, $M(i,j+1)$, $M(i-1,j)$, or $M(i,j-1)$, becomes active. The successor is specified by bits s_1, s_2 in $M(i,j)$'s storage register. If we define the line of successors of a given module as the module itself, its successor, the successor of the sucessor, etc., then a given sub-program in the computer will usually consist of the line of successors of some module.

The action of a module during each time-step can be divided into three successive phases.

1. During Phase One, the input phase, a module's storage register can be set to any number supplied by a source external to the computer. Although the number in the storage register can be arbitrarily changed during Phase One, it need not be; for many purposes the majority will receive input only during the first few moments of time ("storing the program") or only at selected times t_1, t_2, ... ("data input").

2. During Phase Two, an active module determines the location of the storage register upon which its instruction is to operate by, in effect, marking a path to it. The path-building action depends upon two properties of modules. First, each module has a neighbor, distinct from its successor, designated as its predecessor by bits q_1, q_2 in its storage register; the line of predecessors of a given module is then defined as the sequence of all modules M_0, M_1, ..., M_k, ... such that, for each k, M_k is the predecessor of M_{k-1} and M_{k-1} is the successor of M_k. Secondly, by setting bit p in its storage equal to 1, each module may be given a special status which marks it as a point of origination for paths; the module is then called a P-module.

Each path must originate at P-module, and only one path can originate at any given P-module. The modules belonging to a given path can be separated into subsequences called segments. Each segment consists of y modules extending parallel to one of the axes from some position (i,j) through positions $(i+b_1, j+b_2)$, $(i+2b_1, j+2b_2)$, ..., $(i+(y-1)b_1, j+(y-1)b_2)$, where $b_1=1, 0, -1$ and $b_2=\pm(1-|b_1|)$; the module at $(i+yb_1, j+yb_2)$ will be called the termination of the segment. Each module possesses 4 *-registers and if the module belongs to a segment in direction (b_1, b_2) the appropriate *-register, $(b_1,b_2)^*$, is turned on gating lines between (i,j) and $(i+b_1,j+b_2)$. Since each *-register gates a separate set of lines, a module may (with certain exceptions) belong to as many as four paths. Once a *-register is turned on, it stays on until it is turned off; thus a path segment, once marked, persists until "erased."

Each segment of a path results from the complete Phase Two action of a single active module. At the start of Phase Two the path specification bits y_n, ..., y_0 and d_1, d_2 are gated down the line of predecessors from the storage register of the active module to the nearest P-module along the line of predecessors. Let the termination of the final segment of the path originating at that P-module be called the nearest path termination. Then, if $y_n= 0$, bits y_{n-1}, ..., y_0 and d_1, d_2 determine the length and direction, respectively, of a new segment which has as its initial module the nearest path termination. If $y_n=1$, then the final segment of the path is erased (bits y_{n-1}, ..., y_0 and d_1,d_2 not being used in this case).

3. Phase Three, the execution phase, depends upon the fact that, by setting the pair of bits (p,a) in a module's storage register to the value (0,1) the module can be given a special status which marks it as an accumulator; a module in this status is called an A-module. The number in the storage register of an A-module is treated simply as a binary number with y_n being the sign bit and y_{n-1} the high-order digit.

Three modules play an important role during the execution phase of an active module: the active module itself holds the order code in bits i_1, i_2, i_3 of its storage register; the storage register of the nearest path termination contains the word to be operated on; and the A-module nearest along the line of predecessors, after the nearest P-module, serves as the accumulator. For example, if an active module has a TRANSFER ON MINUS order coded in bits i_1, i_2, i_3, and if the number in the nearest A-module is minus, then at the end of the time-step the module at the nearest path termination becomes active instead of the active module's successor.

In the present formulation there are 8 types of instruction: (1) TRANSFER ON MINUS, which has already been described; (2) ADD; (3) SUBSTRACT; and (4) STORE act similarly, the nearest path termination serving as the address for the order; (5) NO ORDER causes the execution phase to pass without the execution of an order; (6) STOP prevents an active module from passing on its activity; (7) ITERATE SEGMENT causes 1 to be subtracted from the number in the the A-module, and, if the result is positive, the active module remains active on the next time-step (rather than passing on its activity); thus the path-building phase of the given module is iterated; and (8) SET REGISTERS causes the first 9 bits of the A-module's number to be used to set all 9 auxiliary registers at the nearest path termination; by setting the auxiliary register which gives the module active status, both that module and the successor of the active module will become active on the next time-step.

To resolve possible conflicts when several active modules interact, a set of interlock and over-ride rules are required. In the present formulation 10 such rules suffice.

Recapitulating briefly, the universal computer described has three prominent features:

1. The basic structure is iterative.

2. The sub-programs have a spatial organization and any number can be executed simultaneously.

3. Features (1) and (2) make possible theoretical investigation of the interactions of various classes of sub-programs (e.g., by considering the rectangular grid to cover an infinite plane, cf. Church's potentially infinite automata and Von Neumann's scheme for self-reproducing automata).

It should be noted that the present formulation is representative of a broad class of similar machines.

DISTRIBUTION LIST

(One copy unless otherwise noted)

Commander 3
European Office, ARDC
47 Rue Cantersteen
Brussels, Belgium

Applied Mathematics and
Statistics Laboratory
Stanford University
Stanford, California

Commander
Fifteenth Air Force
Attn: Operations Analysis Office
March Air Force Base, California

Department of Mathematics
University of California
Berkeley, California

Assistant CINCSAC
(SAC MIKE)
Attn: Operations Analysis Office
Box 262
Inglewood, California

Commander
Air Force Flight Test Center
Attn: Technical Library
Edwards Air Force Base, California

The RAND Corporation
Technical Library
1700 Main Street
Santa Monica, California

Commander
1st Missile Division
Attn: Operations Analysis Office
Cooke Air Force Base
Lompoc, California

Department of Mathematics
Yale University
New Haven, Connecticut

Director of Advanced Studies
Air Force Office of Scientific
Research
P. O. Box 2035
Pasadena 2, California

Commander
Western Development Division
Attn: WDSIT
P. O. Box 262
Inglewood, California

Office of Naval Research 2
Department of the Navy
Attn: Code 432
Washington 25, D.C.

Department of Commerce
Office of Technical Services
Washington 25, D.C.

Director of National Security
Agency
Attn: Dr. H. H. Campaigne
Washington 25, D.C.

Library
National Bureau of Standards
Washington 25, D.C.

National Applied Mathematics
Laboratories
National Bureau of Standards
Washington 25, D.C.

Headquarters, USAF
Assistant for Operations Analysis
Deputy Chief of Staff, Operations,
AFOOA
Washington 25, D.C.

Department of Mathematics
Northwestern University
Evanston, Illinois

DISTRIBUTION LIST (Continued)

Commander 2
Air Force Office of Scientific
Research
Attn: SRDB
Washington 25, D.C.

Commander 2
Air Force Office of Scientific
Research
Attn: SRE
Washington 25, D.C.

National Science Foundation
Program Director for Mathe-
matical Sciences
Washington 25, D.C.

Commander
Air Force Armament Center
Attn: Technical Library
Eglin Air Force Base, Florida

Commander
Air Force Missile Test Center
Attn: Technical Library
Patrick Air Force Base, Florida

Institute for Air Weapons Research
Museum of Science and Industry
University of Chicago
Chicago 37, Illinois

Department of Mathematics
University of Chicago
Chicago 37, Illinois

Department of Mathematics
University of Illinois
Urbana, Illinois

Department of Mathematics
Purdue University
Lafayette, Indiana

Mathematics and Physics Library
The Johns Hopkins University
Baltimore, Maryland

Commander, Second Air Force
Attn: Operations Analysis Office
Barksdale Air Force Base, Louisiana

Institute for Fluid Dynamics and
Applied Mathematics
University of Maryland
College Park, Maryland

Department of Mathematics
Harvard University
Cambridge 38, Massachusetts

Commander, Eighth Air Force
Attn: Operations Analysis Office
Westover Air Force Base
Massachusetts

Department of Mathematics
Massachusetts Institute of Technology
Cambridge 38, Massachusetts

Commander
Air Force Cambridge Research Center
Attn: Geophysics Research Library
L. G. Hanscom Field
Bedford, Massachusetts

Commander
Air Force Cambridge Research Center
Attn: Electronic Research Library
L. G. Hanscom Field
Bedford, Massachusetts

Department of Mathematics
Wayne State University
Detroit 1, Michigan

Willow Run Research Center
The University of Michigan
Ypsilanti, Michigan

Department of Mathematics
Institute of Technology
Engineering Building
University of Minnesota
Minneapolis, Minnesota

DISTRIBUTION LIST (Continued)

Department of Mathematics
Folwell Hall
University of Minnesota
Minneapolis, Minnesota

Department of Mathematics
Washington University
St. Louis 5, Missouri

Department of Mathematics
University of Missouri
Columbia, Missouri

Commander
Strategic Air Command
Attn: Operations Analysis
Offutt Air Force Base
Omaha, Nebraska

The James Forrestal Research
Center Library
Princeton University
Princeton, New Jersey

Library
Institute for Advanced Study
Princeton, New Jersey

Department of Mathematics
Fine Hall
Princeton University
Princeton, New Jersey

Commander
Air Force Missile Development
Center
Attn: Technical Library
Holloman Air Force Base
New Mexico

Commander
Air Force Special Weapons Center
Attn: Technical Library
Albuquerque, New Mexico

Professor J. Wolfowitz
Mathematics Department, White Hall
Cornell University
Ithaca, New York

Department of Mathematics
Syracuse University
Syracuse, New York

Mathematics Research Group
New York University
Attn: Professor M. Kline
25 Waverly Place
New York, New York

Department of Mathematics
Columbia University
Attn: Professor B. O. Koopman
New York 27, New York

Commander
Rome Air Development Center
Attn: Technical Library
Griffiss Air Force Base
Rome, New York

Commander
Rome Air Development Center
Attn: RCWEA
Griffiss Air Force Base
Rome, New York

Institute for Aeronautical Sciences
Attn: Librarian
2 East 64th Street
New York 21, New York

Institute of Statistics
North Carolina State College of
A and E
Raleigh, North Carolina

Department of Mathematics
University of North Carolina
Chapel Hill, North Carolina

DISTRIBUTION LIST (Concluded)

Office of Ordnance Research 2
Box CM, Duke Station
Durham, North Carolina

Department of Mathematics
Duke University, Duke Station
Durham, North Carolina

Commander
Air Technical Intelligence Center
Attn: ATIAE-4
Wright-Patterson Air Force Base
Ohio

Commander
Wright Air Development Center
Attn: Technical Library
Wright-Patterson Air Force Base
Ohio

Commander 2
Wright Air Development Center
Attn: ARL Technical Library, WCRR
Wright-Patterson Air Force Base
Ohio

Commandant 2
USAF Institute of Technology
Attn: Technical Library, MCLI
Wright-Patterson Air Force Base
Ohio

Department of Mathematics
Carnegie Institute of Technology
Pittsburgh, Pennsylvania

Department of Mathematics
University of Pennsylvania
Philadelphia, Pennsylvania

Commander
Arnold Engineering Development
Center
Attn: Technical Library
Tullahoma, Tennessee

Applied Mechanics Reviews 2
Southwest Research Institute
Attn: Executive Editor
8500 Culebra Road
San Antonia 6, Texas

Defense Research Laboratory
University of Texas
Austin, Texas

Department of Mathematics
Rice Institute
Houston, Texas

Commander
Air Force Personnel and Training
Research Center
Attn: Technical Library
Lackland Air Force Base
San Antonio, Texas

Armed Services Technical
Information Agency 10
Arlington Hall Station
Arlington 12, Virginia

Department of Mathematics
University of Wisconsin
Madison, Wisconsin

Mathematics Research Center
U. S. Army
Attn: R. E. Langer
University of Wisconsin
Madison, Wisconsin