Computer Science Journal of Moldova, vol.**6**, no.3(18), 1998

The concept of training device and task description language for teaching programming

Dat Ph. Phan

Abstract

In this paper we consider conceptual bases of construction of training device on programming (TDP) and give theoretical grounds of task description language (TADELA). TDP is a specialized training system of algorithmic approach which allows the student work with real tasks under conditions of his/her superficial acquaintance with the programming language. TADELA is designed to describe solving plans of class of educational computing tasks. On the basis of plans the student can get flow charts of algorithms and fragments of programs in the studied programming language. TADELA is designed on the basis of algorithmic operator models. Being close to algebraic notation, TADELA provides simplicity of its use as the entry language of TDP. Approaches to formalization of syntax, model and algorithmic semantics of TADELA and its implementation were considered.

1 Introduction

From the point of view of teaching methods of programming one can pick out three approaches to the construction of computer-aided learning (CAL) systems: linguistic, algorithmic and combined [9]. In CAL systems of linguistic approach [13] the instruction begins with study of programming tools and is accompanied by artificially selected examples, illustrating those or other operational functions of the language

[6]. The learning process goes from the statement of ready knowledge to excercises. At this process the student learns some concrete material, not going out of the produced framework. Thus, the creative conditions, necessary for promoting independent obtaining of knowledge are not fully created.

In CAL systems of algorithmic approach the language and programming methods are learned by solving various problems. At first the student trains the skills of drawing up solving algorithms of problems, the language is studied only as far as it is necessary to program these algorithms. The algorithmic approach has some variations in implementation, caused by distribution of functions between the man and the machine during the process of problem solving. Depending on this, they can be divided into 3 types: DIPRO (conversational programming) [5], ASSISTANTS [2] and COACHES [1,3].

The combined approach contains both considered methods. Information-training environments [9] are of this approach. They contain subsystems of linguistic and algorithmic approaches, used at different stages of training.

Training devices on programming (TDP) [10] relate to CAL system of algorithmic approach. TDP are the least investigated area, though in all above-considered systems we can find out elements of drill in one or other form. Those elements are used to develop and strengthen practical skills of programming. There is a number of computer programs for developing motor skills, e.g. for working with keyboards. But in this case we are interested in training devices on developing skills of intelligent activity in the area of problem algorithmization and programming. Among the above-considered systems for teaching programming the systems of the types DIPRO and COACHES are most close to training devices on intelligent activity. At the same time they do not relate to TDP as some of them requires from the trainee a detailed description of the solving algorithm for the problem and does not control the process of its creation. The other can do it, but only with problems of a limited set included in its knowledge base, imposing a determined solving technique. Thus, we pick out training devices on teaching problem algorithmization and programming as an independent

type of CAL systems, providing trainees with following opportunities:

- work with *arbitrary* problems of a certain domain on the basis of description of their solving plans;
- receive a graphic representation of flow chart of the algorithm on the basis of the given solving plan;
- investigate execution of the algorithm in dynamics on various sets of initial data to evaluate the correctness of the solution;
- receive text of program in the studied programming language;
- reveal correspondence between the solving plan, the algorithm and the program to analyse the algorithmic techniques used in the solution and ways of their implementation in the studied programming language.

These opportunities provide the trainee with learning on own experience, in an experimentation mode, on real problems. The most effective use of TDP is in the complex approach to preparing base-level programmers, when conventional and computer-aided methods of teaching are combined at various stages. In this paper, our objective is to consider the conceptual bases of construction of TDP, to give the theoretical grounds of the description language for solving plans of tasks and to describe the mechanism of its implementation.

2 Conceptual bases of TDP

The concept of TDP construction offered in this paper is considered on the basis of problem concept accepted in systemology and artificial intelligence [4,8]. A problem is submitted by a set of four elements $Z = (S_0, S, S_k, Y)$, where S is an abstract set of states of the domain, $S_0 \subset S$: set of initial states, $S_k \subset S$: set of final states and Y is set of operators (mappings), transforming states.

We shall give an interpretation of Z applied to the class of programming tasks (ZP). Assume a ZP is formulated as follows: given a vector

M of n elements, place the elements of the vector in the increasing order of their values. In this case, the set of states of domain S includes all intermediate vectors generated by using operators Y on each step of transformation from S_0 into S_k. S_0 includes set of initial vectors of n elements, and S_k includes set of corresponding results

$$M = \{m_k | m_k < m_{k+1}; \quad k = 1, 2, \ldots, n-1\}.$$

The transformation from the initial state S_0 into the final state S_k is named a process of problem solving. For ZP solution, the solving system [4] should have a set of operators $Y = Q \cup D$, where $Q = \{q_i\}$ is the set of planning operators, providing formation of solving plan of the problem, and $D = \{d_j\}$ is the set of executive operators, providing formation of solving program. Accordingly, the plan is presented by a pair $P = (\{q_i\}, R_q)$, where R_q is a sequence relation setting execution order of $\{q_i\}$, and the program is presented as $G = (\{d_j\}, R_d)$, where R_d is a sequence relation on $\{d_j\}$. An intuitive concept of plan and program as a solving method of mathematical problems comes out in widely known works of the outstanding mathematician and teacher G. Polya [7]. We use these concepts for ZP here.

The formalized presentations of plan P and program G are identical down to operations and an essential difference between them is contained just in it. For above-formulated ZP, we shall consider the difference between an element of the plan and the corresponding fragment of the program:

Element of plan P			Fragment of program G		
q_i	:	exchange	d_j	:	$t := m[k];$
		the values	d_{j+1}	:	$m[k] := m[k+1];$
of		of m_k and m_{k+1}	d_{j+2}	:	$m[k+1] := t;$

Obviously, the planning operators $\{q_i\}$ are not formalized and are submitted at high level, while the executive operators $\{d_j\}$ are strictly formalized and are low-level in respect of the plan, for a single operation q_i in the plan P corresponds to a group of executive operations d_j in the program G.

The complications of the initial stage of programming learning are related mainly to the lack of student's algorithmic thinking. Problems arise even when the obvious solution of a problem is to be presented in the form of algorithm. Therefore, it is necessary to find a method effectively showing how the solution of a problem is remapping to algorithm and how the algorithm is transformed into program, e.g. in Pascal. This goal defines two ideas lying in the conceptual basis of TDP construction: the idea of re-coding of algorithmic techniques used in ZP solving by forming algorithmic operator models (operators q_i) of the plan of large information density, and the idea of decoding arising forms, transforming them into corresponding system of elementary algorithmic actions (operators d_i) and generating a program of ZP solving.

3 Algorithmic operator models

Any educational ZP can be divided into subtasks, requiring certain algorithmic techniques to be solved. Each of them is presented in compact formalized form of algorithmic operator model. Due to various interpretations of model concept and the lack of completely satisfying definition, we shall specify the properties which we assign to algorithmic operator model (henceforth model):

- a model is a method reflecting similarity of plan P and program G in the form of isomorphic correspondence between operators of the plan $\{q_i\}$ and operators of the program $\{d_j\}$ and relations R_q and R_d;
- a model allows to transform states of ZP without direct study and use of operators $\{d_j\}$;
- the structure and rules of model functioning is easier to study than structure and rules of program functioning.

Let us consider the stage of mastering principles of problem algorithmization and learning grounds of programming. The analysis of

operational necessities [6] appearing in this stage allows to pick out a set of basic algorithmic techniques used in solving of computing ZP: computation of sums, products, functions, iterative computations, definition of maximum and minimum, etc. These algorithmic techniques are represented in solving plans in the form of corresponding models: sum, product, function, iteration, maximum, minimum, etc. We consider an example of how an algorithmic technique is "folded" to model: assume that in a ZP it is necessary to find the maximum element of a vector among those elements lying in interval $7.5 < b[i] < 25.5$; $i = 1, \ldots, 10$. The model maximum in this case has the form:

$$C = \max(b[i] : (b[i] > 7.5) \quad \text{and} \quad (b[i] < 27.5)) \quad i \text{ from 1 to 10};$$

The model contains all necessary information for its automatic decoding by "rolling" into a program. The program is not trivial. Here an algorithmic idiom of defining the maximum element of a vector is transformed: first, variable C should be originally initialized not by the value of the first element of the vector but by the value of the first one in the sequence which belongs to the given interval. Second, for all subsequent elements before using this algorithmic idiom, it is necessary to check up whether an element belongs to the given interval.

The fragment of the program isomorphic with the indicated model has the following form:

```
Search an element belonging to interval
i := 1;
while not ((b[i] > 7.5) and (b[i] < 27.5) or (i = 10)) do i := i+1;
            if (b[i] > 7.5) and (b[i] < 27.5)
then element is found
      begin C := b[i]; Initialization
                repeat
                    if ((b[i] > 7.5) and (b[i] < 27.5))
                    then element belongs to interval
                     if b[i] > C
                     then C := b[i]; current max. value
```

```
                    i := i+1;
                  until i > 10;
        end
else element not found
        writeln ('There is no element (b[i] > 7.5)and(b[i] < 27.5 )');
```

The comparison of the model of this algorithmic technique and the program implementing this technique makes obvious the capacious information content of models and illustrates their above-stated properties. On the basis of models the description language for solving plans of ZP (shortly TAsk DEscription LAnguage — TADELA) is designed. Being close to algebraic notation, TADELA provides simplicity of its use as the entry language of TDP.

4 Description of solving plans

We shall consider the structure and an example of the description of plan P. Let us consider an educational ZP: given an integer square matrix 10*10, compute the norm of the matrix. We can pick out in the solving plan 2 subtasks described by models: (1) computing vector, the i-th element of which is the sum of modules of elements of the i-th line of matrix (algorithm of summation) and (2) finding maximum element of this vector (algorithm of finding maximum), which is the norm of the matrix. We submit a generalized structure of plans and the solving plan of this ZP in TADELA:

Task _taskname_;	**Task** norma_of_matrix;
Defines	**Defines**
{substitutions}	domen1 = j from 1 to 10;
...	domen2 = i from 1 to 10;
Objects	**Objects**
{ definition	a: array [1..10,1..10] of integer;
of objects}	s, norma, i, j: integer;
...	
Models	**Models**
{description of models}	s = sum(abs(a[i, j]))domen1;
...	norma = max(s)domen2;
Actions	**Actions**
{actions}	input(a[i,j])domen1,domen2;
...	do(norma);
	output(norma);
End	**End**

The description of solving plan of a problem in TADELA consists of heading and 4 sections. Section **Defines** contains substitutions and is used to shorten the text of the plan. In section **Objects** all objects of the task are declared. The feature of this section is that the objects are given in accordance with the syntax of the studied programming language. Within this paper we shall consider that it is Pascal. Section **Models** contains algorithmic techniques necessary for problem solving. Actions of data input-output and launching of models are specified in section **Actions.**

The implementation of TADELA requires a strict description of syntax and semantics of models. Problems of syntax description of formal languages are reflected in the known publications. It is more complex with semantics description. The known ways of semantics description [11] are oriented to compilation of high-level languages to machine code. In the case of TDP there is a basic difference: by description of the solving plan the trainee should get (1) flow chart of the algorithm, (2) opportunity to research dynamic properties of the algorithm and (3) commented text of the program in Pascal implementing the plan.

Thus, we put the problem to develop a mechanism for formalization of TADELA, on the basis of which we can generate flow charts of algorithms, construct an algorithm interpreter and receive texts of programs.

5 Formalization of models

For each model of TADELA we shall describe syntax **Syn**(model), model semantics **Sem**(model) and algorithmic semantics **Alg**(model):

- **Syn**(model) defines the set of correctly constructed models. **Syn**(model) is set by general forms of notation, complemented with Wirth syntactic diagrams [12] exposed up to metalinguistic variables of Pascal. In our case metalinguistic variables of Pascal are considered as terminal symbols of TADELA.

- **Sem**(model) defines the correctness of use of task objects of the model in accordance with its functional essence. **Sem**(model) is set by logic equations, derived for correctly in syntax constructed models. The infringement of the verity of these equations allows to diagnose errors connected with incorrect use of task objects.

- **Alg**(model) defines algorithmic actions of model in processing task objects with the purpose to implement its functional essence. **Alg**(model) is set by operator description, derived from correctly constructed **Syn**(model) and **Sem**(model).

Due to the limitation of this paper we shall consider the basic approaches to formalization of algorithmic operator models of TADELA on model *sum*, corresponding to operator $\sum$ in algebraic notation.

5.1 Syntax

We shall define **Syn**(sum):

$R = \mathbf{sum}(A)D$, where

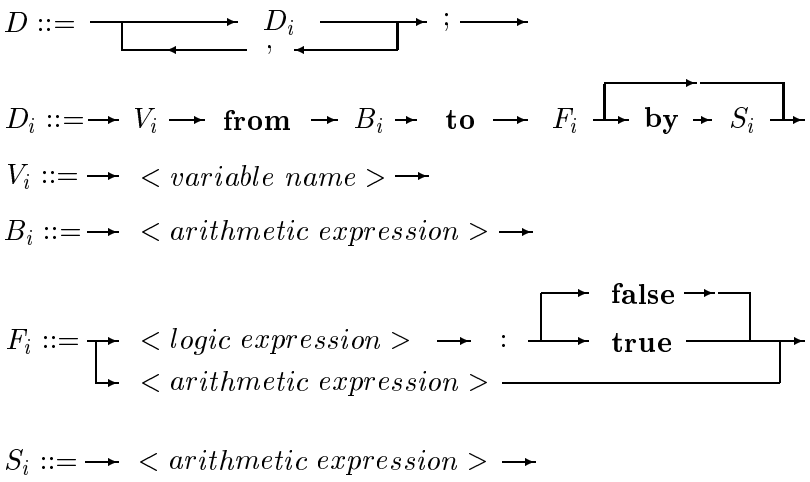
R is $<$ *simple variable* $>$ or $<$ *indexed variable* $>$.
R defines the returned value;

A is the argument of model. A is set as E or $E : C$. Thus, the model *sum* can be given as:

$R = \mathbf{sum}(E)D$ or $R = \mathbf{sum}(E : C)D$, where

E is $<$ *arithmetic expression* $>$; E defines the addend of the sum C is $<$ *logic expression* $>$; C defines condition of inclusion E to R.

When C is absent, E is certainly included in R; D is definition area of R. D is given by list $D_1, D_2, \ldots, D_k$, in which the i-th element of the list D_i defines vector of values of the i-th variable of sum;



In syntactic diagrams for D we have:
V_i – the i-th variable of sum; B_i – initial value of the i-th variable; F_i – final value or final condition of the i-th variable; S_i – step of change of the i-th variable of sum. If the statement 'by S_i' is absent the step $= 1$.

Thus, **Syn**(sum) is defined up to metalinguistic variables of Pascal, which are considered as terminals of TADELA and are not further

exposed as they correspond to the syntax of Pascal. In the description of **Syn**(sum) the key words and separators of TADELA are in bold type, and the terminals are in italics.

We shall show the use of model *sum*: in matrix algebra multiplication of a matrix on a vector is expressed by operator $\sum$:

$$c_i = \sum_{j=1}^{5} a_{ij} * b_j \quad i = 1, \ldots, 5.$$

If a problem has a subtask of multiplication of a matrix on a vector, according to defined **Syn**(sum) the subtask is described in section **Models** as:

$$c[i] = \sum(a[i, j] * b[j]) \qquad j \text{ from 1 to 5;}$$

Obviously, the syntax of models is reasonably close to algebraic notation, which is one of the requirements of TADELA.

5.2 Model semantics

We shall define **Sem**(sum) step by step:

1. Let **M** be set of all task objects. Each object $m_i \in \mathbf{M}$ is characterized by a unique identifier, type and dimension. For the class of computing tasks the type is chosen from **T**={int, real, bool}, and the dimension is defined as $n \in N^0$. If $n = 0$, the object is a scalar, if $n \geq 1$ the object is a n-dimensional array. Thus, **M** is a set of tuples

$$M = \{m_i \mid m_i ::= \quad \textbf{cort} \quad < ident, type, dim >\},$$

where the designation **cort**$< \ldots >$ is introduced to distinguish a tuple from a metalinguistic variable. Actually, **M** is a relation with the domains *ident*, *type* and *dim*. The set **M** is generated by processing section **Objects**. The set **M** is necessary to define of semantics of all models.

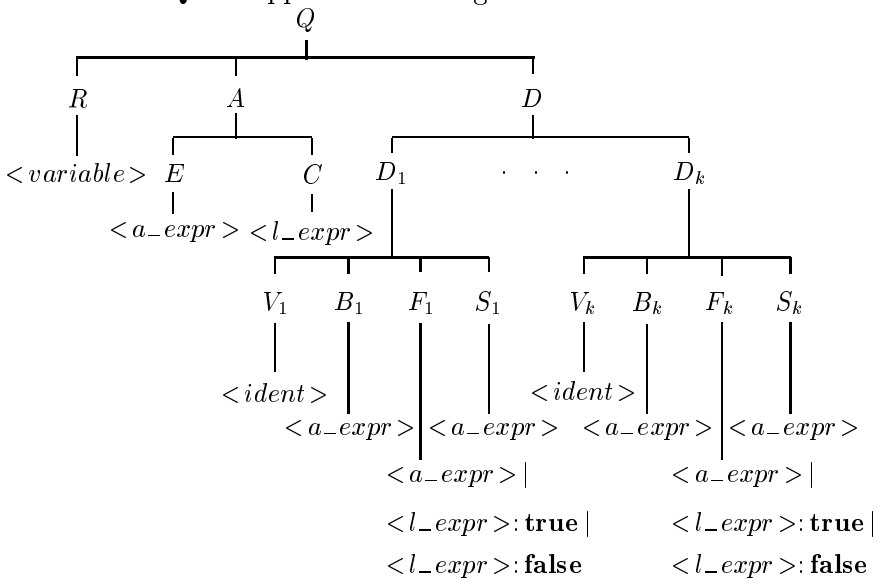
2. We shall introduce concept of structural set: If **Z**, **X** and **Y** are finite sets, then **Z**(**X**,**Y**) is the structural set **Z**, calculated as $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$ through the direct descendants **X** and **Y**. Structural

set transforms context-free **Syn**(model) to a hierarchic tree, on which the context-sensitive limitations given in accordance with functional essence of the model are computed.

Let $Q = \{q_i \mid q_i ::= \mathbf{cort} \ < ident,\ dim >\}$ be set of task objects of a model *sum*. Using symbolics accepted for **Syn**(sum) we define **Q** in the form of structural set:

$$Q = (R, A(E, C), D\,(D_1\,(V_1, B_1, F_1, S_1), \ldots, D_k\,(V_k, B_k, F_k, S_k)))\,,$$

where each set of the lowest level of hierarchic tree has only one direct descendant, which is a terminal (here and henceforth if a non-terminal of model is in bold type, it is considered as a structural set). Thus, the structural set **Q** is mapped to following hierarchical tree:



The context-sensitive limitations are calculated on the sets at the lowest level of tree **Q**, considered as semantic terminals of the model. The elements of these sets are generated by processing direct descendants (terminals of **Syn**(sum)) picking out task objects of them in the form **cort**<ident, dim>. We also note that because tree Q is given on correctly in syntax constructed model, it does not contain terminals of

Syn(sum) which are inessential for computing context-sensitive limitations: **from**, **to**, **by**, etc. Besides, the sets V_i consist of only one tuple, which follows from **Syn**(sum). Thus, **Q** not only defines set of tuples of task objects of model, but also divides them in accordance with their belonging to one or another semantic terminal of model.

3. We shall introduce functions for computing context-sensitive limitations of **Sem**(sum):

(a) **num**(**P**): calculates cardinality of set **P**:

$$\mathbf{num}(\mathbf{P}) = \begin{cases} \text{number of descendants of } \mathbf{P}, & \text{if } \mathbf{P} \text{ is a structural set,} \\ \text{number of elements of } \mathbf{P}, & \text{if } \mathbf{P} \text{ is a usual set.} \end{cases}$$

(b) **zond**(**P**,**X**): defines the entry of set **X** in the set **P**

$$\mathbf{zond}(\mathbf{P}, \mathbf{X}) = \begin{cases} \mathbf{true}, & \text{if } \mathbf{X} \subseteq \mathbf{P}, \\ \mathbf{false}, & \text{else.} \end{cases}$$

(c) **type**(**P**): calculates type of terminal which is the descendant of set **P**, where **P** is a set at the lowest level of tree **Q**. The type is chosen from $\mathbf{T}=\{\text{int}, \text{real}, \text{bool}\}$.

4. On sets **M** and **Q**, using functions calculating context-sensitive limitations, we shall record logic equations of **Sem**(sum), accompanying them by informal comments (most of equations is right not only for model *sum*, but also for any model of TADELA).

(1) Only those task objects defined in section **Objects** can be used in model:

$$\mathbf{zond}(\mathbf{M}. < ident, dim >, \mathbf{Q}) = \mathbf{true},$$

where $\mathbf{M}. < ident, dim >$ is projection of set **M** on domains $< ident, dim >$.

(2) The variables of definition area of model should have different identifiers:

$$\left(\mathbf{num}\left(\bigcup_{v\in D}\mathbf{V}\right)=\mathbf{num(D)}\right)=\mathbf{true}.$$

(3) The addend and the condition of inclusion can not be expressed through the result of summation:

$$\mathbf{zond}(\mathbf{A},\mathbf{R})=(\mathbf{zond}(\mathbf{E},\mathbf{R})\text{ or }\mathbf{zond}(,\mathbf{R}))=\mathbf{false}.$$

(4) The variables of sum should be included in addend:

$$\mathbf{zond}\left(\mathbf{E},\bigcup_{i=1}^{num(D)}\mathbf{V}i\right)=\mathbf{true}.$$

(5) Initial values of variables of sum can not be expressed through themselves and/or the result of summation:

$$(\mathbf{zond}(\mathbf{B}_i,\mathbf{V}_i)\text{ or }\mathbf{zond}(\mathbf{B}_i,\mathbf{R}))=\mathbf{false}\quad\forall i=1,\ldots,num(D).$$

(6) Final values of variables of definition area can not be expressed through themselves and/or the result of summation, if they are arithmetic expressions:

$$(\mathbf{zond}(\mathbf{F}_i,\mathbf{V}_i)\text{ or }\mathbf{zond}(\mathbf{F}_i,\mathbf{R}))=\mathbf{false}$$

$$\forall i=1,\ldots,num(D),\ \mathbf{type}(\mathbf{F}_i)<>\text{boolean}.$$

(7) The type of result of summation should coincide with the type of addend:

$$(\mathbf{type}(\mathbf{R})=\mathbf{type}(\mathbf{E}))=\mathbf{true}.$$

(8) The list of definition area is executed from the right on the left: variable of the i-th element of the list cannot be present at any element lying on the right from it:

$$\mathbf{zond}(\mathbf{D}_j,\mathbf{V}_i)=\mathbf{false},\quad\forall j>i,\ i,j=1,\ldots,num(D).$$

The infringement of verity of even one of equations (1)–(8) entails diagnostics of a semantic error in the record of model *sum*. E.g. model:

$$S = \sum (a[i,j] : a[i,j] > 0) \;\; j \text{ from 1 to 5}, \;\; i \text{ from } S \text{ to 6};$$

is correct in syntax, but is not correct in semantics, because the initial value of variable j is expressed through the result S, i.e. logic equation (5) is infringed.

5.3 Algorithmic semantics

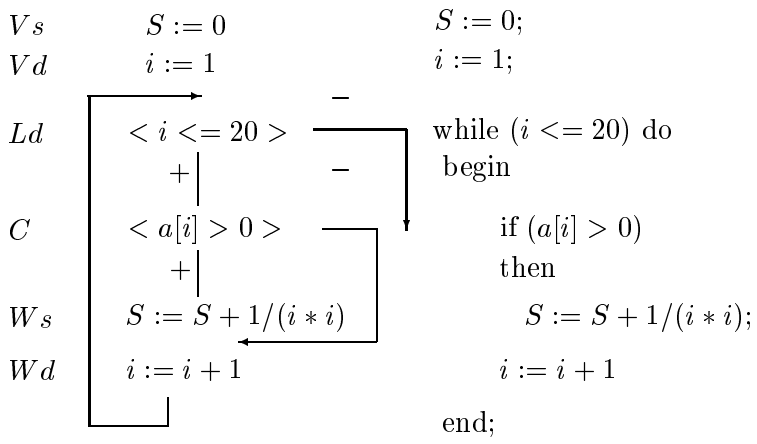
Having a model correctly constructed in syntax and semantics, it is necessary to represent its algorithmic actions in a formal form. On the basis of this form the flow chart of the algorithm and the corresponding fragment of program are to be generated.

We shall consider an example for revealing algorithmic actions of model *sum*: given a real vector $\{a_i\}$, $1 < i < 20$, find the sum of all positive elements of it.

The solving plan contains only one model *sum*:

$$S = \sum (a[i] : a[i] > 0) \qquad i \text{ from 1 to 20};$$

We shall make the flow chart, the fragment of program for it and we shall introduce designations for algorithmic actions:



We consider algorithmic actions in detail:

- V is initialization of a variable. In this case we have Vs and Vd — initializations of the result $S = 0$ and of the variable of definition area $i = 1$;
- W is increment of a variable. Similarly as well as for V we have Ws for the result and Wd for the counter;
- C is checking of a condition;
- L is checking of exit condition of a cycle. In our example it is Ld. The actions Vd, Ld and Wd define the cycle parameters.

We can see from the example that each algorithmic action can have an index specifying model or cycle which it belongs to: s is for sum, d is for cycle, etc. Taking into account that **num**(**D**) may be > 1 (i.e. number of cycles is > 1), each algorithmic action may have a number besides the index. Hence, any algorithmic action is recorded as:

$$< \text{letter} > [< \text{index} >] \quad [< \text{number} >].$$

We call this record an *action identifier*.

Thus, the algorithmic technique of calculation of sum can be expressed through its operator description in the form of a string of action identifiers:

$$Vs\ Vd\ Ld\,(C\,(Ws,\ )\ Wd)\,.$$

We notice that, in the operator description the action C has the form

$$C\,(<T\_\text{part}>,\quad <F\_\text{part}>)\,,$$

where $<T\_\text{part}>$ and $<F\_\text{part}>$ are action identifiers. $<T\_\text{part}>$ is to be executed if the value of the logic expression for C is **true**, $<F\_\text{part}>$ — if it is **false**. One of them can be empty. In such case, if the logic expression for C has corresponding value, the action after C is to be jumped to. Action Ld is recorded as $Ld\,(<\text{cycle\_body}>\,Wd)$, where $<\text{cycle\_body}>$ is a sequence of action identifiers.

We shall define **Alg**(sum) in the form of operator description for model R from section 5.1. It will be defined recursively by number of cycles $k = \mathbf{num}(\mathbf{D})$. Now we shall consider that the condition of inclusion exists, otherwise we need only to replace the action $C(Ws,\ )$ with the action Ws.

1. If $k = 1$

   $$\mathbf{Alg}(sum) = Vs\ Vd1\ Ld1\,(C\,(Ws,\ )\ Wd1)$$

   We denote it as $Vs\ \mathbf{A}1$, where $\mathbf{A}1$ is $Vd1\ Ld1\,(C\,(Ws,\ )\ Wd1)$. Then

   $$\mathbf{Alg}(sum) = Vs\ \mathbf{A}1$$

2. If for $k = n$ we have

   $$\mathbf{Alg}(sum) = Vs\ \mathbf{A}n$$

   then for $k = n + 1$ we have

   $$\mathbf{Alg}(sum) = Vs\ Vdn + 1\ Ldn + 1\,(\mathbf{A}n\ Wdn + 1) = Vs\ \mathbf{A}n + 1$$

Here algorithmic actions have corresponding forms:

$Vs:$ $R := 0;$
$C(Ws,\ ):$ **if** C **then** $Ws;$
$Ws:$ $R := R + E;$
$Vdi:$ $V_i := B_i;$
$Ldi\,(<\text{cycle\_body}>Wdi):$
while F_i **do begin** $<\text{cycle\_body}>;$ Wdi **end**;
$Wdi:$ $V_i := V_i + S_i;$

Thus, we set the algorithmic semantics of model *sum* **Alg**(sum) in general case.

Now we shall consider an example of definition **Alg**(sum) on a concrete model by constructing its operator description. Further we shall "roll" this operator description into flow chart and fragment of program. Let the following ZP be defined: given an integer square matrix 10∗10, find sum of those elements of matrix, which are > 1. In this case syntactically and semanticly correct model *sum* has the form:

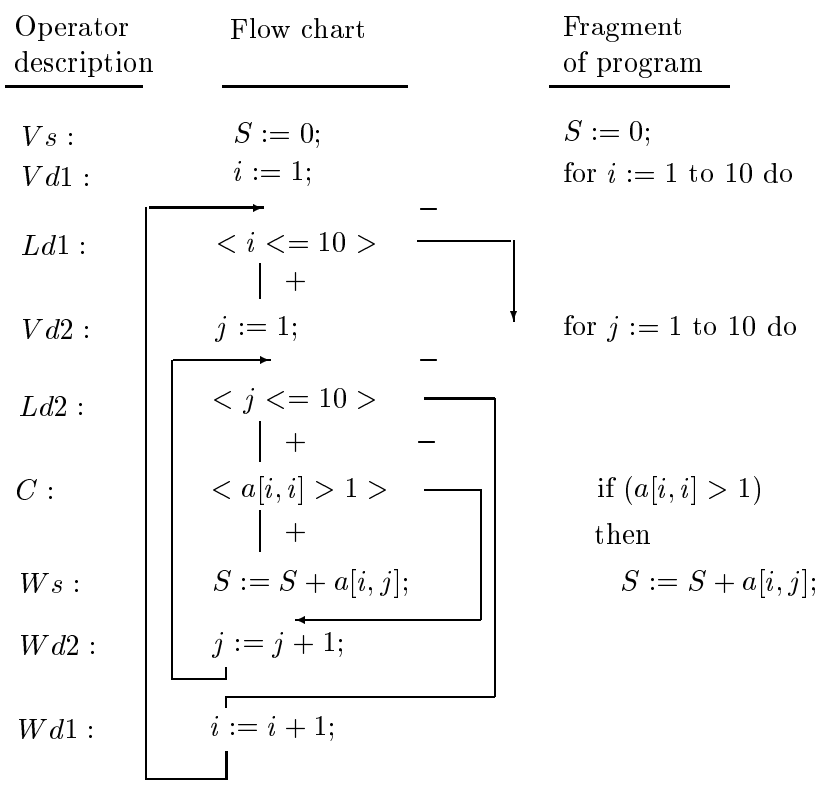
$$S = sum\,(a[i,j] : a[i,j] > 1) \qquad i,\ j \text{ from 1 to 10;}$$

We have $\mathbf{num}(\mathbf{D}) = 2$. Let us define operator description of S:

$k = 1:$ $Vs\ Vd1\ Ld1\,(C\,(Ws,\ )\ Wd1) = Vs\ \mathbf{A}1$

$k = 2:$ $Vs\ Vd2\ Ld2\,(\mathbf{A}1\ Wd2) =$
$= Vs\ Vd2\ Ld2\,(Vd1\ Ld1\,(C\,(Ws,\ )\ Wd1)\ Wd2)$

We get from this operator description the flow chart and the fragment of program:



Thus, having **Sem**(sum) in the form of structural set **Q** and **Alg**(sum) in the form of operator description the system can generate flow chart of the algorithm and fragment of program of calculation of a sum given by a concrete model.

6 Conclusion

In this paper the conceptual bases of training devices on programming and the description language for solving plans of educational computing tasks were considered.

Being simple and close in syntax to algebraic notation, TADELA as the entry language of TDP provides trainee with faster move to computer for problem solving under conditions of his/her superficial ac-

quaintance with the programming language. The study of algorithmic techniques expressed in the form of models is carried out on student's real problems, which essentially increases motivation of learning.

The student's work with TDP acquires research character: solving various problems, the student investigates and learns algorithmic techniques and methods of their software implementation. TADELA has built-in models of 12 basic algorithmic techniques, covering a wide class of computing problems. If the student needs models which are absent, he can expand TADELA, adding to it his/her own models. In this case the student acts as an expert on algorithms and programs.

References

[1] Anderson J.R., Reiser B. The LISP Tutor // Byte. 1985. Vol.10.

[2] Atkinson L., North D. COPAS — a conversational Pascal system // Software — practice and experience. 1981. Vol.11, N.8

[3] Bonar J., Cunningham R., Schultz J. An object-oriented architecture for intelligent tutoring system // ACM SIGPLAN notices. 1986. Vol.21, N.14

[4] Dovgyallo A.M. Dialogue between the user and the computer. Bases of designing and implementation. Kiev, Naukova dumka, 1981 (Russian)

[5] Dovgyallo A.M., Yushenko E.L. An introduction to computer-based conversational programming // Control systems and machines. 1974, N.1 (Russian)

[6] Fooksman A.L. An operational approach to algorithmic languages // N. 3155-75 Dep. in VINITI – Rostov-on-Don, 1975 (Russian)

[7] Polya G. Mathematical discovery. New York-London, Vol.1, 1962, Vol.2, 1965

[8] Popov E.V., Firdman G.R. The algorithmic bases of intellectual robots and artificial intelligence. Moscow: Nauka, 1976 (Russian)

[9] Ryngach V.D. and al. Computer-based systems for teaching programming. Designing and implementation. Chisinau, Shtiintsa, 1989 (Russian)

[10] Ryngach V.D. Training device on programming // Computer technology of training. Hand-book. Kiev: Naukova dumka, 1992 (Russian)

[11] Semantics of programming languages. Collected articles. Moscow: Mir, 1980 (Russian)

[12] Wirth N. Systematic programming. An introduction. Prentice Hall, Englewood Cliffs, New Jersey, 1973.

[13] Yushenko E.A. and al. The computer-based system for learning Cobol language // Cybernetics, N.4, 1973 (Russian)

Phuong Dat Phan,
Dept. of Mathematics and Informatics
Moldavian State University
60, Mateevici str., Kishinev
MD-2009 Moldova
E-mail: *dat@usm.md*

Received Martch 10, 1998