Kybernetika

Evžen Kindler

Programming means for simulation of logical networks. I

Kybernetika, Vol. 8 (1972), No. 6, (517)--534

Persistent URL: http://dml.cz/dmlcz/124661

KYBERNETIKA — VOLUME 8 (1972), NUMBER 6

Programming Means for Simulation of Logical Networks I

EVŽEN KINDLER

The paper contains the information about programming of simulation programs concerning the array of logical networks. The matter is arranged so that it can be accepted also by the users who have not been trained in general methods of algorithmic programming of computers. The present part contains the programming methodics for the first generation computers. The use of the programming means typical for the second and third generation computers is described in the following parts.

1. GENERAL CONCEPTIONS

Nowadays we can hope that the meaning of the word *simulation* has been fixed at least for some decade of years. The origine of the modern interpretation of the same word has been founded by Monte-Carlo methods (see [2]). The modifications of these interpretations have started by using the same word in modelling of continuous systems by means of analogue computers. Of course, there are some theoretical aspects which could put equivalence between both the interpretations, because the same aspects and their similar effects were known from the quantum physics: the mean values of certain attributes of continuous processes (i.e. classical physical variables or state characteristics of analogue computers) are limit cases of mean values of corresponding attributes of corresponding discrete processes (i.e. mean values in quantum physics or in Monte-Carlo methods – see [4]). Nevertheless the analogy has implied no unification in interpreting the word simulation (may be for a great difference in the used metodics in Monte-Carlo methods and in analogue computer applications) but – moreover – has implied a greater chaos: the generalization of the analogy has led to a generalization of the use of the word simulation, e.g. in numerical analysis (in case of numerical methods based on random numbers). The hasty development in computing technics, influenced by propagation slogans of the business-oriented people, has amplified that process, as any new interpretation

of any "scientific" word gave an illusion of a new facilities of the propagated computer techniques.

But the same computing means which have caused the latter explosion in the interpretations of the word simulation have caused also the final unifying of them: the real property of the automatic computers (i.e. computers with the program that can modify itself-see e.g. [5]) – its universality – has led to accentuation of the same universality in case of simulation so that nowadays there is a definition of this word, accepted by the world of investigators and other specialists, which is an general that we can take it as definitive. This definition is the following:

The simulation is a research technique the substance of which is that the researched system is replaced by its model with which the experiments are performed in order to obtain information about the originally given system.

The terms used in the definition and the actual aspects of the same definition have been presented in [1]. Let us only mean that nowadays there are three great fields to apply the simulation: prognostics of the behaviour of systems, determination of structural parameters of systems according to their behaviour, training the work with simulated systems. If the model (mentioned in the definition above) is programmed at a computer, we put an attribute *computer* before the work simulation. If the computer is a digital one (especially an automatic computer) we put an attribute *digital* before the same word. In the present paper only the last type of simulation is already considered; thus we can omit the attribute digital, supposing it before any pattern of the word simulation occuring in the next text.

The *logical network* is a special case of dynamical system (see [1]) risen by a generalization of a classical concept of neuron network (see e.g. [3]). It is a system of objects called *neurons*, each of which has n inputs ($n = 0, 1, 2, \ldots$) and one output. At the inputs logical values can enter into the neuron as input informations. At the output there leaves the neuron also a logical value which is a function of the input values. The input and output values go into and from the neurons in discrete time moments so that either the output value leaves the neuron in the same time as the corresponding values enter the same neuron, or the output value leaves the neuron in the next time moment as the corresponding values enter the same neuron through its inputs. The possibility is given for every pattern of neuron as a fixed one; thus we speak about a *neuron without delay* if the output information leaves it in the same time as the corresponding input informations, or we speak about a *neuron with delay* if the output information leaves the neuron with delay of one time step (difference between two time moment one of which follows the other one) after the corresponding input informations. Of course, the corresponding informations must enter always simultaneously. The logical system is composed of one or more neurons: more neurons can be connected one to other so that the output of one is joined with one or more inputs of other neurons of the same system. The information which leaves the first neuron through its input enters immediately eventual neurons through the joined inputs. The cycles of neurons without delay through which an information

can leave a neuron, go through the other neuron of the same cycle and then return to the first neuron (modified eventually during that way), are forbidden. We shall write the logical values as 1 (true) and 0 (false), because they were in the same way interpreted in the computer models.

2. SIMULATION OF LOGICAL NETWORKS AT FIRST GENERATION COMPUTERS

2.1. The automatic computers of the first generation can be characterized by the following properties: their electronical circuits are based in electronic tubes, their memories are realized as magnetic drums, where one computer can store about 1000 words; these words have constant length which corresponds about 36 bits. Every computer has a small number of input and output units; when a program runs at the computer, one can see clearly the phases of computation at the input and output units and he can control it manually from the controle desc (konsole). The computers are facilitated by standart subroutines and by systems of automatic programming of simple algorithmic type: one writes the rules of computation desired in a sequence which corresponds to the sequence in which they are performed; there are simple facilities to describe cycles jumps and subroutines. Such a description is punched into paper tape or into cards and the translator (compiler) transforms the description into the machine code (see e.g. [6], [7], [8]).

2.2. The neurons can be programmed in such a programming language (called first-generation-language) so that every neuron gets a sequence of instructions, where its output value is computed from the input values, represented by corresponding identifiers. It is suitable to enumerate all the neurons of the simulated system – every neuron gets its order number (a natural number). Thus an identifier for an output value can be written e.g. An where A is the character of the machine alphanumeric code and n is the order number. E.g. $A7$ is the output information which leaves the neuron enumerated by 7 (we shall say simply the 7-th neuron). The input informations are identified similarly; e.g. the neuron enumerated as the 5-th one, which performs the conjunction of the output informations from the neurons enumerated by numbers 2 and 4 is programmed as following:

$$A5 = A2 \,.\, A4$$

The neuron enumerated by 3 joined at the output from the 6-th neuron, which performs, the negation, is programmed as following:

$$A3 = 1 - A6$$

Thus the program is consisted of such patterns of sequences of instructions. After the last pattern there is programmed the print of results: it may contain a sequence of instructions so that they might print a table of values in outputs of neuron which

interest us. Every line contains e.g. k fields where the logical value at certain places of the simulated system are printed. As the facilities for printing are different for every computer, we shall not describe the prints detailly (see the example presented further). This series of patterns of instructions is a body of cycle, which causes that the body is repeated so that the simulation is performed as long as we wish. Before the heading of the cycle one can put a pattern of program which performs the actions desired at the beginning of the simulation, after the last instruction of the body there can be programmed the action demanded after the simulation (e.g. printing of a conclusion, stop instruction). Schematically the simulating program has the following form:

```
inital action
FOR T = 1 STEP 1 UNTIL K
BEGIN
pattern for one neuron
pattern for another neuron
      . . . . . .
pattern for the last neuron
instructions for printing of the results
END
conclusion action
```

The second line is an example taken from ALGOL 60 (see [6]) or from MOST (see [7]) which are typical algorithmical programming languages; it means that the body (closed in the words *BEGIN* and *END* which are here in a function of parentheses) might be repeated for every value of T (identifier of the simulated time) equal 1, 2, 3, ..., K, where K denotes the duration of the investigated process.

2.3. The important problem is the ordering of program segments in the body. Always the pattern for printing has to be the last one in the body because the values must be at first computed and then printed. For the patterns corresponding to the neurons is the ordering controlled by a more complicated rules: evidently if a neuron enumerated by a number k is joined to an neuron without delay enumerated by a number m so that an input of the k-th neuron is connected with the output from the m-th neuron, the pattern corresponding to the m-th neuron must be written before the pattern corresponding to the k-th neuron. Let us note, that this rule is always satisfiable as the cycles of neurons without delays are forbidden. The ordering of patterns for the neurons with delay can be made by two different way which we shall call first type or order and second type of order (this affair will be interpreted also for the second and third generation of the computers).

2.4. The first type of order is controled by the rule that the pattern of program corresponding to a neuron with delay must always follow all the patterns of programs corresponding to neurons the inputs of which are joined at the output of the neuron

with delay. Thus when the k-th neuron is connected by one of its input with the output of the m-th neuron which is that with delay, the pattern, of the m-th neuron must occur after the pattern of the k-th neuron. This way of order need no additional programming tools and all the patterns can be done as it has been demonstrated by the examples in 2.2. From the fact that the instructions in the body are performed in the same order as they are written follows that the delay is well simulated. In one step the instructions simulating the neurons, where the information should come from a neuron with delay, are performed at first, so that the input values are taken from the preceding step. Then the simulation of the neuron with delay is performed, but the output information which leaves it, is only printed, if it is desired: as the input it can function only in the next step, because all the patterns which need it have been written before and thus they have been already performed before, if we consider the present step only. The described type of ordering has one obstacle: if there is a cycle composed only of the neurons with delay, the rule cannot be satisfied. In this case one can put a "fictive" neuron in any place of such a cycle, which is without delay and which interprets at his aoutput the information identical with the information in its only one input. Let us present an example: let a neuron enumerated by 4 is connected with a neuron enumerated by 7 in a cycle; let both the neurons are with delay.

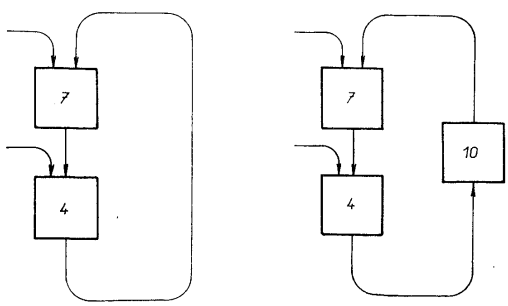

Fig. 1. a) b)

Thus their patterns of programs cannot be ordered so that the pattern of the 7-th neuron is before the pattern of the 4-th one and inversely. Thus we can enrich the original cycle (see fig. 1a) by a new neuron without delay, enumered e.g. by 10 (see fig. 1b). The new cycle can be ordered satisfying the rule:

$A10 = A4$
pattern for the 4-th neuron
pattern for the 7-th neuron

The first line of the program part is the pattern from the fictive neuron enumerated by 10.

2.5. The second type of order admits any ordering for the neurons with delay (the ordering of the neurons without delay must however satisfy the general rule expressed above). Even any fictive neurons are not needed. But the pattern of the programs corresponding to the neurons with delay must be done in a different way: the result of computation simulating the action of the neuron must be identified by another variable, e.g. *Bn* where *n* is the number with which the neuron is enumerated. Thus a neuron performing the conjunction of the values which come from the neurons enumerated by 4 and 5, if it is with delay and if it is enumerated by 6 has the corresponding pattern:

```
B6 = A4 . A5
```

The body of the cycle contains one program pattern, which is not present in the first type of order: it is a pattern for transfer (assigning) of all the values of *Bk* to corresponding *Ak*, for all the order numbers *k* of neurons with delay. The described pattern is placed at the end of the body of the cycle, after the pattren for the prints. Thus the organization of the simulation program is the following:

```
initial action
FOR T = 0 STEP 1 REPEAT K
BEGIN
pattern for one neuron
pattern for another neuron
. . . . . .
pattern for the last neuron
instructions for printing of the results
transfers from Bk to Ak
END
conclusion action
```

This way has an obstacle: beside the use of new address for *Bk* one must take attention about all the possible *k* which correspond to neurons with delay, and then one must specially program all the transfers from *Bk* to *Ak*. It is not suitable, even for the automatization of the whole programming process (see further – for the second generation computers). One can minimize the obstacle so that the enumeration is performed at first for the neurons with delay, the number of which is stored in the memory (of programmer or – in case of eventual automatization of the programming – in the memory unit of the translator), and then the enumeration of the remaining neurons, i.e. those without delay is done. In this case the transfers from *Bk* to *Ak* (see the previous schema) can be programmed by one cycle – a subcycle of the main cycle of simulation:

```
FOR I = 1 STEP 1 REPEAT R
BEGIN
AI = BI
END
```

where *R* identifies the number of the neurons with delay. But on can present a more

uniform technique of programming, which needs some more of computer run (there are some instructions without importance) which can be nevertheless well automatized. In this technique one programs the neurons with delay in the same way as it has been just demonstrated. But the neurons without delay are programmed more complicatedly: the result in sent to *An* both to *Bn*, where *n* is the order number of the corresponding neuron. E.g. a program pattern for the 7-th neuron without delay which performs the conjunction of informations comming from the neurons enumerated by 2 and 3 is the following:

$A7 = A2 \,.\, A3$ or $B7 = A2 \,.\, A3$
$B7 = A7$ $A7 = B7$

Another example for the 5-th neuron which performs the negation of the information comming from the 4-th neuron:

$A5 = 1 - A4$ or $B5 = 1 - A4$
$B5 = A5$ $A5 = B5$

Then the transfers from *Bi* to *Ai* can be performed at the end of every step for all $i = 1, 2, \ldots, n$ where n is the complete number of all the neurons which occur in the simulated system. The transfers are superfluous in case of the neurons without delay but the programming is more compact: the organization of the simulation program is the following:

initial section
FOR T = 0 STEP 1 REPEAT K
BEGIN
pattern for one neuron
pattern for another neuron
. . . . .
pattern for the last neuron
instructions for printing of the results
FOR I = 1 STEP 1 REPEAT n
BEGIN
AI = BI
END
END
conclusion action

We can see that the variable identified *R* (or the corresponding constant) is not used here.

2.6. The first generation languages admit to program all sorts of neurons. Let us present here the program patterns for the most known sorts:

The neurons with one input, connected with the output of the *k*-th neuron, while the described neuron is enumerated by the integer *j*:

negation:

$Aj = 1 - Ak$

identical transfer (the neuron which reproduces in its output the same value which has entered through its input):

$$Aj = Ak$$

identical constant 0 (the output information of the neuron is always 0, without regard to the input information):

$$Aj = 0$$

identical constant 1:

$$Aj = 1$$

The neurons with two inputs, connected with the outputs from the neurons k-th and m-th respectively, while the described neuron is enumerated by the integer j:

conjunction:

$$Aj = Ak \,.\, Am$$

disjunction:

$$Aj = Ak + Am - Ak \,.\, Am$$

the same operation programmed for very simple programming facilities (admitting only binary operations in the instructions):

$$\begin{array}{lll} X = Ak \,.\, Am & \text{or} & X = 1 - Ak \\ X = Ak - X & & X = X \,.\, Am \\ Aj = X + Am & & Aj = X + Ak \end{array}$$

implication $(Ak \to Am)$:

$$Aj = 1 - Ak + Ak \,.\, Am$$

the same operation programmed in binary operations:

$$\begin{array}{lll} X = Ak \,.\, Am & \text{or} & X = Am - 1 \\ X = X - Ak & & X = X \,.\, Ak \\ Aj = 1 + X & & Aj = 1 + X \end{array}$$

equivalence:

$$Aj = 1 - Ak - Am + 2 \,.\, Ak \,.\, Am \quad \text{or} \quad Aj = (1 - 2 \,.\, Ak) \,.\, (1 - Am + Ak)$$

the same operation programmed in binary ones:

$$\begin{array}{lll} X = Ak \,.\, Am & \text{or} & X = Ak - 1 \\ X = X + X & & X = Ak + X \\ X = X - Ak & & X = Am \,.\, X \\ X = X - Am & & X = X - Ak \\ Aj = 1 + X & & Aj = X + 1 \end{array}$$

Sheffer stroke:

$$Aj = 1 - Ak \,.\, Am$$

The same operation in the binary ones:

$$X = Am \,.\, Ak$$
$$Aj = 1 - X$$

Symmetrical difference:

$$Aj = Ak + Am - Ak \,.\, Am \,.\, 2$$

The same operation programmed in the binary ones:

$X = Ak \,.\, Am$	or	$X = 1 - Am$
$X = X + X$		$X = X - Am$
$X = Ak - X$		$X = X \,.\, Ak$
$Aj = Ak + Am$		$Aj = X + Am$

One can similarly program other eventual operations performed by neurons with two inputs. As concern the neurons with more inputs the technique of programming is similar but the possibilities would overflow the capacity of this paper. Thus we shall present only two examples:

The neuron which performs the disjunction of 4 inputs; they come from the neurons 2-th, 3-th, 4-th, 5-th while the neuron itself is the 6-th one:

$$X = A2 + A3 + A4 + A5$$
$$A6 = 1 \; IF \; X > 0$$
$$A6 = 0 \; IF \; X = 0$$

The same function programmed by binary operations:

$$X = A2 + A3$$
$$X = X + A4$$
$$X = X + A5$$
$$X = 1 \; IF \; X > 1$$
$$A6 = X$$

The neuron with three exciting inputs (connected with the outputs from the 2-th, 3-th and 4-th neurons) and with one suppressing input connected with the output from the 7-th neuron. The neuron inself is enumerated by 8. The same neuron has sensitivity bound 2. It means: if at the exciting inputs come at least two values 1 and if at the suppressing input comes 0, the output value is 1, otherwise it is 0 (see [3]).

$$X = 0$$
$$IF \; A7 = 1 \; GO \; TO \; M$$
$$Y = A2 + A3 + A4$$
$$IF \; Y < 2 \; GO \; TO \; M$$

$X = 1$
$M: A8 = X$

The same neuron can be modelled by the following pattern of program composed of binary operations:

$X = 0$
IF $A7 = 1$ *GO TO* M
$Y = A2 + A3$
$Y = Y + A4$
IF $Y < 2$ *GO TO* M
$X = 1$
$M: A8 = X$

Let us note that the presented patterns concern the case of the first type of ordering. In the other case one would either modify the last instruction so that instead of the letter A at the left hand side of the assign ment the letter B is put, or join a new instruction $Br = Ar$ where r is the index of the assigned variable at the left hand side of the last instruction, according to the discussion presented in the paragraph 2.5.

2.7. In the logical networks there are often generators. They can generate a logical constant, random logical value or a function of time. As concern the generation of logical constants, one possibility has been already presented in the preceding paragraph — the generator of a logical constant would be a neuron with "fictive" input. But it is rather sophisticated possibility which can be formulated among some specialists but not for people who would like to simulate the logical networks for the purposes of the logical networks and not for the purposes of the theory of programming. Thus we can define an element generator as a neuron without inputs, the output information of which is generated according to the proper function of the neuron: it can be a constant, a random logical value, a value of a function of the time etc. The program patterns for all the generators are to be put before all the other program pattern, therefore at the beginning of the body of the main cycle.

Let us note that the program patterns for the generators of constants are the same as presented in the preceding paragraph. The program pattern for the generation of random logical value is the following:

$X = RANDOM(0, 1)$
$Ar = X$

where *RANDOM* is the function in machine code which generated random or probably pseudorandom numbers (see [9], page 48), while the assignment in the second instruction includes rounding of the value X (from the interval $\langle 0, 1 \rangle$) to the nearest integer. If the function generating random numbers does not exist in the machine software it must be programmed in machine code. If the assignment does not include the rounding we must use the sequence of the following instructions in place of the last assignment instruction:

$Ar = 0$ *IF* $X < 0{\cdot}5$
$Ar = 1$ *IF* $X \geqq 0{\cdot}5$

Let us present the sequence of machine instructions which has appeared as very suitable sourse of trandom numbers from the interval $\langle 0, 1\rangle$. The author thanks to his collaborator Z. Režný for testing the statistical properties of that source:

1. transfer the contents of the address S into the accumulator;
2. multiply it in fixes point by 1220703125 (i.e. by a certain power of 5) so that the upper part of the product remains in the accumulator, the lower part goes in a register;
3. transfer the contents of the register in the accumulator;
4. transfer the contents of the accumulator into the address S;
5. take the contents of the accumulator as the fixed-point result of the generating and transform it eventually into floating-point form.

Thus the random number remains in the accumulator from what it can be transferred in any address. The address S must contain an odd integer at the beginning of the simulation.

As concern of the generators of any function which is not a constant we do not recommend to use them, as one can come into contradictions if he use the procedures (see further): such generators need certain "inner states" which could influence one another in case that the semantics of variables used in procedures is not exactly known (that semantics varies for diverse softwares of computers). We recommand to compose such generators of more simple generators (if they are needed) and of neurons with inputs. E.g. the generator of the function which has its value 1 in odd time moments and 0 otherwise can be made of a cycle containing the only one neuron which performs the negation of the value occuring at its input. It is a neuron with delay and its output (identical with the output of the generator) is connected with its input (thus one must join an identical-function neuron in that cycle in case of the first type of order).

2.8. In the studying of the logical networks one need often the inputs into the whole system which is to be simulated. In other words one need often to simulate the logical networks which process the logical values which enter them from their surroundings. It can be simulated so that the surrounding of the simulated network is modelled by the surrounding of the computer and the inputs from outside into the simulated system are simulated by the patterns of program which read the values punched at a paper medium (or – and it is logically equivalent – from a control console, teletype etc.). In the figures those inputs are presented by means of certain lines the end of which corresponds to the physical entering of the signal, carrying the information, from the surroundings into the system. This mode is suitable for arranging in the programming tool: we can consider such an end of a line as a generator; the program pattern for its simulation contains the instructions for reading an input information, i.e. an information perforated in the computer input medium. Similarly as the other generators

the input units get their proper order number. If such a unit gets e.g. the number k the corresponding program pattern is the following:

READ Ak

More input units into the simulated system can be simulated by more input units of the computer or by special ordering of the information at the only one input: if there are e.g. three input units for the simulated system, enumerated 3, 4 and 6 so that their program patterns are in the order: pattern of the 6-th input unit, pattern of the 3-rd unit and patternd of the 4-th unit the input file of the computer must contain the following informations: $I_1^6, I_1^3, I_1^4, I_2^6, I_2^3, I_2^4, I_3^6, I_3^3, I_3^4, I_4^6, \ldots$ where I_s^t is the information prepared for the t-th input unit at time s. In the following parts we shall not distinguish between proper generators and input units: both types are neurons called generators, without input simulated in the considered network, for both types there will hold the same rules expressed only for generators.

2.9. The output units – i.e. the lines through which the information leaves the simulated system – need not interess us, because after leaving the simulated system the information has no meaning for the simulation. As the leaving informations are usually those which interess us so that we let them to be print, we can metaphorically say that the input unit of the system is simulated by the pattern of program which performs the printing of the results in every step of the simulation.

2.10. The use of procedures and functions, which is a facility of a lot of algorithmical programming systems even in the first generation of the computers, can cause the programming more simple: instead of writing the instructions for simulating the functions of any neuron in any pattern of program we can declare all the necessary logical functions outside the simulating program (often only once and to let it rewrite in any program for the purpose of simulation) while the program patterns would contain the only instruction for each of them, i.e. calls of the corresponding function. E.g. the program pattern for a neuron which is enumerated by 6 and which performs a disjunction of the informations comming from the 2-th and from the 4-th neurons, is the instruction

A6 = DISJUNCTION (A2, A4)

while outside the simulating body there is declared

FUNCTION DISJUNCTION (X, Y)
DISJUNCTION $= X + Y - X \cdot Y$

Another possibility is that the program pattern has the following form:

DISJUNCTION (A2, A4, A6)

while the declaration of the procedure outside the program body has the following form:

From the form of the declarations one can see the danger which can be present in the case of the generators of functions: the declarations of various functions need the variable which occurs only in the declaration; it is the variable (or a system of variables) representing the "inner state" of the generator. If the same procedure is called in two or more places of the simulating program (i.e. if the simulated system contains two or more generators of the same function) one can get into a conflicting situation if the programming system does not represent any called procedure by its proper pattern of variables. Theoretically it would be possible to restrict the calling of generators so that one could call the same generator only once but even in this situation it would be possible to construct conflicting situations (e.g. by using the same letter in different declarations). The descriptions of the semantics of applied programming languages are often as unexact that one cannot clearly recognize whether the mentioned conflicts would be caused or not.

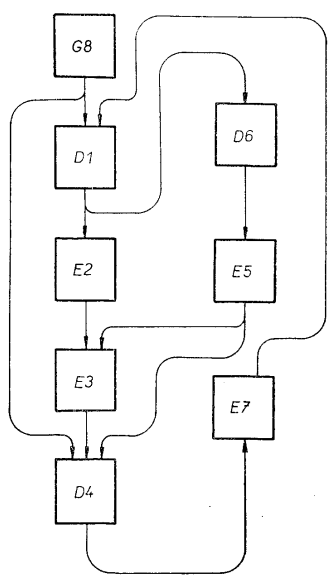

Fig. 2.

2.11. Let us present an example for simulation of a simple logical network by a simple computer. The network is presented at the fig. 2. There the numbers mean

the enumeration of the neurons while the letters preceding the numbers mean the type of the neuron: G is a generator, D is a neuron with delay, E is a neuron without delay. The functions of the neurons are the following:

1-st neuron: conjunction,
2-nd neuron: negation,
3-rd neuron: disjunction,
4-th neuron: conjunction,
5-th neuron: negation,
6-th neuron: identical,
7-th neuron: negation,
8-th neuron: input.

The simulating computer is ODRA 1013 (see [10]), facilitated by the system MOST of automatic programming (see [7]), which satisfies all the properties of the algorithmic languages of the first generation, described in this part. Let us present the simulation program using the first type of order; at the right hand side of the program there are comments and explications:

```
INTEGER X, T, D, A8                heading of the program according to
LABEL 1                            the rules of MOST
BEGIN
1: READ D                          initial section (only reading of D)
FOR T = 0 STEP 1 UNTIL D           heading of the basic simulation cycle
READ A8                            pattern of the 8-th neuron
A5 = 1 - A6                        pattern of the 5-th neuron
A7 = 1 - A4                        pattern of the 7-th neuron
A6 = A1                            pattern of the 6-th neuron
A2 = 1 - A1                        pattern of the 2-nd neuron
A1 = A8 * A7                       pattern of the 1-st neuron
X  = A2 * A5                       }
X  = A2 - X                        } pattern of the 3-rd neuron
A3 = X + A5                        }
X  = A8 * A3                       } pattern of the 4-th neuron
A4 = X  * A5                       }
PRINTLINE 1                        instructions for printing of results
PRINT T, 4
PRINT A1, 1
PRINT A2, 1
PRINT A3, 1
PRINT A4, 1
PRINT A5, 1
PRINT A6, 1
PRINT A7, 1
```

PRINT A8, 1	the last instruction for printing
END T	the end of the basic simulation cycle
STOP	the concluding action (only stop of the computer)
START 1	the end sign of the program according to the rules of MOST

The first line introduces the variables used in program, i.e. X as an auxiliary variable, T as the variable where time information is stored, D identifying the end of the simulation and eight variables $A1$, $A2$, ..., $A8$. The second line and the third line are necessary because of the syntax of the language MOST; then the initial section follows which is composed in our example of the only one instruction, therefore for reading of the value of the simulated time determining the end of the simulation. Then the main cycle of the simulation follows; at the end there are the instructions which print in every step a new line of the results so that in every line there is the value of the actual simulated time and then the output values for each neuron are printed. The integer following the comma in every printing instruction determines the length of the printed information. The end of the main simulation cycle is indicated by the word *END* followed by the variable which controles the cycle (T in our example). Then the conclusion of the simulation follows (in our case it is only stop) and then a necessary indication that the program description is finished. The asterisc denotes the multiplication. The informations perforated at the input medium are the following: the first one is an integer determining the last value of the time when it is to be simulated. Then the values 0 and 1 follow representing the information which enters the system through the generator enumerated by 8. Every value corresponds to the entering information in one step. The entering values are printed at the end of every line during the simulation (as they are identical with $A8$).

If we wish to program the simulation of the same logical network using the second type of order, we can use the same ordering of the program patterns corresponding to different neurons; but for the illustration we present here another ordering, which would not be possible for the application in the first type of order. The program can be made in two forms; the first one is the following:

INTEGER X, T, D, A8, B8	heading of the program
LABEL 1	
BEGIN	
1: *READ D*	initial section
FOR T = 0 *STEP* 1 *UNTIL D*	heagind of the basic simulation cycle
READ A8	pattern of the 8-th neuron
$A7 = 1 - A4$	pattern of the 7-th neuron
$B1 = A8 * A7$	pattern of the 1-st neuron
$A5 = 1 - A6$	pattern of the 5-th neuron
$B6 = A1$	pattern of the 6-th neuron
$A2 = 1 - A1$	pattern of the 2-nd neuron

```
X  = A2 * A5
X  = X + A2          } pattern of the 3-rd neuron
A3 = X + A5
X  = A8 * A3
B4 = X  * A5         } pattern of the 4-th neuron
PRINTLINE 1          the instructions for printing
PRINT T, 3
PRINT A1, 1
PRINT A2, 1
PRINT A3, 1
PRINT A4, 1
PRINT A5, 1
PRINT A6, 1
PRINT A7, 1
PRINT A8, 1
A1 = B1              the transfers of the values for
A4 = B4              the neurons with delay
A6 = B6
END T                the end of the basic simulation cycle
STOP                 conclusion
START 1              the end sign of the program
```

The second form of the program is the following (compare with the discussion presented in the paragraph 2.5):

```
INTEGER X, T, D, A8, B8, J     heading of the program
LABEL 1
BEGIN
1: READ D                      initial section
FOR T = 0 STEP 1 UNTIL D       heading of the basic simulation cycle
READ A8                        } pattern for the 8-the neuron
B8 = A8                        }
A7 = 1 - A4                    } pattern for the 7-th neuron
B7 = A7                        }
B1 = A8 * A7                   pattern for the 1-st neuron
A5 = 1 - A6                    } pattern for the 5-th neuron
B5 = A5                        }
B6 = A1                        pattern for the 6-th neuron
A2 = 1 - A1                    } pattern for the 2-nd neuron
B2 = A2                        }
```

$X = A2 * A5$	pattern for the 3-rd neuron
$X = X + A2$	
$A3 = X + A5$	
$B3 = A3$	
$X = A8 * A3$	pattern for the 4-th neuron
$B4 = X * A5$	
PRINTLINE 1	the instructions for printing
PRINT T, 4	
*PRINT A*1, 1	
*PRINT A*2, 1	
*PRINT A*3, 1	
*PRINT A*4, 1	
*PRINT A*5, 1	
*PRINT A*6, 1	
*PRINT A*7, 1	
*PRINT A*8, 1	
FOR I = 1 *STEP* 1 *UNTIL* 8	the heading of the transfer cycle
AJ = *BJ*	the body of the transfer cycle
END J	the end of the transfer cycle
END T	the end of the basic simulation cycle
STOP	conclusion
START 1	the end sign of program

Note. We have used the language MOST as it is a typical first generation language and as the presented examples have been realized in the same languages in a real computer ODRA 1013 in the Biophysical Institute of Charles University. The other illustrations presented in the second part of this paper have been so formulated that they might represent typical facilities of the first generation algorithmic programming languages. There is no real language corresponding that illustrations, which would be implemented in any computer but it is easy to translate the presented illustrations in any implemented language.

(Received February 24, 1972.)

REFERENCES

The list of references will be presented in the part III.

Programovací prostředky pro simulaci logických sítí I

EVŽEN KINDLER

Článek podává informace o obecných rysech konstrukce simulačních programů pro logické sítě. Informace jsou formulovány tak, aby byly přístupny i uživatelům samočinných počítačů, kteří nejsou profesionálními programátory. Tato část obsahuje metodiku programování modelů logických sítí na počítačích první generace, s použitím algoritmicky orientovaných jazyků. Využití prostředků typických pro druhou a třetí generaci počítačů bude podáno v následujících částech.

PhDr. RNDr. Evžen Kindler CSs.; Biofyzikální ústav Fakulty všeobecného lékařství Karlovy university (Biophysical Institute, Faculty of General Medicine, Charles University), Salmovská 3, Praha 2.