KYBERNETIKA ČÍSLO 4, ROČNÍK 5/1969

Simulation System COSMO

DESCRIPTION OF ITS LANGUAGE AND COMPILER

EVŽEN KINDLER

The detailed description of the simulation system COSMO of automatic programming is presented, i.e. syntactical rules of its language, interpretation of them in the physical universe and the algorithm of the compiler.

1. INTRODUCTION. THE DEVELOPMENT OF THE SYSTEM

The system COSMO has been developed in the Biophysical Institute of the Faculty of General Medicine at the Charles University in Prague. It has been reasoned by the situation in the investigation of the transport of substance in living organisms by tracer method: as the substance is generally unaccessible for measuring apparatus, some part of it is "labelled" — usually by radioactive isotopes — so that it can be detected and measured. From the transport of labelled substance one can make deduction of the transport of all substance. If one neglects the transport of labelled elements, he may be interested in behaviour of a system from the point of view of quantitative properties of the substance travelling through it, especially if volumes of the substance contained in certain parts of the system can influence one another: it represents a case when not only transport of substance but also a transport of information is important in the investigated system, which can be however investigated by methods proper to the transport.

The classical mathematical methods have been very limited and they have failed in more general necessary cases. The method of simulation on a digital computer has appeared as very useful.

The investigation can be branched into several groups:

1.1. The structure of the system has been known, but not numerical values of its parameters. Not only modelling but also optimizing of our hypotheses would be made.

1.2. The structure of the system has not been known. Thus one must often change the structures of his hypotheses before the acces to the true result.

1.3. The structure of the system is known and its parameters as well, but one need to know a prediction on the system's behaviour in certain conditions, eventually to appreciate it.

From all three points it has followed that one need a suitable and fast source of corresponding programs, i.e. a simulation system of automatic programming the language of which would be near to the expressing means of the doctors and biologists as much as possible, in order to

allow them an ease dialogue with the computer. Thus the system COSMO has risen. Its name is the abbreviation of Compartmental System Modelling.

Its language has been described originally so that one might easily recognize its potential application and the reason of its syntactical means. Simultaneously the compiler has been programmed for a small computer ODRA 1013. The description has not been exact in the mathematical meaning in order to be more readable for those who had not been acquainted with the system and for the specialists not interested in problems of the theory of programming. The author wished not to precise the description in order to admit stimuli of other institutes in the following time, before the compiler would have been cleared out of all its errors. The original description was delivered as a privat print of the Charles University [4] to other centers which are in a relation to nuclear medicine and biology. After processing their suggestions and namely after presenting the system in the IFIP Working Conference on Simulation Programming Languages in Oslo, May 1967 [1], which was a rich source of stimulations concerning applications and clearing of COSMO, the definitive description of the system has been risen; it is presented here. Thus also the suggestions of the specialists who have demanded an exact description of the system in order to exchange biological models and diagnostical programs are satisfied.

It has appeared that the system has been useful not only for the mentioned purposes. It can be applied also for pedagogical pourposes and for solving some problems of social sciences, because the mathematical properties of substance mixing interpreted in the semantics and in the compiler are very general. They are presented in the paragraph 2.1.1. Moreover, the numerical method used by the compiled programs permits having profit of its "inacurracy" for the research of the systems not satisfying the basic properties of the compartmental systems (see the hypotheses in 2.1.1), as it has been presented in [1] chapter 13.

Relating to the original language presented in the mentioned articles, the definitive one has been enriched by the following facilities:

i. generalization of the semantics of the declaration *TIME IS NORMAL*;
ii. proper means for instruction to print results in the form of graphs;
iii. the possibility of omitting the format declaration.

2. THE LANGUAGE COSMO

2.1 Preliminary notes

2.1.1. *Compartmental systems*

The semantics of the language has been usually defined as the interpretations of its syntax in the real universe. In the case of COSMO the part of the real universe where it is mapped can be described by compartmental systems. They are abstract objects where the important properties of the transport of substance are expressed. The substance is supposed as continuous, non-atomical. Every compartmental system is a complex of a finite number of compartments connected by channels (see [2], [3]). Each of the compartments can contain a certain volume of substance; the substance is transported from one compartment to others through the channels. Both the volumes and the flow rates through the channels can vary in the time: thus the case of external influences, of information flow and of the control inside the system can be interpreted. A part of substance can be a tracer, i.e. it carries an at-

tribute measured in proper units. For a so called opened system, which does an exchange of the substance with its environment, its inputs and outputs are considered as degenerated compartments. They are called in-compartments and out-compartments. The words input and output are reserved for the channels concerning a compartment: input into a compartment, in-compartment into a system etc.

The substantial properties of the compartmental systems can be expressed in the following methodological hypotheses:

2.1.1.1. The measure of the substance occuring in the channels can be neglected.

2.1.1.2. The substance which has remained in a compartment is immediately mixed with the substance entering it.

2.1.1.3. The proper change of the measure of the tracer, not caused by mixing and entering new tracer through inputs and in-compartments, is proportional to the measure of the tracer; it concerns every place of the system, namely each compartment. The coefficient of proportionality can vary in the time but is the same for all places of the system at the same moment.

One can express the quantitative properties of the compartmental systems by the following equalities*

$$V_i = V_{i,0} + \int_{t_0}^{t} \left(\sum_j r_j^i - \sum_j r_i^j\right) \mathrm{d}t \,, \tag{2.1.1.4}$$

$$H_i = H_{i,0} + \int_{t_0}^{t} \left(\sum_j \frac{H_j r_j^i}{V_j} - \frac{H_i}{V_i} \sum_j r_i^j + BH_i\right) \mathrm{d}t \tag{2.1.1.5}$$

where

H_i is the measure of tracer contained in the i-th compartment,
$H_{i,0}$ is its initial value at the time t_0,
V_i is the volume of the substance contained in the i-th compartment,
$V_{i,0}$ is its initial value at the time t_0,
r_j^i is the flow rate of the substance which goes from the j-th compartment into the i-th compartment,
B is a quotient of the proper change of the tracer.

All of the presented terms can vary in the time.

One can define mathematically the halftime of the tracer by the expression

$$-\frac{\lg 2}{B} \,. \tag{2.1.1.6}$$

It has practical importance only if B is a negative constant.

* Analogous description of compartmental system by means of differential equations is presented in [8].

Note. The tracer has been originally a radioactive substance, its proper change being the fission. In COSMO the tracer can represent any attribute of the individuals forming the transported substance, e.g. prise of products, sex or another category of persons. The individuals themselves are, however, interpreted neither in the compartmental systems nor in COSMO.

2.1.2. *The time of the system*

If one investigate a real system, two important classes of approach can be considered:

2.1.2.1. Measuring instruments – in the most general sense – are connected with certain places of the system; they give us an information of the situation in the corresponding places in distinguished time moments; the information can be expressed by a matrix where every line corresponds to a time moment and every column corresponds to a certain place of the system; or the same information can form a set of curves in a coordinate system, where a curve corresponds to a certain place of the system and an axis corresponds to the time.

2.1.2.2. Measuring instruments are applied not at the same moment in all important places. The uniform method of measuring is left, eventually the information is led to a center, where it is partially processed and a more complex information of the system is displayed.

Contrary to the first method, there is not possible to describe à priori a general form of the information displayed by the second method.

In the system COSMO, the images of these two classes exist. Time of the modelled system is modelled by the discrete time of its model. The time steps can be constant or not. For obtaining the information in a form of a matrix or a set of curves, special declarations are established à priori in COSMO. For other forms of result displaying, algorithmical means have been built in the system COSMO. Their interpretation is similar as in the case of traditional algorithmic languages. As they own usual assignment statements, jumps and cycles, they can be used by a more ingenious programmer not only for non-standart prints but also for completing modelling by other procedures.

2.1.3. *Metalanguage*

The language COSMO uses one sort of letters. They are printed as block letters in this article. The metalanguage is distinguished so that it is printed in minor italics. Every metalinguistical mean – therefore matalinguistical variables and signs à priori defined – are represented by strings of letters eventually dashes: thus one word or several ones joined by a dash form one "word" of the metalinguistical definition "phrases". The space can be only between two metalinguistical variables: it signifies that a string representing the first variable is to be followed by a string representing the second one in order to form a representation of the metalinguistical variable

newly defined. Comparing with the traditional definition used e.g. for ALGOL 60 [5] we can write that the brackets ⟨and⟩ have been omitted and the space has replaced a couple ⟩⟨. The phrase structure grammar is expressed similarly as in [5], using the words *is* instead of a sign ::= and *or* instead of |. Moreover, we use à priori declared words *empty* for an empty text and *iterated* and ***sequenced*** for sequences – non-empty and possibly empty respectively; exactly, let q be any metalinguistical variable; then

q-iterated is q or q-iterated q,
q-sequenced is empty or q-sequenced q.

The second definition can be expressed equivalently as

q-sequenced is empty or q-iterated.

In the following paragraph, the algorithmic components of the language COSMO are defined simultaneously with the target language. It is used in the description of the compiler from COSMO. It is an algorithmic language of the size of a simple autocode. Thus the described compiler can be realized – almost mechanically – for any computer which has had a traditional algorithmic system of automatic programming the language of which contains the target language. If we neglect some minor modifications of the languages – interpreted as slight modifications in the described compiler – we can state that the target language is a part of any habitual algorithmic language.

The algorithmic components of COSMO are distinguished from the target language often only by a difference of the set of permitted identifiers. If a metalinguistical variable of COSMO is named e.g. q a metalinguistical variable of the target language with the corresponding semantics is named *tar-q*.

The space character of the both defined languages is always noted in the definitions by the word *space*. It has an importance in the both languages. There is another important sign noted *es* which is an abbreviation of end sign; it is a special character. In the hardware representation of COSMO in the place of its origine it is represented by the teletype signal of a new line (the carriage return signal is there interpreted as a symbol without syntactical importance). This representation admits to express certain syntactical units in very readable form, using lines and paragraphs. For other input mediums than teletype we advice using a new punch card signal, semicolon, punch in special column of the cards etc. It may be also useful to represent the first *es* by another signal as the following in the case of *es-iterated*; the compiler must not distinguish them.

2.2 Algorithmic means

2.2.1. *Identifiers and constants*

2.2.1.1. *Syntax:*
digit is 0 *or* 1 *or* 2 *or* 3 *or* 4 *or* 5 *or* 6 *or* 7 *or* 8 *or* 9

integer is digit-iterated
real is digit-iterated · digit-sequenced
identifier is A or D or E or G or J or K or M or N or P or Q or S or T or U or V or W or X or Y or Z or HV or HK or HG
tar-identifier is A or B or C or D or E or F or G or H or I or J or K or L or M or N or O or P or Q or S or R or T or U or V or W or X or Y or Z
index is identifier or integer
tar-index is tar-identifier or integer
variable is identifier or identifier index
tar-variable is tar-identifier or tar-identifier tar-index
declared is identifier or identifier integer
tar-declared is tar-identifier or tar-identifier integer
type-declaration is REAL space declared-iterated or es or INTEGER space declared-iterated es
tar-type-declaration is REAL space tar-declared-iterated es or INTEGERspace tar-declared-iterated es

2.2.1.2. *Semantics.* All conceptions have been well known from the traditional algorithmic languages. The identifiers *V*, *G*, *K* followed by an index *i* represent the measure of substance which is contained, enters and leaves respectively from the compartment with its order number *i* (see 2.3.4.2), during the actual time interval. If one puts the letter *H* in front of them we obtain the identifiers representing no more the substance, but its labelled components, measured in proper units (they are however proportional to those in which the substance is measured). The identifiers *T* and *D* mean the time and the time step respectively. All these identifiers are à priori established as reals in COSMO and they cannot occur in the type declarations. The identifier *X* as well; it means an auxiliary real variable and in the declarative programs – see 2.3.1. – it represents the declared term. In the target language all used identifiers must be declared. We say that a variable is of the type real or integer if it has been declared or à priori established as real or integer respectively. The same predicate applied to numbers is clear.

In the declarations an identifier followed by an integer represents an array of variables with one index in the bounds from 1 to the presented integer. The variables *V*, *G*, *K*, *HV*, *HG*, *HK* represent arrays with the index up to the highest order number.

2.2.1.3. *Limitations.* The variables established à priori cannot occur in the declaration and the other can occur there once only. The identifier which is an index must be of the type integer. Analogous rules hold for the target language.

2.2.2. *Expressions*

2.2.2.1. *Syntax*:
operator is + or − or ∗ or /

unary is integer or real or variable
binary is unary operator unary
designator is ABS or SQRT or EXP or LN or SIN or COS or TAN or ARCSIN or ARCTAN or ENTIER or STAND
function is designator es unary
expression is unary or binary or function

If we put the preposition *tar-* to the metavariables *variable*, *unary*, *function* and *expression* in the syntactical rules above, we obtain the rules for the analogous syntactical means of the target language.

2.2.2.2. *Semantics* is clear. *ABS* means the forcing of a positive sign to the argument without change of the type. The function *ENTIER* is applied to an argument of the type real and the result is the nearest integer equal or less than the argument. The function *STAND* transforms the value of its argument which is of the integer type to the type real. The other functions, being applied to an argument of the real type gives a result of the same type. The sign * means the multiplication. The sign / means division; it can be applied only to the operands of the type real giving the results of the same type.

2.2.2.3. *Limitations.* Besides the limitations concerning the types which has been presented in the semantics, one cannot use two operands of different types in the same expression.

According to the rules which have been just presented, we generalize the conception of the type from the variable to the expression, due to its result. The type of a binary is equal to the type of its operands.

2.2.3. *Statements*

2.2.3.1. *Syntax*:
assignment is variable = expression es
jump is GO space TO space integer es
stop is STOP space index es
input is READ space variable es
branching is GO space TO space integer space integer space integer space IF space unary = unary es
output is PRINTLINE space index es or PRINTSPACE space index es or PRINT space unary form es or PRINTOUNT text

We do not describe here the form and the text in detaill as they may depend substantially on the physical properties of the output mediums of the computer. If we put the preposition *tar*-before the metalinguistical variables *assignment*, *variable*, *expression*, *jump*, *index*, *branching*, *stop*, *input* and *output*, we obtain the rules for the similar syntactical means of the target language. The following rules are:

instruction is assignment or jump or branching or stop or input or output
tar-cycle is FOR space tar-index = 1 space STEP space 1 space UNTIL space tar-index es tar-instruction-sequenced END es
tar-instruction is tar-assignment or tar-jump or tar-branching or tar-stop or tar-input or tar-output or tar-cycle
statement is instruction or integer: instruction
tar-statement is tar-instruction or integer: tar-instruction

2.2.3.2. *Semantics. Assignment* is an instruction maximally three-address with usual interpretation. The *jump* represents an interruption of the sequentional run of the instructions: the next instruction which is to be done has before it the integer separated by a colon – therefore a label – which is equal to the integer following the words *GO TO*. The instruction preceded by a label or not is called statement. *Branching* represents the possibility of three jumps: if the first unary of the relation at the end of the branching is greater, equal or less than the second one, the jump to the first, second or third label respectively is made. The labels before an instruction can be positive integers, for COSMO only greater than 4 (see 2.4). Zero can be present in jumps and branchings: it means always the next instruction. *Stop* means the stop of computer; after one has let it run the jump to the label following *STOP* is done. *PRINTLINE n* is an instruction for *n* new lines in print, *PRINTSPACE n* is an instruction for *n* spaces, *PRINTOUT* is an instruction for printing the following text. *PRINT unary* is an instruction for printing the value of the following unary in the form which follows. Input assigns the value read in the input medium to the variable presented after *READ*. The input medium is supposed à priori fixed and the form of representing it too. *Tar-cycle* repeats the set of instructions following its heading; the end of the set is noted by the word *END*; we have omitted the cycles for *COSMO* in order to eliminate the obstacles with the semantics of them if cycles established by statements might confuse with the cycle established a priori by modelling.

2.2.3.3. *Limitations.* The labels are not localized; one integer value – with the exception of zero – must have only one interpretation as a label before an instruction. Assignment of the value of one type to a variable of another type is not admitted.

2.2.4. *Programs*

2.2.4.1. *Syntax*:
program is statement-sequenced
tar-heading is es type-declaration-iterated es
tar-program is tar-heading tar-statement sequenced es

2.2.4.2. *Semantics.* Programs are used in COSMO as components forming initial conditions, final diagnostics, preparing constants during modelling and for unstandart prints of results. Their semantics will be fully clear only after having known the

semantics of the body of the model – see the paragraph 2.3. *Tar-program* is an algorithm described in a traditional form. All declarations must be before the statements.

2.3 Simulation means

2.3.1. *Common principles, declaratives*

The declarations of the parts of the model which is realized in the computer by the system COSMO use words for the declared things, which are called generally terms of the model. The entities by which is the term declared are represented by identifiers and constants which occur in expressions eventually in programs.

The identifiers are, however, fixed à priori eventually by the type declarations.

Table 1.

T	*TIME*	*time-name*
ITS space V	*ITS VOLUME*	*volume-name*
ITS space O	*ITS OUTPUT*	*output-name*
S	*STEP*	*step-name*
C	*COMPARTMENT*	*comp-name*
IN-C	*IN-COMPARTMENT*	*incomp-name*
OUT-C	*OUT-COMPARTMENT*	*outcomp-name*
FO	*FORMAT*	*format-name*
empty	*NO*	*empty-name*
G	*GRAPH*	*graph-name*
P	*PROGRAM*	*program-name*
C	*CONSTANT*	*constant-name*
U	*UNKNOWN*	*implicite-name*
H	*HALFTIME*	*halftime-name*
ITS space I	*ITS INPUT*	*input-name*
ITS space F	*ITS FLOW*	*flow-name*
L	*LABELLED*	*label-name*
FI	*FISSION OF TRACER*	*fission-name*
N	*NUMBER OF COMPARTMENTS*	*number-name*
F	*FROM COMPARTMENT*	*component-name*

The words can be modified according to the temperament of the user of COSMO and according to his abilities for orientation in formulas. One can use long words and comments in order to be near to the natural language. Another person can use abbreviations and identifiers in order to save his time. Thus any name must be initiated by a certain letter or a group of letter à priori fixed; in the table 1, they are presented in the first column. After that initiation a string can follow immediately; for the first group of three words, this string must be composed only of letters, for

the other ones, it can contain letters and spaces. The first group is separated by a more significant line from the other one in the table 1. In the second column of the table, there are the representations of the words which the author considers as the most currents. In the third column there are the metalinguistical variables under which the corresponding name with all of its possible modifications can be presented in the syntactical formulas. The table thus represents a group of other syntactical formulas, which could be very easily made by the reader himself and which are here represented only by two examples:

time-name is T or T tar-identifier-iterated
word-expression is space or tar-identifier or word-expression word-expression
comp-name is C or C word-expression

Thus *time-name* is *T, TIME, TD* etc., *comp-name* is *C, COMP, CM, COMPARTMENT, COMPARTMENT OF IRON IN MARROW* etc.

One can make *comments* also in other places in COSMO texts. They are noted by comment in the syntactical formulas. Thus:

comment is es or word-expression comment or digit comment or operator comment or . comment or : comment

A syntactical mean used almost in all simulation means of COSMO is *declarative*. Its syntax:

declarative is = expression es or : es program

The first case means that the declared value, mentioned before the sign of equality, is determined by the arithmetic expression. In the second case the declared value (noted before the colon) is to be computed by the following program; in it, it is identified by *X* but if the ordinary identifier representing it occurs there it means the corresponding value in the preceding step. In the case that declarative is formed by an expression the same interpretation of the corresponding identifier takes place. It permits recursive determining of values. Examples of the application of declaratives (the sign *es* is represented by a new line signal):

TIME = *T* + 3·0

It means that the time corresponding to the actual step is greater of three units than the time of the last step.

TIME:
READ X
X = *X* * 0·5

It means that the values of the time may be various and their actual values divided by *2* are punched in the input medium.

2.3.2. *Initial conditions*

2.3.2.1. *Syntax:*
initial is I comment program es

2.3.2.2. *Semantics. Initial* is the component of COSMO text determining the initial conditions of modelling. The values are determined algorithmically: a value which is determined according to other values (by assigning etc.) must be presented in the program after assigning for those values. If D is not assigned in the initial, it obtains automatically the value *1.0*, the variables T, Ki, Vi, Gi, HKi, HVi, HGi receive automatically the initial values *0* if they are not assigned.

2.3.3. *Time declaration*

2.3.3.1. *Syntax:*
time-declaration is time-name declarative or step-name declarative or time-name space IS space NORMAL es

2.3.3.2. *Semantics.* The first case defines directly the value of T and the value of D follows automatically as a difference between the preceding value and the actual one. In the second case, it is D which is defined directly and the actual value of T follows from the preceding one by adding D automatically. The third case represents a readable abbreviation for the most usual situation: T is increased in every step by adding D which is not modified by the declaration of course, it can be modified by a side effect – see 2.4.3. The third case can be expressed e.g. as $TIME = T + D$.

2.3.4. *Declaration of in-compartment*

2.3.4.1. *Syntax:*
flow-part is flow-name declarative
label-part is label-name declarative or empty
in-comp-heading is incomp-name integer comment
in-comp-declaration is in-comp-heading flow-part label-part

2.3.4.2. *Semantics. In-comp-declaration* declares the substantial values of an in-compartment. All compartments, including in-compartments and out-compartments, are enumerated by integers, called order numbers. The order number i of the in-compartment is presented in the heading after *incomp-name*. The flow-part declares the measure of substance which enters to the system through the in-compartment in the actual time step. Exactly, there is determined the value of Vi and Gi (they are put as the same numbers in this case). If the substance transported through the in-compartment carries a non-empty measure of tracer, it is declared in label-part (thus HVi and HGi are declared).

2.3.4.3. *Limitation.* The comment in the heading must not be initiated by a digit.

2.3.5. *Declaration of compartment*

2.3.5.1. *Syntax*:

component is component-name integer declarative
input-part is input-name comment component-iterated or input-name empty-name es
output-part is output-name declarative or output-name space IS space constant-name es or output-name space IS space implicite-name es
volume-part is volume-name declarative or volume-name space IS space constant-name es or volume-name space IS space implicite-name es
comp-heading is comp-name integer comment
comp-declaration is comp-heading input-part output-part volume-part

2.3.5.2. *Semantics. Comp-declaration* declares the important values for a compartment. Its order number is presented after *comp-name* in the heading. The measure of the substance which is transported to this compartment from a compartment with the order number j during the actual time step is determined in the component which has j in place of the integer following the component-name. The sum of all those values gives Gi, where G is the order number of the declared compartment. If there is no input into the declared compartment, one need the second case of *input-part*. In the *output-part* the total measure of the substance which leaves the declared compartment in the actual time step is determined. The case *ITS OUTPUT IS CONSTANT* expresses that the measure of leaving substance has not been modified regarding to the preceding time step (it can be, however, modified by the side effects). The case *ITS INPUT IS UNKNOWN* etc. represents the case of implicite definition: if one knows the preceding value of the volume and the actual one and if one knows the value of Gi as well, he can determine the value of the leaving substance. Then he writes simply that implicite definition and the compiler makes the action for determining the value of output automatically. The *output-part* determines therefore the value of Ki. The *volume-part* determines the measure of the substance which is to be in the declared compartment in the actual step, therefore Vi. Its semantics is analoguous to that of *output-part*. The values of HVi, HKi and HGi follow automatically.

The ordering of the components of the input is not determined and the order numbers following the *component-name* need not satisfy any rule. One order number can occur in two or more components in a declaration of the same compartment: it represents two distinct channels between the same couple of compartments.

2.3.5.3. *Limitations.* The explicite definition can occur only once in the same declaration; thus, if it is in the *output-part* it cannot be in the volume-part of the same declaration of compartment. *Comment* cannot begin by a digit.

2.3.6. *Declaration of out-compartment.*

2.3.6.1. *Syntax*:
out-comp-declaration is out-comp-heading component-iterated
out-comp-heading is outcomp-name integer comment

2.3.6.2. *Semantics.* The out-compartment is considered as a compartment the input of which is its only important value. It is formed similarly as an input to an ordinary compartment, but the case of zero number of components has no importance. The components added give the value of *Gi* where *i* is the order number of the declared out-compartment. presented in the heading. That value is the measure of substance which leaves the system through the declared out-compartment in the actual time step. As to the order numbers of the components, see 2.3.5.2, second paragraph (HGi follows automatically).

2.3.6.3. *Limitation.* Comment cannot be initiated by a digit.

2.3.7. *Tracer declaration*

2.3.7.1. *Syntax*:
tracer-declaration is halftime-name = unary or fission-name declarative

2.3.7.2. *Semantics.* The proper change of tracer is declared, see 2.1.1. If it does not exist, the tracer declaration can be omitted. The first case corresponds to the declaration of the halftime, defined in 2.1.1.6. The second case determines the value of the quotient

(2.3.7.2.1.)

$$K = \frac{\textit{measure of the tracer at the beginning of the actual step}}{\textit{measure of the tracer at the end of the actual step}}.$$

The measure can be considered of any place of the modelled system, but of the same place for the both terms of the quotient, because K is bounded with B of 2.1.1.5 with the following relations:

(2.3.7.2.2) $$K = \frac{1}{B \,.\, D + 1} \qquad B = \frac{1 - K}{K \,.\, D}.$$

In COSMO there is no limitation for K or halftime.

2.3.8. *Format declaration*

2.3.8.1. *Syntax*:
format-list is space-iterated variable or format-list space-iterated variable
*number-form is *-iterated or *-sequenced · *-iterated*
form is space-iterated number-form

format-declaration is format-name comment or format-name comment format-list es form-iterated

2.3.8.2. *Format declaration* determines the form of the table mentioned in 2.1.2. Format-list is a line which is completely reproduced as a heading of the table. The *i*-th number-form corresponds to the *i*-th identifier of the format-list: its value is printed in the same place on the line as the number-form and the asteriscs represent the digits. Zeros at the beginning of the number are replaced by spaces, an eventual sign of minus is placed before the first valid digit.

2.3.8.3. *The heading* is printed at the beginning of modelling. If one needs to print anything into the lines preceding the heading he can instruct it by algorithmic means in the initial conditions.

The values are printed at every step, forming thus columns following the heading. Each line corresponds to a time step. If one wishes to put other printed information to the same line he must state it by algorithmical means after the declaration. If one wishes to print anything after the table he must instruct it by algorithmical means in the appendix of the COSMO text – see further. If one would not print the results in the standart form of the table he can use the first case of the format declaration, may be *FORMAT*: *NO*, or he omits this declaration at all. In both cases, he uses for printing algorithmical means.

2.3.8.4. *Limitations.* Two or more format declarations are not permitted in the same modelling.

2.3.9. *Graph declaration*

2.3.9.1. *Syntax*:
point-symbol is digit or operator or · or : or = or, or (or)
point-value is variable, unary, unary, point-symbol es
graph-declaration is graph-name comment point-value-iterated

2.3.9.2. *Semantics.* Graph declaration serves to print results in the form of curves; every graph declaration causes a line in printing during a time step. At the beginning the letter *I* is printed: the sequence of *I* one after another forms the time axis; if the input prints minor letter, the semantics can be modified by replacing *I* by *l* etc. Beside this symbol, point-symbols are printed at the same line according to the point-values: every point-value causes printing of one point-symbol so that a value of the variable presented at the beginning of the point-value is diminished by the first unary, then divided by the second unary, the entier of the result is done – let us call it *N* – the istruction for *N* spaces is performed and the corresponding point-symbol is printed. Then the printing mechanism returns to the beginning of the line. Thus each variable forms a curve composed from the corresponding point-symbols, the first unary means the position of time axis and the second unary means the mini-

mum interval of the values of the printed variable discernible by the printing device. If it is negative it causes the inverting of the axis of values.

If we wish to print other information before, after or into the graphs, we must follow the same rules as in 2.3.8.4. If we use a format declaration and a graph declaration in the same modelling, the lines of the table are printed alternating, similarly as if more graph declarations are present. The order of lines in one step corresponds to the order of the declarations.

2.3.10. *Program blocks*

2.3.10.1. *Syntax*:
program-block is program-name comment program

2.3.10.2. *Semantics.* However, COSMO has algorithmical facilities, namely in programs in declaratives, other ones, which are not syntactically joined with any declarations, are permitted. One can use them for preparing special values, needed in a following declaration, or needed at the end of modelling, for spacial printing etc. X is there an auxiliary variable with no determined value when entering into the program block. The values of the volumes, inputs and outputs of compartments and the corresponding terms representing tracer are interpreted so that if the program block is written before the declaration where the considered variable is determined the value at the beginning of the step is put in the program block, while if the program block is written after the declaration where the considered variable is determined in the program block is used the value at the end of the step.

2.3.11. *Models*

2.3.11.1. *Syntax*:
body-component is time-declaration or in-comp-declaration or comp-declaration or out-comp-declaration or tracer-declaration or format-declaration or graph-declaration or program-block
body-part is body-component es-iterated
body is B comment es-sequenced body-part-sequenced REPEAT space WHILE space unary space IS space GREATER space THAN space unary es
model is es number-name integer es type-declaration-sequenced es initial body program es

2.3.11.2. *Semantics.* Model is a text in COSMO which describes a complete problem which should be solved by means of the system COSMO. At the beginning of the model, the number of compartments is put. Exactly, the integer following the number name must not be smaller than the greatest order number which occurs in the following text. Thus it is permitted to omit an integer in enumerating the compartments, which can be suitable when one omits a compartment from the

modelled hypotheses. The following type declarations introduce the variables which have not been established automatically – see 2.2.1.2. The body determines the substantial properties of the system during the modelling. The declarations need not to be ordered there according to the order numbers. The statement *REPEAT*... determines the duration of the modelling. After having modelled the program following the body is performed. If there is no jump to the beginning – see 2.4.2. – the stop is done automatically after it; when one lets the computer running, the modelling begins from the beginning.

2.3.11.3. *Limitations.* Only one format declaration is permitted. We advice the same idea for the tracer declaration and for the time declaration; after studying the compiling algorithm one recognizes that the doubling of these declarations has certain sense but its complicated character is rather contrary to the sense of automatic programming as an aid for easy expressing of problems for the computer. Initial conditions cannot begin by an instruction having a label before itself – see 2.4.2. The same limitation holds for a program block if it is the first component of the body.

2.4. Complements

2.4.1. *Division by zero*

The semantics of the operation of division in the target language is usual if the operation has been risen by copying from a program or an expression of COSMO texts. The cases of that operation which are generated by the compiler, which have thus no corresponding operation in the COSMO text, have their semantics generalized so that if the value of the divisor is zero, the result is zero. That affaire can be realized either by an interpretative system which gives the generalized semantics to all the cases of the target program or by enlarging the target language by other operations, e.g. those of the machine code: the division is thus a set of operations. The described affaire simplifies substantially the compiling algorithm.

2.4.2. *Jumps*

The labels are not localized in the target program, as there are not blocks in the sense of e.g. ALGOL 60. The same rule is valid in the COSMO texts, because it simplifies the compiler. However, the jumps from one program to another have no interpretation in the compartmental systems, with the following exceptions:

2.4.2.1. The jump from a body component to the appendix interrupts the modelling and the remaining part of the appendix is performed.

2.4.2.2. The jump from a body component to the label *4* interrupts the running modelling and a new modelling is started from the beginning. The same rule holds

for the same jump from the appendix: the finished modelling is immediately followed by a new run of the same modelling. The same jump which is presented in the initial conditions causes a completely new starting of the modelling.

2.4.2.3. The jump to the label *1* is equivalent to the jump to the beginning of the step. It has a non-sophistical sense only if it is present in the appendix: it can prolongate the modelling, however, it has been interrupted by the statement *REPEAT*...

2.4.2.4. One can omit printing a line of the table or of the graph so that he puts before the corresponding format declaration or graph declaration a program block with a branching: Omitting is described by a jump to a label (greater than *4*); this label is present in front of an instruction forming a part of a program block following the considered declaration.

2.4.2.5. Jumps to the labels *2* and *3* have no readable interpretation in the compartmental systems.

2.4.3. *Side effects*

Side effects are permitted, similarly as in ALGOL etc. For example a program in a declarative can assign a new value for *D* and thus the time declaration can be modified in a distinct place of the COSMO text.

2.4.5. Examples

The author means that there is no correlation between cases which are suitable for illustration of the syntactical facilities of COSMO and those which are suitable for illustration of modelling and optimizing methods. Therefore he has chosen simple examples which are clear regarding to the application but which seems to illustrate in a good sense the use of COSMO. Another example is presented in [1].

2.4.5.1. Two compartments *A* and *B* have constant volumes, they are joined by channels, the first one is joined with an in-compartment through which a labelled substance enters into the system the measure of which is punched in every step in the input medium of the computer for 100 steps. The compartment *B* is joined with an out-compartment. All flow rates are constant. Our interest concerns the measure of labelled substance contained in B and leaving through the out-compartment. The spaces are interpreted here in the usual graphical form and the *es* signs are replaced as new lines.

```
NUMBER OF COMPARTMENTS 4

INITIAL CONDITIONS
V2 = 300·0
V3 = 500·0

BODY
IN-COMPARTMENT 1
ITS FLOW = 20·0
LABELLED:
READ X
```

```
COMPARTMENT 2(A)
ITS INPUT
FROM COMP 1 = 20·0
FROM COMP 3 = 30·0
ITS OUTPUT IS UNKNOWN
ITS VOLUME IS CONSTANT

COMPARTMENT 3(B)
ITS INPUT
FROM C2 = 50·0
ITS OUTPUT IS UNKNOWN
ITS VOLUME IS CONSTANT

OUT-COMPARTMENT 4
FROM C3 = 20·0

FORMAT
HV 3      HG 4
***·*     **·*

TIME IS NORMAL
REPEAT WHILE 101 · IS GREATER THAN T
```

2.4.5.2. The system modelled is the same as in 2.4.5.1. but we do another experiment with it: in the compartment *A* there is a certain measure of the labelled substance at the beginning. We have a paper tape with numbers which have been risen as results of an experiment with the corresponding system of the real universe. There numbers correspond to *HV 3*. We need to receive a graph where the experimental results are compared with the hypothetical ones. The points corresponding to the experimental values are represented by + and the points corresponding to the hypothetical values are represented by /.

```
NUMBER OF COMPARTMENTS 4
REAL E

INITIAL
V 2 = 300·0
V 3 = 500·0
HV 2 = 10 000·0

BODY
IN-COMPARTMENT 1
ITS FLOW = 20·0

COMPARTMENT 2(A)
ITS INPUT
FROM C1 = 20·0
FROM C3 = 30·0
ITS OUTPUT IS UNIMPORTANT
ITS VOLUME IS CONSTANT

COMPARTMENT 3(B)
ITS INPUT
FROM C2 = 50·0
ITS OUTPUT IS UNKNOWN
```

ITS VOLUME IS CONSTANT

OUT-COMPARTMENT 4
FROM C3 = 20·0

TIME IS NORMAL

PROGRAM
READ E

GRAPH
E, 0·0, 100·0, /
HV3, 0·0, 100·0, +

REPEAT WHILE 101 · IS GREATER THAN T

3. COMPILER

3.1 The structure

The compiler reads the text in COSMO and produces the corresponding programs in the target language, see 2.1.3. The compiler can be considered as a facility F before which a queue takes place. The queue is formed by the symbols of the compiled text. The ordering in the queue is the same as in reading: thus the symbols which would be read before the others are seized by F also before the others. The facility F is controlled always by one symbol: we call it *present symbol*. In some phases of compilation the present symbol is released and the next one is seized.

The facility F behaves as it would have two small stacks R a S. R has the capacity of three words, S has the capacity of four words. Beside the stacs F has two addresses c and d and an array B 1, B 2, ... whose length is equal to the most possible number of the symbols in a line (it depends on used printing devices). Due to its run, the facility F is considered as divided to segments. To each segment its symbols is established which is called its name and which is used as a signal controlling the work of F. Recursively: if a segment ends its work the following segment goes on, the name of which is the last one in the stack S.

Each segment is described as a finite sequence of *actions*. We distinguish three groups of the actions:

3.1.1. *Controlling actions*

They change the contents of the stack S. We consider three different actions (X is a name of a segment):

Enter(X); the new symbol X is put into S the contents of which thus rises of 1 word.

Leave; the last symbol is deleted out from S and thus its contents is diminished of 1 word.

Change(X); the last name of S is replaced by X; this action can be considered as an abbreviation for *Leave*, *Enter*(X).

We advice to implement these actions as subroutines or procedures.

3.1.2. *Organizing actions*

They do various works during the segments run as copying or omitting parts of the COSMO source texts, assigning of important values for next run of the compiler etc. They are detailly described in 3.3.

3.1.3. *Generating actions*

They generate various parts of the target program (see 3.4).

At the beginning of the compilation the facility F is put into the segment *Start* whose name is given into the stack S. The present symbol is the first symbol of the source text, therefore *es*. The following processing is given by the recursive rules described below. The stack R is empty, the contents of c is *1* and the contents of d and of the array B is not important.

3.2 Segments

The names of the segments have been chosen so that they might illustrate the corresponding metalinguistical variables processing in each of them. The work of each segment is usually branched according to the present symbol. Thus each segment is described so that after its name (and an eventual short explication) a sequence of paragraphs follows, each of them corresponding to a symbol of COSMO. This symbol is noted at the left hand side, then a hyphen takes place and then the description follows, which tells what is to be done if the corresponding symbol is present. The description is usually a sequence of actions. After having performed the sequence of actions, F gets to the segment according to the last name in S and in that segment it is immediatelly branched according to the present symbol etc.

The symbols which are not mentioned at the left hand side of a segment description must not occur as present ones when the segment is branched. Thus their occuring in this situation signalizes a syntactical error.

Similarly as in [7] or [6], the facility F can be described by a translation matrix, the columns of which would correspond to the segments and the lines to the symbols. An element of the matrix corresponds to a paragraph of a segment description, or it signalizes a syntactical error. The reader can construct the translation matrix himself according to the following descriptions of the segments.

Note. The paragraphs of the segment descriptions degenerates often to lines.

Start

The segment used in starting the compilation.
es — P 3
N — P 5, Change (Type), P 3

Type

The segment used in eventual processing of type declarations.
es — Change (Initial), G 34, G 25
R — Change (Initial), G 34, P 4, G 25
I — Change (Initial), G 34, P 4, G 25

Initial

I — P 2 (es), P 3, P 4, G 22, Change (General)
es — P 3

General

The segment used during the whole compilation of the body and of the appendix (the program taking place after the body). This segment performs the branching to the other segments which process the components of the body so that their names are caused to enter in the stack *S*.

T — P 2(pace, :, =), Enter(Time), Enter(Copy)
F — P 3, Enter(Fission)
es — P 3
S — P 2(:, =), Enter(Step), Enter(Copy)
I — P 5, P 2(es), P 3, P 2(=, :), Enter(In), Enter(Copy)
B — G 22, P 2(es)
O — P 5, P 2(es), P 3, G 10, Enter(Out)
H — P 2(es), G 6, P 1, G 7, G 5
G — P 2, G 26, Enter(Graph)
C — P 5, P 2(es), P 3, G 10, P 2(es), Enter(Out)
P — P 2(es), P 3, P 4
R — P 2(space), P 2(space), G 21, P 11, P 2(space), P 2(space), P 2(space), P 1, P 4, G 33, P 14, G 34, P 15

Time

The segment used for processing the time declaration if it is initiated by time-name.
space — G 3, P 2(es), P 3, Leave
es — G 2, Leave

Step

The segment for processing the time declaration if it is initiated by the step-name.
es — G 4, Leave

Copy

The segment used for processing of declaratives.
= — G1, P 1, Leave
: — P 2(es), P 1(es, L, F, I), Leave
space — Leave
es — Leave

In

The segment used in the processing of the in-compartment declarations and of the parts of the compartment declarations concerning outputs into the compartments.

space — P 8
es — G 8, G 13, G 14, P 6, Leave
L — G 8, P 2(=, :), Change(Label), Enter(Copy)
I — G 8, G 15, P 2(space), P 3, P 2(=, :, space), Change(Volume), Enter(Copy)
U — P 10, G 19, P 2(es), P 3, P 2(space), P 3, P 2(=, :, space), Change(Volume), Enter(Copy)
C — G 15, P 2(es), P 3, P 2(space), P 3, P 2(=, :, space), Change(Volume), Enter(Copy)

Out

The segment used in the processing of the out-compartment declarations and of the parts of the compartment declarations concerning the inputs into the compartments.

es — G 12, P 6, Leave
F — P 5, Enter(From), Enter(Copy)
I — G 12, P 7, P 2(space), P 3, P 2(=, :, space), Change(In), Enter(Copy)

From

The segment used in the processing the components of the inputs into the compartments and of the out-compartments.

es — G 11, P 6, Leave
F — G 11, P 6, P 5, Enter(Copy)
I — G 11, P 6, Leave

Label

The segment used for the processing of the label-part of the in-compartment declarations.

es — G 9, G 14, P6, Leave

Volume

The segment used for the processing of the volume-part of the declarations of the compartments.

space — P 8
es — G1 6, P 9, G 17, P 2(es), P 6, Leave
U — G 20, P 2(es), P 6, Leave
C — P 9, G 17, P 2(es), P 6, Leave

Fission

The segment used for the processing of the declaration of tracer in the case that it is expressed by means of fission-name; for the declaration of the halftime there is no necessary segment. The segment is changed by the segment *Format* rather immediately if it is needed.

O — Change(Format), P 2(es), P 3
I — P 2(:, =), Enter(Copy)
es — G5, Leave

Format

es — G 23, G 24, Leave
space — G 23, G 31, P 13, G 24, Leave
tar-identifier — G 23, G 31, P 13, G 24, Leave

Graph

es — Leave
tar-identifier — G 1, P 16, G 27, P 16, G 28, P 16, G 29, P 4(es), P 3, G 30

P 1: if it has no parameters: release the present symbol and copy the following symbols from the source text into the target program until *es*; this symbol is copied as well and the next symbol is then present. The identifiers *HV*, *HG* and *HK* are replaced in the copied text by *O*, *H*, *I* respectively. If parameters follow: analogical copying is performed but until the symbol of es which is followed by some of the parameters. The parameter is the present symbol after leaving the action, the preceding *es* is the last copied symbol.

P 2: the present symbol is released and the following symbols as well until one of the parameters is present. It remains the present symbol when this action is left. In the implementation for the computer ODRA in Biophysical Institute of Charles university in Prague the action *P* 2 is organised so that in the case that es is not a parameter the action is branched: if the present symbol during its operating is *es*, compilation is stopped and the syntactical error signal is given: thus a lot of syntactical errors is detected.

Note. In the present paper the parameters are separated by a comma; it cannot lead to a chaotic expressing because a comma cannot be never a parameter.

P 3: release the present symbol and seize the next one.

P 4: empty in the case that the present symbol is *es*, otherwise the text is copied similarly as in *P* 1; the first copied symbol is the present one, the last symbol copied is the first *es* followed by another *es*. The second *es* is not copied but it is the present symbol after having left *P* 4.

P 5: release the present symbol and seize the next one; do this operation until a digit is the present symbol. That digit and eventually following ones form an integer which is pushed down into the stack *R*. The present symbol is the symbol following immediately the read integer.

P 6: the contents of the stack *R* is contracted of one word.

P 7: zero is assigned to *d*.

P 8: release the present symbol and the next three symbols as well.

P 9: if $d = 1$ then do the generating action *G* 18.

P 10: 1 is assigned to *d*.

P 11: copy (similarly as in *P* 1) the source text to the target text; the first copied symbol is the present one, the last copied symbol is that followed by *space*. The symbol *space* is the present one after leaving the action.

P 12: zero is assigned to *c*.

P 13: beginning from the present symbol all symbol are copied to *B* 1, *B* 2, *B* 3, ... until *es*, which is not copied more but is present. Then the following symbols are read until a new *es*, while the corresponding instructions for input are compiled according to the read symbols and *Bi*. We tell nothing more of this affaire as the printing instructions differs for different types of the computers and as their compiling puts only technical obstacles.

P 14: If $c = 1$ then do *G* 32.
P 15: Stop of the compilation.

3.4 Generating actions

They generate parts of the target program. Each of them can be exactly described according to the table 2: an action *Gi* operates so that the present symbol is not changed, and the string presented at the right hand side of the table is copied into

Table 2.

G 1	*X=*
G 2	*D=X—T es T=X es*
G 3	*D=1 es T=T+1 es*
G 4	*D=X es T=T+D es*
G 5	*FOR space B=1 space STEP space 1 space UNTIL space r es OB=OB|X es*
G 6	*X=0—0·693147 es X=X * D es X=X|*
G 7	*X=EXP space X es*
G 8	*Kr=X es*
G 9	*Ir=X es*
G 10	*C=0 es L=0 es*
G 11	*F=X * Or es F=F|Vr es C=C+X es L=L+F es*
G 12	*Gr=C es Hr=L es*
G 13	*Ir=0 es*
G 14	*Vr=Kr es Or=Ir es*
G 15	*F=Or * Kr es Ir=F|Vr es*
G 16	*Vr=X es*
G 17	*X=Or—Ir es Or=X+Hr es*
G 18	*L=L+Gr es Kr=L—Vr es F=Or * Kr es Ir=F|Vr es X=Or—Ir es Or=X+Hr es*
G 19	*L=Vr es*
G 20	*X=Vr—Kr es Vr=X+Gr es*
G 21	*GO space TO space 0 space 0 space 1 space IF space*
G 22	*GO space TO space 2 es 1:*
G 23	*GO space TO space 3 es 2:*
G 24	*GO space TO space 1 es 3:*
G 25	*REAL space VrGrKrHrOrIrTDXFLC es INTEGER B es 4: FOR space B=1 space STEP space 1 space UNTIL space r es KB=0·0 es GB=0·0 es VB=0·0 es OB=0·0 es IB=0·0 es HB=0·0 es END es T=0·0 es D=1·0 es*
G 26	*PRINTLINE space 1 es PRINTOUT space 1 es*
G 27	—
G 28	*es X=X|*
G 29	*es B=ENTIER space X es PRINTSPACE space B es PRINTOUT space*
G 30	*es PRINTOUT space carriage-return-symbol es*
G 31	*PRINTLINE space 1 es*
G 32	*2:GO space TO space 1 es*
G 33	*STOP space 4 es*
G 34	*es*

the target text, while at the left hand side there is the name of the corresponding action. The presented texts are written in the symbols of the target language excepting the letter r which means that in its place the contents of the last word of the stack R is to be copied, without contraction of R.

(Received November 11th, 1968.)

REFERENCES

[1] Kindler, E.: COSMO (Compartmental System Modelling), Description of a programming system. Proceedings of the IFIP Working Conference on Simulation Programming Languages, Oslo 1967, North-Holland Publishing Company, Amsterdam 1968.
[2] Sheppard, C. W.: Basic Principles of Tracer Method, John Wiley et Sons, New York 1962.
[3] Rescigno, A., Segre, G.: La Cinetica dei Pharmaci e dei Traccianti Radioattivi. Edit. Univ., Bordingheri, Torino 1961.
[4] Kindler, E.: COSMO (Compartmental System Modelling), Description of a programming system. Charles University, Prague 1967, for internal use of the University.
[5] Naur, P. (editor): Report on the Algorithmic Language ALGOL 60. Comm. ACM *3* (1960), 5, 299—314.
[6] Grau, A. A.: The Structure of an ALGOL-translator. ORNL-3054. Oak Ridge National Laboratory, 1961, No 23.
[7] Kindler, E.: Translation of Arithmetic expressions by EPOS ALGOL Compiler. Information Processing Machines, No. 9, Prague 1963, p. 69—114.
[8] Kindler, E.: Automatic Modelling of Compartmental Systems. IFIP Congres 68, Edinburgh, August 1968, pp. H 75—H 79.

Simulační systém COSMO
POPIS JAZYKA A KOMPILÁTORU

EVŽEN KINDLER

Simulační systém COSMO automatického programování umožňuje snadné programování modelů složkových (kompartmentových) systémů a jejich vnořování do obsáhlejších programů, psaných v obvyklé algoritmické formě. Složkové systémy se zatím používají hlavně v modelování živých organismů; zobrazují obecné vlastnosti dopravy a míšení kapalin. Článek obsahuje formální popis syntaxe jazyka COSMO, popis jeho sémantiky a jeho kompilátoru, který z popisů modelů sestavuje programy v algoritmické formě.

Texty v jazyku COSMO může sestavovat i odborník, který se nezajímá přímo o programování. Jsou sestaveny z odstavců, které odpovídají jednotlivým složkám modelovaného systému, nebo se v nich popisuje, co se má při modelování tisknout, počáteční hodnoty atd. Tyto popisy vycházejí z terminologie obvyklé v teorii složkových systémů.

RNDr. Evžen Kindler, Biofyzikální ústav FVL UK, Salmovská 3, Praha 2.