Evžen Kindler

Programming means for simulation of logical networks. II

Kybernetika, Vol. 9 (1973), No. 1, (11)--29

Persistent URL: http://dml.cz/dmlcz/124653

KYBERNETIKA — VOLUME 9 (1973), NUMBER 1

Programming Means for Simulation of Logical Networks II

EVŽEN KINDLER

The serie of papers contains the information about programming means which enable to program the models of logical networks also for the non-computer-oriented users. The present part is directed for the second generation computers. There is presented a simulation programming system oriented to the considered class of problems — its language and compiler into an algorithmical language. A description of a programming methodics follows which enables to use the system COSMO for simulation of logical networks though it has been originally determined to continuous problems.

3. SIMULATION OF LOGICAL NETWORKS AT SECOND GENERATION COMPUTERS

3.1. The first generation of automatic computers is represented by a lot of computers in the first decade of years following the invention of the first automatic computer. One can approximate the decade by the years 1950–1960. The development of other facilities has caused that a new type of computers was studied, which has got the foundations for the concept of the *second generation* automatic computer. We have used the word *type* for the reason that there are much computers which have some facilities and some properties of the second generation of the computers while the other are from the first generation. There are even some computers with certain aspects familiar with the third generation of computers through their basic facilities are mixed of the both preceding generations. Therefore one must consider the following statements as a basis which can be used for every computer proportionally with its properties consistent with those that are accepted as typical for the second generation. Nevertheless the computers produced between the years 1960 and 1970 satisfy a lot of the properties of the type of the second generation computers so that the concept of them is accepted as a rather clear one by the people specialized for the computing technique: the concept is as clear as much concepts used e.g. by biologists in their systematics.

The computers of the second generation have the same properties as those of the first generation, concerning the length of words by them processed (they are of constant length, similar to the length in the first generation). The circuits of the computers of the second generation are based no more at the electron tubes but at transistors. The main internal memory is realized as a core memory while the drums serve as auxiliary memory which is not directly used in the fast arithmetic and control operations. The core memory has its capacity of about 10 000 words (thus about 10 times greater than the capacity of internal storage in the first generation) and its short access time causes that the computers of the second generations perform tens of thousands arithmetic operations in a second. The capacity of the auxiliary memory mediums is almost unlimitted: beside the mentioned drums there are the magnetic tape units. The computers are so constructed that one can join new memory units or input-output ones, too. The second generation computers are facilitated by the operating systems which prefer the closed shop mode of the computer run: the users give their demands in a form of a packet of punched cards (decks) containing the programs, the data and the control cards which cause the demanded use of them. The control cards contain also the information about the personality of the user. The decks are transformed into a magnetic tape which is given to the computer as the input file. The operating system reads the tape as its input data and according to the control cards it switches various programs to satisfy the domanded computation. It produces another file, so called output file, which is recorded during the computation at another magnetic tape. When one computation is finished the operating system reads immediately the following information at the input file so that the computer does not wait. Its time is well used (excepting certain situations described in the part concerning the third generation computers). The output file is taken down after a long time and then it is let to be print; after printing a tape of paper with text is received containing headings which can be well distinguished. The operators break the paper tape before such headings: as the headings contain also the names of the users (known by the computer because of the control cards) one can easily find his proper part of prints: there one can read the results, completed by notes of the operating system in case of an error; other complements are also present, as the price of computation, used memory, the time when the computation was performed etc. The operating system performs naturally all evidence concerning accounts.

3.2. The operating system can call in use the systems of automatic programming. They are stored in a magnetic tape of the operating system and they can be used by putting one control card into the deck of the user. The systems of automatic programming are either those of algorithmic type or those of simulation type. The first ones give nothing new for our subject, as they are only very powerful modifications of the algorithmic languages for the first generation computers. The *simulation programming systems* however carry new facilities in, even in the problem of simulation of logical networks: the simulation programming systems work similarly as the algorithmic ones (the program is described in so called programming language, perforated and the translator or the compiler reading it, produces the program in machine code), but the main difference is in the programming language in which the program is described; while the algorithmic languages admit only that one describes the sequence of demanded operations in more "readable" from than it would be written in the machine code, in the simulation programming language we describe the demands to the computer in the following way: the structure of the investigated system is described, completed by eventual parameters, description of the situation of the system is joint (the initial conditions, the duration of the computation, the input information for the system etc.). The whole description is also completed by very simple information which interest us and in which form it is to be printed (there are suitable facilities to print tables, graphs, eventually to do a simple statistical analysis). The description of much simulation languages is presented e.g. in [11], [12]; in the last source there is a rich article [13] which serves as a survey in a great group of the simulation languages.

3.3. After considerations done in the preceding part we can design a simple programming system for simulation of logical networks. Let us mention that there have been presented various propositions concerning programming languages never realized: to propose a programming language is not difficult, but a relatively hard and trouble work is to realize the translator which might translate the texts written in the programming language into programs performable by a computer. In our case the translation from the proposed simulation language into a first generation algorithmic language can be described in few words, because in the first part the general properties of the target programs (risen by such a translation) have been sufficiently studied. The simulated system is described by a paragraph containing lines so that one line corresponds to one neuron. There the neuron is described so that its order number is given at the beginning of the line. Then two spaces follow after that a letter is put. It means the type of the described neuron: *G* means a generator (including inputs), *D* means a neuron with delay and *E* that without delay. The letter is immediately followed by a stroke, which is followed by a number. It is a code number for the function which is performed by the described neuron. We propose the following correspondence between the code numbers and the operations:

0 – logical zero (identically generated at the output),
1 – logical one (identically generated at the output),
2 – identical mapping of the information entering at the first input of the declared neuron,
3 – negation of the information entering at the first input of the declared neuron,
4 – conjunction,
5 – disjunction,
6 – implication,
7 – equivalence,

8 – symmetrical difference,
9 – Sheffer stroke,
10 – input unit,

etc. The numbers from 4 to 9 means the mentioned function at the first and second input information. The language with the compiler can be arranged so that if the number of inputs is greater than 2 the superfluous input informations are ignored. The similar situation can be similarly solved for the unary operations (code numbers 2 and 3) in case that the number of the described neuron is greater than one. Also for generators the same relations can exist: if there are inputs which would enter into the described generator they are ignored as they might not enter into the declared neuron. Let us mention that the conception of a generator of a constant and the conception of a neuron which reacts to any input information by generating a constant, which lead to some sophisticated formulations in the first part have their analogies which are however very comprehensible: thus one can write $G/1$ or $E/1$ without any error. Moreover it is possible to declare $G/10$ or $E/10$, it is possible to declare $D/10$. While $G/10$ and $E/10$ are equivalent, $D/10$ means that a value is read from the input medium but it comes into the simulated system with a delay of one step. Theoretically one could omit the letter G completely from the proposed language using always the latter E in place of it but it seems more suitable to let it: the language is therefore more readable and the complications of the compiler (translator) caused by it can be neglected.

After the code number a left parenthesis follows. In the same line the corresponding right parenthesis must occur. Between both of them a list of input is written: any input is identified by Ak where k is the order number of the neuron from which the input comes; where there are more than one inputs they are distinguished by commas. Thus the description of the logical network presented in the paragraph 2.11. is the following:

```
8 G/10
5 E/3(A6)
7 E/3(A4)
6 D/2(A1)
2 E/3(A1)
1 D/4(A8, A7)
3 E/5(A2, A5)
4 D/15(A8, A3, A5)
```

We suppose that the conjunction of three inputs is coded by 15. Let us mention that the form of the texts occuring in the figure 3 is different from that of the proposed language: e.g. $G8$ occuring in the figure 3 means the third neuron which is a generator while $G/10$ in the example just presented means that the generator performs the action coded by 10; its order number is written before the letter.

3.4. In the preceding paragraph the means for the description of the simulated network has been presented. Such a description forms thus a paragraph called *body*.

That name can be put before the description in a special line, forming a *key word* for the computer (see later). But if we wish to simulate really a neuron network we must describe the situation by means of other paragraphs that receives the computer together with the body. Thus before the body it is suitable to present a paragraph concerning the initial conditions, introduced by its key word which we propose as *INITIAL*. Before this paragraph it is to be a heading of the whole description; there the "global" information about the simulated system is to be presented, i.e. the number of neurons in the simulated network and – facultatively – eventual variables used in the description, so that one could use the algorithmic means (see further – in the description of the program sections). The number of neurons can be determined by a line (the initial line of the description for the computer):

NUMBER OF NEURONS k

where k is an integer. After the body it is suitable to put a paragraph which has at least two lines: the first one has its form

FORMAT $Ak_1\ Ak_2\ Ak_3\ \dots\ Ak_n$

It declares what is to be printed. Between any terms Ak_i and Ak_{i+1} more spaces can be present. The corresponding simulating program prints the heading which has the following form:

$T\ \ Ak_1\ Ak_2\ Ak_3\ \dots\ Ak_n$

and under each identifier of the heading the same program prints in every step the corresponding values of the identifier. The second line of the same paragraph has the following form: either

REPEAT WHILE g IS GREATER THAN h

or

FINISH WHEN g IS GREATER THAN h

where g and h are identifiers or constants and the line means clearly the duration of the simulation. It can be e.g.

FINISH WHEN T IS GREATER THAN 200
*REPEAT WHILE A*4 *IS GREATER THAN* 0

The first example determines the duration of 200 steps, the second one determines that the simulation must be finished when the fourth neuron emits logical one as the output information. If the concluding action of the simulation has not to be the only stop operation it can follow immediately the line determining the duration of the simulation.

3.5. Thus a description of a simulation is a sequence of paragraphs initiated each by its key word: *NUMBER, BODY, INITIAL, FORMAT. REPEAT* or *FINISHED*. Each paragraph is clearly defined also for computer processing as an empty line (or in the card mediums: an empty card) is present. The same signal follows the last paragraph so that the end of the description is determined also clearly. Let us present the whole description of the example presented in 2.11.:

```
NUMBER OF NEURONS 8
INITIAL
READ D
BODY
8 G/10
5 E/3(A6)
7 E/3(A4)
6 D/2(A1)
2 E/3(A1)
1 D/4(A8, A7)
3 E/5(A2, A5)
4 D/15(A8, A3, A5)
FORMAT A1 A2 A3 A4 A5 A6 A7 A8
FINISH WHEN T IS GREATER THAN D
```

3.6. The presented means are suitable for a pure simulation of pure logical networks giving a table as the printed results. It is suitable for the specialists who are not trained in usual programming techniques. But there are specialists who would like to join certain actions specially prepared by them which are exceptional (e.g. special processing of simulated values). Such specialists must know to determine they wishes algorithmically, thus they must be educated in the algorithmic programming (i.e. they must be programmers for the first generation computers; for the third generation means the problem is completely different – see the fifth paragraph). The proposed simulation language would be obliged to have facilities for easy synthesis of the simulation means and the algorithmic ones so that the non-computer-oriented users might use only the first means without complications. The non-standard actions performed before the simulation can be simply programmed in the initial conditions; they need no additional rules. The same situation is for the affairs concerning the actions performed at the end of the simulation: the concluding action itself is an algorithmic component. If one wish to let an action perform in every step of the simulation he can join a paragraph headed *PROGRAM* containing the algorithmic description of the demanded affaire. That paragraph can be put between the body and the last paragraph or between the initial conditions and the body; the first ordering causes that the algorithm is performed at the end of every step, the second ordering means that the algorithm is performed at the beginning of every step. We can admit to let perform an algorithm at the beginning of every step and another algorithm at the end of every step by writing both the patterns of the described algorithms at the corresponding places; one can even admit to let perform algorithmic actions among segments of the step: thus the body can be divided into more paragraphs (each headed by *BODY*) and among them the paragraphs headed *PROGRAM* can be present. For the compiler (see further) it does no obstacles. Let us note that in the algorithmic parts one can need new integers, new labels etc., eventually newly identified by a letter. Thus – according to the used computer – it would be necessary to introduce the new variables in the heading. Thus in the example presented in 3.5. it would be the following heading:

```
NUMBER OF NEURONS 8
REAL D
```

Let us note that the only variables introduced automatically by the simulation system are *T* for time and *A*1, *A*2, *A*3, ... for the informations going through the system (see the second part).

3.7. Let us describe the compiler (or the translator) which reads the perforated descriptions of the simulations and generates an algorithmic program according to them. In other words, in this paragraph the ideas of a program are presented which translates the programs written in special second generation language for the simulation of logical networks, into first generation algorithmic language so that the target program performs the demanded simulation.

The translator is switched in several states. The initial state is called *INIT*. When the compiler is in it it reads the first line of the processed text. It omits the letters, stores the integer which follows them in the address *n* of the translator. The same translator then generates the first line of the target algorithmic program: it is a line *INTEGER T*, *An* where after the letter *A* the integer stored at the address *n* is reproduced. Then the translator assign one at its address *k* and is switched into the state *COPY*.

In the state *COPY* the translator reads the following lines of the source text and reproduces them into the taget text until an empty line is read. Then the state of the translator is switched according to the address *k*: if its contents is 1 the new state is *HEAD*; if it is 2 the new state is *BODY*; if it is 3 the new state is *END*. Let us describe the mentioned stated:

In the state *HEAD* the instructions

```
   T = 0
   GO TO 2
1: T = T
```

are generated. The last instruction is a dumb one indicating only the point of the algorithm labelled by 1:, where it is to jump from the further parts of the program. The instruction *GO TO* 2 prepares the printing of the heading during the simulation (see the description of the state *BODY*). After the generating of the instructions the contents of *k* is modified to be 2 and the translator is switched into the state *BODY*.

In the state *BODY* the translator reads the following line; if the first symbol is *P* the read line contains the key word *PROGRAM* and the translator is switched into the state *COPY*. If the first symbol is *F* the read line contains the key word *FORMAT* and the following instructions are generated in the target text:

```
   GO TO 3
2: PRINTLINE(...)
   GO TO 1
3: PRINT T,5
```

The first presented instruction causes the jump to the last presented instruction; therefore the second instruction and the third one are omitted. They are performed after the initial conditions, because they cause printing of the heading. The three points in the parentheses in the second instructions mean the text which is to be printed; it is the contents of the line just read from the source text, where the first word *FORMAT* is replaced by the word *TIME*. Then the same text (excepting the word *TIME* or *FORMAT*) is once more processed: the following text units *Ar* are put into instructions *PRINT Ar*,1; thus the instructions for printing the values indicated in the heading are generated and sent to the target program. The instructions can be completed by eventual instructions for spaces, if the terms of the heading are too distant (the generating of them is evident, they rise by a mere counting of spaces). After such a processing of the last term an instruction $T = T + 1$ is generated and the following line of the source text is read. If it has a form

```
REPEAT WHILE g IS GREATER THAN h
```

the instruction

```
GO TO 1 IF g > h
```

is generated into the target program. If the read line has a form

```
FINISH WHEN g IS GREATER THAN h
```

the instruction

```
GO TO 1 IF g ≦ h
```

is generated into the target program. Then the contents of the address *k* is modified to be 3 and the translator is switched into the state *COPY*.

Let us return to the begin of the state *BODY*. In the line just read the letter *B* can be present as the first symbol. In this case the line contents the key word *BODY* and the state works by the following way: the next line of the source text is read; if it is empty the translator jumps at the begin of the work in the state *BODY*. Otherwise the first number (integer) is read and stored at the address *e*. The following two spaces are omitting and the next letter is stored at the address *d*. The following symbol – the stroke – is omitted and the next integer is stored at the address *c*. Then the terms in the following parentheses – if they exist – are stored at the addresses $b1, b2, b3, \ldots, bf$ where *f* is stored at the address which we can call also *f*. If there are not terms in the parentheses in the processed line, the value of *f* is zero. From the contents of the addresses $b1, b2, b3, \ldots, c, d, e, f$ the instructions of the target program are generated; it is a special problem which will be discussed in the following paragraph.

The state *END* of the translator causes that the instruction *STOP* of the target program is generated, followed by the signal of the end of the program; the translator is switched into the state *INIT* and the translation is finished.

3.8. Let us consider the action of the state *BODY* of the compiler in case of processing of the paragraph body of the source text. Every non-empty line of that paragraph is processed by the same way, the beginning of which has been described in the preceding paragraph. Let us assume that the addresses $b1$, $b2, \ldots, bf$, c, f, d and e corresponding to a line are assigned by the values. The translator generates from them a pattern of the program corresponding to the declared neuron. The pattern both the work of the translator can reflect the method of programming used in the target program. That method can be distinguished by the type of ordering or by the form of the patterns. The type of ordering can be the first one or the second one, similarly as in the second part. The form of the patterns can be either *simple* or *composed*. The simple form maps each neuron into a pattern programmed similarly as in the second part. The terms in the parentheses, occuring in the source program line occur in the instruction for the operation performed by the neuron, as well as the term identifying the output information of the neuron. For example the declaration

$2\ E/4(A3, A4)$

is transformed into an instruction

$A2 = A3 \,.\, A4$

The second form of patterns – the composed one – organizes each pattern so that it contains a *kernel* which is invariant for the same operations (e.g. for all the conjuctions it is $X1 = X2 \,.\, X3$); this kernel is preceded by a sequence of assignement from the terms identifying the inputs into the declared neuron for the special introduced variables used in the kernel; the same kernel is followed by assignment of $X1$ for the variable which identifies the output information of the declared neuron. Thus the simple form of patterns implies that the work of the translator is rather complicated while the target programs are simple; the composed form of patterns causes that a simplification of the translator has place but the target programs are more expansive regarding the necessity of addresses and the run time during the simulation. The composed form of patterns has no analogy in the first generation computers because it would be evidently uneffective if one programs algorithmically by manual techniques. (In other words, the complication of translator has no important interpretation in the manual programming of algorithms.)

The complete number of combinations is therefore 4. We shall describe all of them regarding also the eventualities of the neurons with delay and those without it. In the paragraph 2.5 there have been presented various techniques of the programming in case of the second type of ordering; we use the last technique of them as it is evidently the most suitable one for to be automatized (see the discussion in the mentioned paragraph). The composed form of patterns needs always however introduction of auxiliary variables $X1, X2, \dots, Xn$. It can be realized in the state *INIT* of the translation so that the first line of the target program is enriched:

```
INTEGER T, An, Xn
```

The translation of the mentioned example

```
2 E/4(A3, A4)
```

gives the following pattern if the composed form is applied:

X2 = A3	assigning of the first input value
X3 = A4	assigning of the second input value
X1 = X2 . X3	the kernel fixed for all the code numbers 4
A2 = X1	assigning for the output value

Let us present the algorithms for each technique:

The composed patterns, the first type of ordering: For $i = 1, 2, \dots, f$ the instructions $Xj = bi$ are generated where $j = i + 1$, X and $=$ are the fixed symbols and bi is a term copied from the corresponding address. Then the kernel according to the contents of c is joined to the generated instructions. The kernel is followed by the instruction $Ae = X1$ where e denotes the copy of the contents of the address e.

The simple patterns, the first type of ordering:

The kernel is copied so that its fixed symbols are directly transferred into the target program while the "parameters" are replaced: the parameters in the kernel have the form *?i* where i is a natural number (intuitively – the i-th operand) and always when *?i* is to be copied it is replaced by the contents of the address bi. The kernel for the code number 4 has therefore the following form:

```
?e = ?1 . ?2
```

The kernel for the code number 5 has the following form, if using the binary operations only:

```
X  = ?1 . ?2
X  = ?1 − X
?e = ?2 + X
```

The interpretation of *?e* is similar: the letter A followed by the contents of e is copied instead *?e*.

Let us mention that the kernels are prepared in the translator by the people who have constructed it and the user have no necessity to meet them.

The composed patterns, the second type of ordering: The generation of the assignments of the input values is performed in the same way as in the technique "composed patterns, first type of ordering", as well as the copying of the kernel. After it the instruction $Be = X1$ is generated where $B1$, and $=$ are the fixed symbols, e denotes the copy of the contents of the address e. Then the translator tests the contents of the address *d*: if it corresponds to the letter *G* or *E* the instruction $Ae = X1$ is joined to the pattern, where *A*, *X*, 1 and = are the fixed symbols and *e* is the same as for the preceding instruction.

The simple patterns, the second type of ordering:

The translation is similar as in case of the first type of ordering for simple patterns; also the kernels prepared a priori in the translator are the same. The difference is in the interpretation of *?e*, which is replaced in the generated pattern as *Be* (while *?* followed by digits is replaced by the letter *A* as in the first type of ordering), and in the conclusion of the generation of the pattern: if the contents of *d* is the letter *E* or *G* an instruction is joined: $Ae = Be$ where *e* is interpreted as before.

Naturally the possibility of a slight simplification of the generating algorithms exists, which idea is to use the figure *§e* or *!e* instead of *?e* in the prepared kernels. Then the interpretation of the first symbol is determined by it uniquely and does not depend on the symbol which follows.

Of course the second type of ordering needs to introduce the variable *B*1, *B*2, ..., *Bn* which can be done in the state *INIT* of the translator that instead of *INTEGER T, An, Xn* the line *INTEGER T, An, Xn, Bn* is introduced in the case of the composed form of patterns, or – in the case od simple form of patterns – the line *INTEGER T, An, Bn* is generated instead of *INTEGER T, An*. Moreover the second type of ordering must generate the sequence of instructions which perform the following action

```
FOR I = 1 STEP 1 UNTIL n DO AI = BI
```

which is generated always after the last instruction for printing (thus before the instruction $T = T + 1$). For this purpose the variable *I* must be introduced at the beginning of the target program similarly as we have just mentioned about *B*1, ..., *Bn*.

3.9. We have proposed the language so that the translation of the body is rather simple. We could modified the expression means so that the simulated logical system would be more readable for the humans but the translation would be more complicated; e.g. it would be possible to write

```
NEURON 6 BINARY CONJUNCTION
ITS INPUTS: FROM NEURON 2, FROM NEURON 4
```

instead of

```
6 E/4(A2, A4)
```

or

```
DELAY 5 NEGATION
ITS INPUT: FROM NEURON 5
```

instead of

```
5 D/3(A5)
```

or

GENERATOR 8 *READ TAPE*

instead of

8 *G*/10

but it is clear that the translation is complicated only by omitting the redundant symbols and by replacing certain symbols by other ones (in case of code numbers by a searching in a vocabulary), while other expression means are ordered in another way that in the original proposal. Both the possibilities have their proper advantages and disadvantages and it is difficult to decide which possibility is better (compare e.g. with the versions of CSMP for various machines IBM which have similarly distincted their languages although even the names of the programming system are the same — see [13], [14]).

4. SIMULATION OF LOGICAL NETWORKS BY A CONTINUOUS SIMULATION PROGRAMMING SYSTEM

4.1. The facility of simulation programming languages for the second generation computers has a great advantage for the users, who need not to express their demand algorithmically. Nevertheless any special language needs its corresponding translator which must be made and which needs a lot of a hard programming work. To do a universal second generation language leeds to algorithmical languages: but it returns to the first generation facilities. Thus a possibility is studied to use a simulation language, which has been designed for a certain class of problems, to another class of problems: in order to be efficient this affaire must satisfy one of the following properties:

either there is a methodics of programming which enables easy and clear expressing of the new problems demands in the programming language oriented for the original class of problems;

or a new programming language can be designed so that it is oriented to the new class of problems and that an automatic translation of it into an existing programming language for simulation of certain class of systems is more simple as the translation of the new programming language into an algorithmic one.

The problems to which the original programming language is oriented can be mathematically very different from those which ask the adaptation of it. We shall present an example how to use a simulation programming system for continuous problems with certain specilization, in case that it is to simulate the logical networks. The example is important not only for the reason that it forms certain external analogy of the representation of the neuron networks by electrochemical continuous systems in living organisms but also for its implementation: the continuous simulation system has been implemented in the Biophysical Institute of Charles University in Prague for problems of radiation biology and nuclear medicine, without regard's to the neurophysiology, but then it has been discovered as appliable also in the last branch.

4.2. The continuous simulation system is called COSMO which is an abbreviation for the words Compartmental System Modelling. The system is implemented for the computer ODRA 1013 (see [10]).

The system COSMO has been realized in the years 1967–1969 and it is still tested to satisfy all the possible demands risen in the original array of problems (see [15], [16], [17], [18], [19] and [20]). It is oriented to the class of simulations of *compartmental systems.* They are abstract systems risen by an idealization from the real hydrodynamical networks. *Compartment* is a vessel where a liquid is present, mixed homogeneously with certain substance, called *tracer* (e.g. a coloring substance, a radiactive isotope). Into the vessel the same liquid comes containing the same tracer but generally in another concentration. The entering liquid is supposed to be immediately well mixed with the liquid which has remained in the vessel (it is the first idealization). The liquid can go from any compartment into any compartment through the channels; the size of the liquid which takes place in the channels is neglected (another idealization). The flow rates in the channels can vary during the time and the volumes of the compartments as well, though the greatest part of the problems solved nowadays concerns the compartmental system with constant parameters (see [21], [22]).

Using the language COSMO one describes the simulated compartmental system in several paragraphs: each of them corresponds to a compartment; in the first line of such a paragraph one writes the order number n of the described c ›mpartment by a line

COMPARTMENT n

which is followed by k lines each of which determines one input of the described compartment; a line can have a form

FROM COMPARTMENT $r = e$

where the integer r is the order number of the compartment from which the substance comes and e is an arithmetical expression determining the size which comes in the actual step. When all the input channels have been described the output is determined globally for the compartment (without eventual branching after leaving the compartment) in a line:

ITS OUTPUT $= e'$

where e' is an arithmentic expression with the analogous meaning as e. The last line determines the volume of the compartment:

ITS VOLUME $= e''$

There are various facilities to express the most probably properties by the human words (without foımulas) or to omit them at all. All these possibilities are presented in [16]. Let us only mention those which are important for our subject:

The time can proceed by varying steps; if the situation is "normal" (for us: if step is equal to one) one must write the paragraph containing the only word *TIME*;

The variable concerning the k-th compartment are Vk for its volume, Gk for its input, Kk for its output (thus $G4$ means the size of the liquid which enters into the fourth compartment in the actual step). The identification of tracer is given by putting the letter H before the identifier (thus $HG4$ means the size of the tracer which enters the fourth compartment during the actual step).

To define a value which is more complicated that one arithmetic expression can be done so that instead of the sign of equality a colon is written and then the defining program follows, where the defined values is identified by X. Thus the declaration

```
ITS OUTPUT = 3·245 + K3 − K2/3
```

is equivalent to

```
ITS OUTPUT:
X = K2/3
X = K3 − X
X = 3·245 + X
```

The values Vk are assigned in the beginning by one while the other value are assigned by zero.

The facilities for initial condition, for defining the duration of the simulation and for the first information about the simulated system (the heading) are the same as in the language designed in the preceding part. The description of printing tables forms a special paragraph of the following form:

```
FORMAT
text, i.e. the heading of the table
asteriscs determining the positions of digits in the printed table.
```

In the table an asterisc corresponds to a digit, a space to a space and a decimal point to the same decimal point. E.g.

```
FORMAT
 T   V3   HV5   HK7
***  **  .***  .***
```

In the language for the logical network simulation the last line is not necessary because the printed values are only one and zero.

4.3. To use the system COSMO for programming the neuron network simulation brings immediately various advantages if we compare it with the programming in the algorithmic languages introduced in the part 2. Thus the heading of the description, containing the introducing of the number of compartments (neurons) can be used without necessity of any modification, as well as the paragraphs for the initial conditions and for the concluding section (here only another type of identifiers must be written — without Aj it is to be Vj, as we shall demonstrate further). We can use also the paragraph for the prints of the simulated results according to the rules of COSMO: though it is a bit complicated if compared with the format-line described in the preceding part, it causes no complication for the non-computer-oriented users. If one wish to print the values of the time he must express it in the paragraph for the printing of the results and befero it he must introduce a new paragraph containing the only word *TIME*. Concerning the description of the neurons one must present special methodics of programming:

4.4. The language COSMO has been designed for the problems concerning of the mixing and the transport of matter, which satisfies certain physical laws. The processing of information which takes place in the logical networks satisfies other laws. Thus one must describe the neurons as compartments with relations between input and output adequate to those which hold in the neurons. Our situation is simplified because of the absence of the tracer. Thus we can formulate a general rule for programming any neuron which performs a function $f(Vi_1, Vi_2, \dots, Vi_s)$ of the input information comming from the neurons with their order numbers $i_1, i_2, \dots, i_s$, be the following way:

COMPARTMENT h
FROM COMPARTMENT h $= f(Vi_1, Vi_2, \dots, Vi_s)$
ITS OUTPUT = Gh
ITS VOLUME = Gh

where the first line introduces the order number of the declared neuron, the second line contains formally the left hand side of the assignment (it can be placed any integer less than the number of all neurons in the simulated system instead of h in it) while the right hand side contains the function which is to be performed by the declared neuron; the last two lines must be written as they are recommended. The relations between the input and the output, regarding to the volume, do not satisfy generally the law of conservation of matter but it causes no obstacles. If the function f is more complicated so that it cannot be programmed in one expression instead of the second line the following segment can be present:

FROM COMPARTMENT h:
program which computes $f(Vi_1, Vi_2, \dots, Vi_s)$ *assigning it for X*

Let us present an example how to program a neuron enumerated by 5 which performs the disjunction of the values comming from the neurons enumerated by 2 and 3:

COMPARTMENT 5
FROM COMPARTMENT 5 $= V2 + V3 - V2 . V3$
ITS OUTPUT = G5
ITS VOLUME = G5

or — as it has been implemented for the computer ODRA:

```
COMPARTMENT 5
FROM COMPARTMENT 5:
X = V2 * V3
X = V2 − X
X = X + V3
```

We can program in the same way also the generators, including inputs; the system COSMO has however certain facilities for describing the inputs into the simulated system. We shall illustrate how to have use of them in programming of generators but we do not recommend it as it complicates the programming methodics. The description of a generator performing a function g of time (which can be naturally a fictive argument) can be formed either as

```
IN-COMPARTMENT h
ITS FLOW = g(T)
```

or as

```
IN-COMPARTMENT h
ITS FLOW:
program which computes g(T) assigning it for X
```

Naturally the same element is described using the preceding meens (by means of *COMPARTMENT* instead of *IN-COMPARTMENT*) where instead the function f the function $g(T)$ is written.

4.5. The system COSMO admits only the first type of order. Thus the problems with transfers between *Ai* and *Bi* introduced in the preceding chapters has no analogy here.

In COSMO one can omit all the redundant letters. Thus one can write *F* instead of *FROM COMPARTMENT*, *C* instead of *COMPARTMENT* etc. One can change the redundant letters for any other letters and spaces (to complete the description by comments). The texts can be joined after the order number introduced in the first line. We present the description of the example of 2.11 programed in COSMO; at the left hand side there are used all the facilities for comments while at the right hand side there are the corresponding texts written in the most short form:

```
NUMBER OF NEURONS 8            N8
REAL E                         REAL E

INITIAL CONDITIONS             I
READ E                         READ E

COMPARTMENT 8 READER           C8
FROM COMPARTMENT 8:            F8:
READ X                         READ X
ITS OUTPUT = G8                ITS O = G8
ITS VOLUME = G8                ITS V = G8
```

```
COMPARTMENT 5 NEGATION                      C5
FROM COMPARTMENT 5 = 1 - V6                 F5 = 1 - V6
ITS OUTPUT = G5                             ITS O = G5
ITS VOLUME = G5                             ITS V = G5

COMPARTMENT 7 NEGATION                      C7
FROM COMPARTMENT 7 = 1 - V4                 F7 = 1 - V4
ITS OUTPUT = G7                             ITS O = G7
ITS VOLUME = G7                             ITS V = G7

COMPARTMENT 6 IDENTICAL                     C6
FROM COMPARTMENT 6 = V1                     F6 = V1
ITS OUTPUT = G6                             ITS O = G6
ITS VOLUME = G6                             ITS V = G6

COMPARTMENT 2 NEGATION                      C2
FROM COMPARTMENT 2 = 1 - V1                 F2 = 1 - V1
ITS OUTPUT= G2                              ITS O = G2
ITS VOLUME = G2                             ITS V = G2

COMPARTMENT 1 CONJUNCTION                   C1
FROM COMPARTMENT 1 = V8 * V7                F1 = V8 * V7
ITS OUTPUT = G1                             ITS O = G1
ITS VOLUME = G1                             ITS V = G1

COMPARTMENT 3 DISJUNCTION                   C3
FROM COMPARTMENT 3:                         F3:
X = V2 * V5                                 X = V2 * V5
X = V2 - X                                  X = V2 - X
X = V5 + X                                  X = V5 + X
ITS OUTPUT = G3                             ITS O = G3
ITS VOLUME = G3                             ITS V = G3

COMPARTMENT 4 CONJUNCTION OF THREE          C4
FROM COMPARTMENT 4:                         F4:
X = V8 * V3                                 X = V8 * V3
X = X * V5                                  X = X * V5
ITS OUTPUT = G4                             ITS O = G4
ITS VOLUME = G4                             ITS V = G4

TIME                                        T

FORMAT
  T   V1  V2  V3  V4  V5  V6  V7  V8
 ***   *   *   *   *   *   *   *   *
FINISH WHEN T IS GREATER THAN E
```

The last four lines (beginning from FORMAT) cannot be sufficiently shortened.

4.6. Though the methodics for programming the simulation of logical networks in COSMO is very simple we can give another programming tool which is more suitable for the non-computer-oriented users. This mean is a programming language similar to the special programming language for the simulation of logical networks presented in the preceding paragraph. We present another one, which is from the original slightly different but which can be simply translated into COSMO. The differences from the original one are the following: only the first type of ordering is admitted, the letter *A* used in the identifiers is here replaced by the letter *V* and the prints are declared in a special paragraph which must satisfy the rules for the format paragraph in COSMO. We see that the modifications cannot carry any obstacles for any user.

The translator from the described language into COSMO is more simple that the translator from the simulation language proposed in the third part into an algorithmic language: the global logics of the translator gets a mere copying: the translator copies the paragraphs for prints, for finishing together with the conclusion, for initial conditions and the heading. Before copying of the last paragraph (or – equivalently – after copying the paragraph for the prints) the translator generates a new paragraph containing the only word *TIME*. One can compare the work of this translator with that described in 3.7.

Concerning the translation of the body, the logics of the translator is similar to that presented in 3.8 for the original translator, but certain details (concrete strings of symbols by the translator generated) differ from the original one. Similarly as in 3.8 we can assume that any line of the body has stored the essential informations in the addresses *b*1, *b*2, ..., *bf*, *c*, *d*, *e* and *f* of the translator. From each such a vector a paragraph of a target text (in COSMO) is generated which has the following form:

```
Ce                  or  COMPARTMENT e
Fe:                     FROM COMPARTMENT e:
program pattern         program pattern
ITS O = Ge              ITS OUTPUT = Ge
ITS V = Ge              ITS VOLUME = Ge
```

The left hand form is better because the target text does not come to any human being. The right hand form is presented here only for better comprehension of the reader.

Concerning the program pattern we can referenced all the discussions presented in 3.8 (limited naturally to the possibilities of the first type of order), because the program patterns in case of the present paragraph differ from those of 3.8 only so that instead the letter *A* in the identifiers the letter *V* is used.

The reader can imaginate the eventual modifications of the translator in case that we would receive COSMO texts from the language of the third part without any modifications (replacing all occurings of the letter *A* by the letter *V*, a slight translation of the format line).

The fact that the target language is COSMO permits us to introduce the facility of omitting redundant texts also in the paragraphs which are copied into COSMO.

(Received February 24, 1972.)

REFERENCES

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

PhDr. RNDr. Evžen Kindler, CSc., Biofysikální ústav FVL UK (Biophysical Institute, Charles University), Salmovská 3, 120 00 Praha 2, Czechoslovakia.