AUTOMATIC PROGRAMMING SYSTEM WITH ACCESS IN THE NATURAL LANGUAGE

Jan Kazimierczak[1]

Abstract. In this paper, acquisition of knowledge from user's programs written in Pascal to the knowledge base of the computer is introduced. It is shown that system with such a kind of knowledge can be used as the automatic programming system. First, representation of knowledge acquired by the computer from different programs is characterized. Then searching for pieces of knowledge in the knowledge base needed for synthesis of a program specified in user requirement is described. Finally, the construction of a new program from the found pieces of knowledge is shown.

Keywords. Artificial intelligence, automatic programming, knowledge representation

1 Introduction

One of the more important problems in the area of Artificial Intelligence is automatic programming. One of approaches to automatic programming that could be practically used we want to introduce in this paper.

Our approach to automatic programming is such that on the basis of user's requirement expressed in natural language the computer performs the synthesis of a program to solve the user's task mentioned in the requirement. In order to accomplish that approach, the computer possesses suitable knowledge in its knowledge base (KB). That knowledge is acquired by the computer from some users programs written in Pascal. These programs are purposely entered into the computer by the knowledge engineer to construct the initial knowledge base.

As the computer should construct a user program on the basis of a requirement provided by a user in the form of natural language sentences, in the proposed solution the KB consists of two parts. The first part contains the knowledge necessary for the natural language understanding. Such a knowledge is acquired by the computer from comments included in user programs. The second part of the KB contains the knowledge needed to synthesize user programs. Such a knowledge is acquired from algorithms structures, sequences of opcodes and declaration statements.

The Automatic Programming System with such kind of the KB behaves in such a way that user's requirement is given on the first part of KB. The computer transforms this requirement into representation of its semantics.

[1] Wrocław University of Technology
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mail: kazim@ict.pwr.wroc.pl

Then computer checks whether this semantics has its counterpart in the first part of KB. If such a semantic counterpart is included there then symbol c_j of its semantics is entered into the second part and causes automatic synthesis of user program from pieces of knowledge included in this part. The above described bahaviour of automatic programming system is shown in Figure 1.

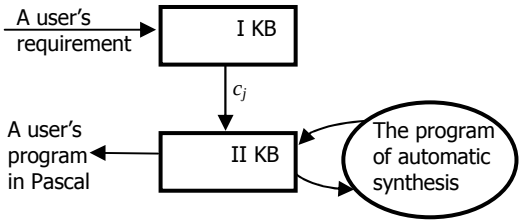

Figure 1. The behavior of the automatic programming system

The leading idea of the presented approach to automatic programming arises from two facts. The first fact tells us that in each user program written in a programming language there are human knowledge about a problem and method of solving it. We well know that the present computer does not remember the knowledge included in user program. When executing a given user program is completed by the computer, the program is erased from computer memory. Hence, after erasing the user program, the computer does not know what problem was solved and does not remember the human knowledge that was concealed in the executed program. According to the above remarks, the computer should acquire the knowledge from user programs to own knowledge base. The second fact tells us that user programs written in a programming language consist of finite number of components such as structural components of algorithm, groups of structural components, structural components of declarations, kinds of operation codes and so on. Note that the same components appear in different programs. The above mentioned components can be considered as small pieces of knowledge which, after transforming into symbolic form, should be stored in the KB. From those pieces of the knowledge a user program can be constructed.

In the proposed automatic programming system we distinguish two cases of the computer behavior during automatic synthesis of a user program. The first case is retrieval of such a program from which the knowledge was acquired for the KB. The second case is creation of a new program from pieces of knowledge in the KB acquired from another different programs.

The first case occurs when user's requirement has its semantic counterpart in the first part of the KB. The second case occurs when a user's requirement is not understood by the computer, because for the requirement there is no its semantic counterpart in the KB. This means that

the computer has not knowledge for synthesis a program indicated with the requirement. In this case the user must extend his requirement by writing titles of subtasks belonging to the task expressed in the requirement or by writing name of action to be performed by the computer and names of objects appearing in the action. In this paper we consider only the second case.

2 Representation of Natural Language Sentences and its Using in Automatic Programming System

The knowledge acquired from programs written in Pascal is divided by the computer into small pieces of knowledge. To such a piece of knowledge a sentence in natural language is assigned. This sentence describes the role of the given piece of knowledge in the program. The division of knowledge included in a program into pieces is determined by positions of comments in the program. These comments, as the natural language sentences, are transformed by the computer into their symbolic representation in the first part of the knowledge base (IKB).

At first, the computer codes the words appearing in a given comment using symbols d_i (i = 1,2,...). Assignments of symbols d_i to words are written down in the dictionary $\overline{D}$. Together with the dictionary $\overline{D}$ the other dictionary $\overline{\overline{D}}$ is formed in which the words are written down in alphabetical order and to each word a symbol w_j (j = 1,2,...,9) of lexical category is assigned. On the basis of lexical categories w_j each sentence is divided into groups of words according to some rules [5]. For example, suppose that the KB is empty and the main comment in a program from which the knowledge is acquired has the form as follows:

This program determines the best move in chess from a given board position (*i*)

After coding words by symbols d_i the sentence (*i*) is divided into the following groups of words:

$d_1,d_2 = a_1 \leftrightarrow$ *This program*;

$d_3 = a_3 \leftrightarrow$ *detrmines*;

$d_4,d_5,d_6 = a_3 \leftrightarrow$ *the best move*;

$d_7,d_8 = a_4 \leftrightarrow$ *in chess*;

$d_9,d_{10},d_{11},d_{12},d_{13} = a_5 \leftrightarrow$ *from a given board position*.

Each sequence of symbols d_i can be introduced in the form a graph. For example, the graph represented the sequence $d_4,d_5,d_6 = a_3$ has the form shown in Fig. 2. On the basis of the above graphical interpretation the sequence a_3 is transformed by the computer into the symbolic expression $D^+(a_3)$,

$$D^{+}(a_3)=^{0}(d_4{}^{1}(z_1a_3d_5{}^{2}(z_1a_3d_6{}^{3}(z_3a_3d_4)^{3})^{2})^{1})^{0}$$

where the symbol z_1 denotes the space between symbols d_j and z_3 means the connection between the last and first symbol d_j in the given sequence. The symbolic expressions $D^{+}(a_i)$, representing groups of words are combined together to obtains only one symbolic expression D'^{+} representing the set of groups of words.

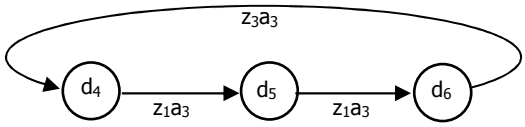

Figure 2. Graph representing the sequence $d_4,d_5,d_6=a_3$

The sentence (*i*) being the comment in a program is treated by the computer as the sequence s_j of symbols a_i denoting names of groups of words. Namely the sentence (*i*) is considered as follows:

$$s_1 = a_1, a_2, a_3, a_4, a_5$$

The sequence s_1 is transformed by the computer into the following symbolic expression $A^{+}(s_1)$,

$$A^{+}(s_1)=^{0}(a_1{}^{1}(z_1s_1a_2{}^{2}(z_1s_1a_3{}^{3}(z_1s_1a_4{}^{4}(z_1s_1a_5{}^{5}(z_3s_1a_1)^{5})^{4})^{3})^{2})^{1})^{0} = A'^{+}$$

The symbolic expression A'^{+} represents in the KB the set of natural language sentences. In the considered case the expression A'^{+} represents the set consisting of only one sentence (*i*).

The symbolic expressions A'^{+} and A'^{+} are stored in the first part of KB where they represent knowledge about the computer knowledge in the domain of automatic programming. As it is known to the computer with such knowledge a user's requirement expressed in natural language is entered. This requirement is transformed into such form of representation which is stored in the first part of KB. If the sentence being user's requirement is the same as a sentence represented in the first part of KB then this means that the computer is able to construct a program adequate to the given requirement. In this case we say the user's requirement has the same semantics c_j as a sentence in the first part of KB and symbol c_j (see Fig. 1) initiates the synthesis of a program from pieces of knowledge included in the second part of KB. Notice, that many natural language sentences can have the same semantic meaning c_j. Each sentence represented in the first part of KB has its semantic meaning c_j. If to the computer such a sentence is entered as not represented in the first part of KB the computer should performs some operations both on the representation of the requirement and the representation of sentences in the KB. These operations are made in order to check whether the requirement has the same semantic meaning as one of sentences represented in the first part of KB. However, most

often the result of these is unsatisfactory because in the requirement another words can be included than words included in a sentence which has the same semantics as the requirement. In order to avoid difficulties in the indification of semantic meaning of user's requirement two forms of knowledge representation in the first part of KB should be stored, i.e. representation of sentences and representation of their semantics.

To obtain representation of the semantics to each group of words in the given sentence the semantic category is assigned. In this way the representation $s_1 = a_1, a_2, a_3, a_4, a_5$ of the sentence (*i*) is transformed by the computer into the following sequence of semantic categories:

AGENT ACTION RESULT OBJECT ELEMENTS

For the purpose of formal representation of this sequence, to each pair of semantic categories a symbol u_j ($j = 1,2,\ldots$) denoting this pair is assigned. According to this the above sequence is transformed into the following form:

AGENT u_1 ACTION u_2 RESULT u_3 OBJECT u_4 ELEMENTS u_0 (1)

where u_0 denotes the end of sequence and connection of it with beginning. Assignments of symbols u_j to pairs of semantic categories are stored in the table $\overline{U}$. The sequence (1) obtains the symbolic name h_1 and the symbol c_1^* of semantics on the level of semantic categories. Then the sequence (1) is transformed into symbolic expression $\tilde{U}'^{+}$ as follows:

$$\tilde{U}'^{+} = {}^{0}(u_1{}^{1}(c_1^* h_1 u_2{}^{2}(c_1^* h_1 u_3{}^{3}(c_1^* h_1 u_4{}^{4}(c_1^* h_1 u_0{}^{5}(c_1^* h_1 u_1)^{5})^{4}\ldots)^{1})^{0}$$

In the first part of KB the expression $\tilde{U}'^{+}$ represents the set of sequences of the type (1), in this case this set contains only one sequence.

After obtaining the representation of the sentence (*i*) on the level of semantic categories the computer determines the semantics of sentence (*i*) on the level of semantic features. The words which constitute semantic meaning within groups are replaced by semantic features. Under concept of semantic feature we mean a set of words having the same semantic meaning. For example, the words *boy*, *girl*, *women*, ... have the semantic feature *human*. There are the following semantic feature in the sentence (*i*):

b_1 = regulation (for "program"),
b_2 = MBUILD (for "determines"), in agreement with CD theory,
b_3 = move,
b_4 = game (for "chess"),
b_5 = position (for "board position").

After replacing semantic categories appearing in the sequence (1) by semantic features the following sequence is obtained:

$$\underline{b_1u_1, b_2u_2, b_3u_3, b_4u_4, b_5u_0 \leftrightarrow c_1}$$

where symbol c_1 denotes the semantics of the sentence (*i*). This sequence is transformed into the symbolic expression B'^{+},

$$B'^{+} = {}^{0}(b_1{}^{1}(u_1c_1b_2{}^{2}(u_2c_1b_3{}^{3}(u_3c_1b_4{}^{4}(u_4c_1b_5{}^{5}(u_0c_1b_1)^{5})^{4}\ldots)^{1})^{0}$$

The expression B'^{+} represents the set of semantics of sentences, it will be extended by sequences represented semantics of another sentences entering to the computer. The relation between representation s_1 of the sentence (*i*) and representation c_1 of its semantics has the following form:

$$u_1a_1, u_2a_2, u_3a_3, u_4a_4, u_0a_5 \leftrightarrow s_1$$

This sequence is transformed into the symbolic expression U'^{+} as follows:

$$U'^{+} = {}^{0}(u_1{}^{1}(a_1s_1u_2{}^{2}(a_2s_1u_3{}^{3}(a_3s_1u_4{}^{4}(a_4s_1u_0{}^{5}(a_5s_1u_1)^{5})^{4}\ldots)^{1})^{0}$$

The expression U'^{+} represents the set of relations between representation of sentences and representation of their semantics.

Suppose now that in the first part of KB the representation of sentence (*i*) and representation of its semantics are stored and to the computer the following user's requirement is entered:

Compute the best walk strategy for a given configuration on chessboard (*ii*)

This sentence is transformed by the computer into the following symbolic representation:

$$s_2 = a_6, a_7, a_8, a_9$$

The sentence (*ii*) is divided by the computer into four groups of words as follows:

a_6 = compute
a_7 = the best walk strategy
a_8 = for a given configuration
a_9 = on chessboard

The following sequence of semantic categories assigned to groups of words is determined:

ACTION u_2 RESULT u_5u_5 ELEMENTS u_6 OBJECT u_0

The sequence u_2, u_5, u_6, u_0 is searched in the symbolic expression $\tilde{U}'^{+}$ but it is not included there in agreement with assumption given above. Notice that some sequences of semantic categories have the same semantics c_j^{*}. For example, the sequence

ACTION u_7 OBJECT u_4 ELEMENTS u_8 MANNER u_0

has the same semantics c_j^* as the sequence

AGENT u_1 ACTION u_7 OBJECT u_4 ELEMENTS u_8 MANNER u_0 (2)

There is table $\overline{V}$ in the computer memory where equivalent of subsequences of symbol u_i is written down. In this case in the table $\overline{V}$ the equivalent $u_1 \leftrightarrow u_1, u_7$ is written down. In agreement with the above interpretation of equivalent of semantic categories sequences, the symbols u_j appearing in the sequence of semantic categories derived from the user's requirement (*ii*) are searched for in the table $\overline{V}$. The equivalents $u_2 \leftrightarrow u_1 u_2$ and $u_5, u_6 \leftrightarrow u_3, u_4$ are found in the table $\overline{V}$. According to this, the sequence u_2, u_5, u_6, u_0 included in the sequence (2) of semantics categories is transformed into the sequence u_1, u_2, u_3, u_4, u_0. This sequence is compared with sequence of symbols u_j included in the sequence (1) of semantic categories. Both sequences are identical what denotes that the user's requirement has the same semantics c_1^* as the sequence (*i*) represented in the KB. At the next step the computer determines the semantics of the user's requirement (*ii*) on the level of semantics features. The following sequence is obtained:

$$\underline{b_2 u_2, b_3 u_5, b_5 u_6, b_4 u_0} \qquad (3)$$

This sequence is searched for in the symbolic expression B'^{+}, but it is not included there. Therefore the computer performs jump to the table $\overline{V}$ where equivalents $u_2 \leftrightarrow u_1, u_2$ and $u_5, u_6 \leftrightarrow u_3, u_4$ are found. According to this, the above sequence (3) of semantic features is transformed into the sequence

$$\underline{b_* u_1, b_2 u_2, b_3 u_3, b_4 u_4, b_5 u_0}$$

This sequence is searched in the symbolic expression B'^{+}. It is included there and has assigned to itself the symbol c_1 of semantics of the sentence (*i*) and $b_* = b_1$. This fact denotes that user's requirement (*ii*) has the same semantics c_j as the comment of a program and the computer is able to constructs this program.

3 Representation of Knowledge included in Computer Program

During knowledge acquisition the algorithm included in the program is split into small parts. To each of such small parts a comment with semantics c_j is assigned. Then each small part of the algorithm is split into algorithm structure and the sequence of opcodes. To algorithm structure in small part and to the sequence of opcodes in this part the symbol c_j of semantics is attached. Algorithm structure is divided into structural components. Structural components of algorithm form a set of structural components in

KB. From structural components acquired from small part with semantics c_j the group with name e_i of structural components is constructed. It represents in KB the algorithm structure with semantics c_j. From groups e_i of structural component the set is formed in KB. Acquisition of knowledge from algorithm is shown in Figure 3.

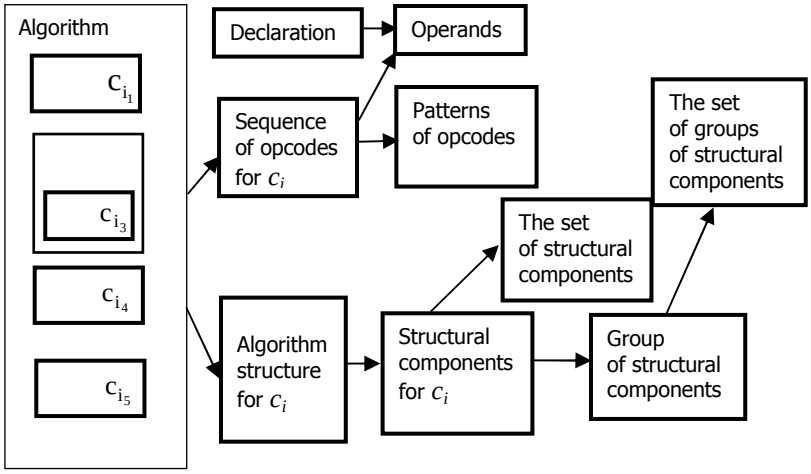

Figure 3. Acquisition of knowledge from algorithm in Pascal

A structural component of algorithm is defined as a sentence in Pascal that begins with one of the following words: **begin**, **for**, **if**, **while**, **repeat**, **case**. If in the given component there is other component then it is removed and replaced by the word "next". Moreover, the place occupied in the given component by opcode is replaced by the word "op". For example, let's consider the following fragment of an algorithm

```
for i:=1 to n do
    begin
        s:=s+a[i];
        diff:=a[i] – avg;
    end
```

It will be represented by two structural components as follows:

for op **do** next – $f_i u_3$

begin op **end** – $f_{i+1} u_0$

where f_i (i = 1,2,...) is symbolic name of structural component and the symbol u_j (j = 1,2,...) indicates the structure of the given component f_i. The structural components of algorithms are transformed by the computer into symbolic forms. The different symbolic forms of components are combined together and create in KB the set expressed in the form of a symbolic expression B'^{+}. From sequence of structural components f_i included in the

given functional part of algorithm a group e_n of structural components is constructed. It represents in KB the algorithm structure of the given functional part The set of those groups is stored in the second part of KB in the form of a symbolic expression F'^{+}. Relations between the first part of the KB and the second one are included in a expression C'^{+}.

The declaration of a program, from which the knowledge is acquired, is transformed by the computer into another form called further the form 2. Namely, names of constants are coded by the symbol t, variables are coded by the symbol q and names of types are coded by the symbol o. For example, the statement **var** i, j: 1..n is transformed into **var** q, q: 1..t where term q, q: 1..t we call a structural component of declaration. The words and terms appearing in a given structural component are coded by symbols l_j (j = 1,2,...). Each structural component of declaration treated as the sequence of symbol l_j obtains the symbolic name k_i (i = 1,2,...). In the KB the set of structural components is stored in the form of symbolic expression L'^{+}. The sequence of symbols k_i is split by the computer into groups of structural components assigned to the words **const**, **type** and **var**, respectively. Each group obtains symbolic name h_i (i = 1,2,...). The set of groups h_i is stored in the KB in the form of a symbolic expression K'^{+}. The declaration of a given program g_i is treated by the computer as the sequence $h_{i_1}, h_{i_2}, h_{i_3} \leftrightarrow c_j$, where c_j denotes the semantics of the main comment in the program g_i. The set of sequences $h_{i_1}, h_{i_2}, h_{i_3}$ is stored in the KB in the form of a symbolic expression H'^{+}. Notice that a given structural component k_i can appear in many different groups h_i. Similarly a given group h_i can appear in many different declarations.

During the knowledge acquisition the names of operands defined and declared in a given declaration are replaced by names k_i of structural components. According to this the operands appearing in opcodes are replaced by symbols k_i. Each operation code is represented in the KB by two components. The first component is the kind of opcode to which the given operation code belongs, this kind we called a pattern of opcode. The second component consists of operands, denoted by symbols k_i, which should be inserted into the pattern of opcode. For example, the operation code j:=1 to n which is included in the instruction **for** is transformed into the pattern m_1:=1 to m_2, where symbols m_1, m_2 denote virtual operands. During knowledge retrieval those virtual operands are replaced by real operands in agreement with the retrieved form 2 of the given declaration. The real operands are denoted by symbolic names k_i of such structural components of the given declaration as indicate these operands.

The patterns of opcodes obtain symbolic names r_i (i=1,2...) and are stored in Table $\overline{R}$. The sequence of names r_i attached to sequence of opcodes in the given functional part of algorithm with semantics c_j is inserted into a symbolic expression R'^{+}. This expression represents the set of all sequences of opcode patterns appearing in programs from which the

knowledge was acquired to the KB. Together with the expression R'^{+} a expression $\dot{K}'^{+}$ representing the set of sequences k_j of operands named by symbols k_i is stored. The relations between the semantics c_j of the given functional part of algorithm and the first symbols r_i and k'_j assigned to this part are stored in Table $\overline{C}$.

4 Synthesis of a New Program

Synthesis of a new program is that the computer constructs a program from small pieces of knowledge acquired from another different programs. Let us assume that a user's requirement entered into computer has not its semantic counterpart in the first part of the KB. In this case the computer asks the user to divide the problem included in the requirement into subproblems. According to this, the user writes new requirement consisting of sentences defining subproblems of problem to be solved by computer. Assume that the new requirement consists of two sentences s_{i_1} and s_{i_2} referring to some subproblems. The computer checked that semantic meaning of the sentences s_{i_1} and s_{i_2} have their semantic countparts c_{j_1} and c_{j_2} in the first part of KB. Actions of the computer during understanding user's requirement are shown in Figure 4. For simplify we assume that $c_{j_1} = c_3$ and $c_{j_2} = c_{19}$. The symbols $c_{j_1} = c_3$ and $c_{j_2} = c_{19}$ are searched for in the symbolic expression C'^{+},

$$C'^{+} = {}^{0}(\ldots c_3{}^{k}(e_3 g_1 c_{m_1}, \ldots, c_{19}{}^{t}(e_{11} g_3 c_{m_2} \ldots \qquad (4)$$

On the basis of the expression C'^{+}, the computer finds out that the algorithm to solve subproblem with semantic meaning c_3 was included in the program g_1 from which the knowledge was acquired to the KB. Similarly, the algorithm to solve subproblem with semantics c_{19} was included in the program g_3. We assume that, before transforming into symbolic form in KB, the part with semantics c_3 of algorithm in the program g_1 had the original form as follows:

```
for i:= 2 to n do        {...c3...}
    begin
       for j:= n downto i do
            if a[j-1]>a[j] then
               begin        {...c4...}
                    x:= a[j-1];
                    a[j-1]:= a[j];
                    a[j]:= x
               end
    end
```

Similarly, the part with semantics c_{19} of algorithm in the program g_3 had the original form as follows:

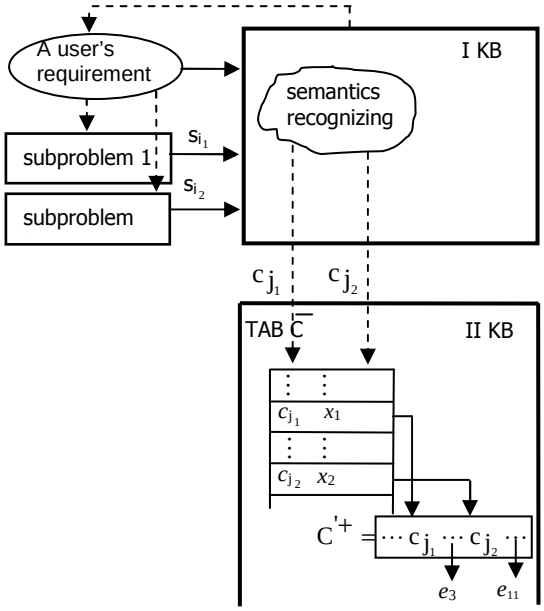

Figure 4. User's requirement understanding

```
begin                          {...c19...}
   total:=0;
   for  i:= 1  to numelts do
      begin                    {...c20...}
          read (num[i];
          total:= total + num[i]
      end
    avg:= total/numelts;
    writeln ('average is' : avg)
end
```

From the expression C'^{+} (4) it is followed that the group of structural components f_i of the algorithm structure for subproblem with the semantics c_3 is denoted by the symbol e_3 and for subproblem with c_{19} is denoted by the symbol e_{11}. In the second part of the KB there are stored Table $\overline{E}$ and the symbolic expression F'^{+} representing the set of structural components f_i. The Table $\overline{E}$ ($e_r - x_n - f_i$) consists of three columns, in the first one the symbol e_r is stored, in the second one the address x_n of e_r in E'^{+} is stored, in the third one the symbol f_i of the first structural component in the group e_r of components is stored. We assume that the expression F'^{+} has the form as follows:

$$F'^{+} = \ldots f_4{}^4(u_3e_{11}f_*, u_3e_3f_6, u_3e_3f_5{}^5(u_2e_3f_0, u_1e_3f_4)^5)^4\ldots)^1, f_6{}^1(u_3e_3f_*{}^2$$
$$(u_*e_3f_5, u_*e_{11}f_9{}^3(u_2e_{11}f_0, u_5e_{11}f_{10}, u_1e_{11}f_4)^3)^2)^1, f_{10}{}^1(u_7e_{11}f_9)^1\ldots)^0$$

On the basis of Tables $\overline{E}$ and $\overline{F}$, and the symbolic expression F'^{+} the computer determines sequences of structural components f_i forming the groups e_3 and e_{11} as it is shown in Fig. 5. These sequences have the following forms:

$f_4u_3, f_5u_1, f_4u_3, f_6u_3, f_*u_*, f_5u_2, f_0$ for $s_3 - c_3$
$f_9u_1, f_4u_3, f_*u_*, f_9u_5, f_{10}u_7, f_9u_2, f_0$ for $s_{11} - c_{19}$

The pairs f_*u_* denote that in the considered groups of structural components f_i there are also another groups. We assume that in the sequence e_3 and e_{11} there is the group $e_4 = f_7u_0$. The sequences e_3 and e_{11} are combined together. On the basis of Table $\overline{F}$, Table $\overline{B}$ and the symbolic expression B'^{+} each term f_iu_j appearing in the sequence is translated into the sequence of words in Pascal. As the result the following algorithm structure of the new program is obtained:

```
f11u4 – begin
f4u3  –   for op do     {...c3...}
f5u1  –       begin
f4u3  –           for op do
f6u3  –               if op then
f7u0  –                   begin op end    {...c4...}
f5u2  –       end
f9u1  –   begin         {...c19...}
              op
f4u3  –       for op do
f7u0  –           begin         {...c20...}
                      op
                  end
f9u5  –       opp
f10u7 –       writeln op
f9u2  – end
f11u2 – end
```

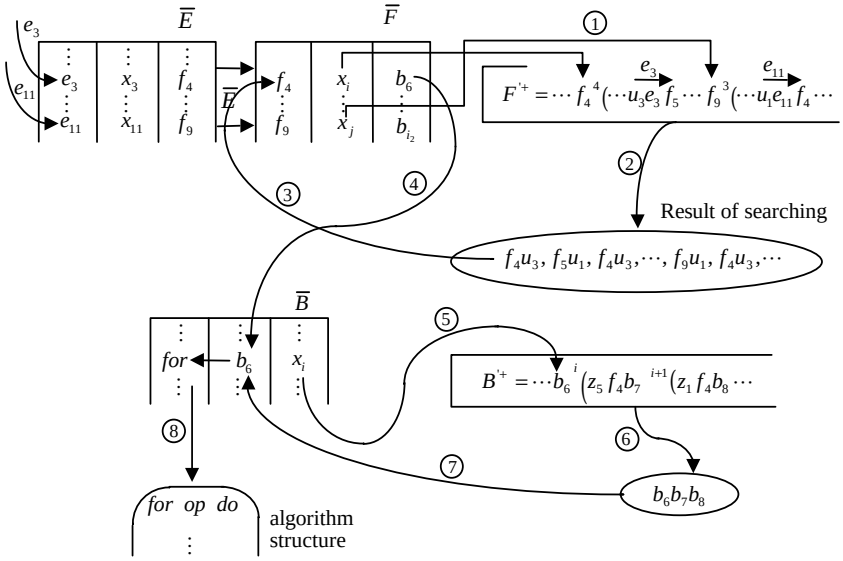

Figure 5. Synthesis of the algorithm structure of a new program

Some operations performed by the computer during synthesis of declaration and operations codes are shown in Figure 6. Having the algorithm structure of the new program, the computer proceeds to construct the declaration and opcodes which should be inserted into places occupied by symbols “op”. Because the information about declaration is included in the representation of operation codes at first the representation of sequence opcodes assigned to the part with the semantics $c_j = c_3$ is considered. Recall that each opcode is represented by a pattern r_i of opcode and sequence k'_j of operands. The sequence of symbols r_i attached to the semantics c_3 is concealed in the symbolic expression R'^{+}, whereas the sequence of symbols k'_j is concealed in the symbolic expression $\dot{K}'^{+}$. In the considered example we assume that the symbolic expressions R'^{+} and $\dot{K}'^{+}$ have the form as follows:

$$R'^{+} = {}^{0}(\ldots r_3{}^{3}(z_3c_2r_1, z_2c_{20}r_{13}{}^{4}(z_3c_{20}r_3)^4)^3)^2)^1, r_4{}^{1}(z_1c_3r_5{}^{2}(z_1c_3r_6{}^{3} (z_3c_3r_4)^3)^2)^1, r_7{}^{1}(z_2c_4r_8{}^{2}(z_2c_4r_9{}^{3}(z_3c_4r_7)^3)^2)^1, r_{10}{}^{1}(z_1c_{19}r_2, z_3c_{19}r_{11})^2)^1, \ldots)^0 \quad (5)$$

$$\dot{K}'^{+} = {}^{0}(\ldots k'_3{}^{3}(z_3c_2k'_1, z_3c_3k'_4{}^{4}(z_1c_3k'_5{}^{5}(z_1c_3k'_3)^5)^4)\ldots)^1, k'_9{}^{1}(z_1c_{19}k'_{10}{}^{2} (z_1c_{19}k'_{11}{}^{3}(z_1c_{19}k'_{12}{}^{4}(z_3c_{19}k'_9)^4)^3)^2)^1, k'_{13}{}^{1}(z_2c_{20}k'_{14}{}^{2}(z_3c_{20}k'_{13})^2)^1, \ldots)^0 \quad (6)$$

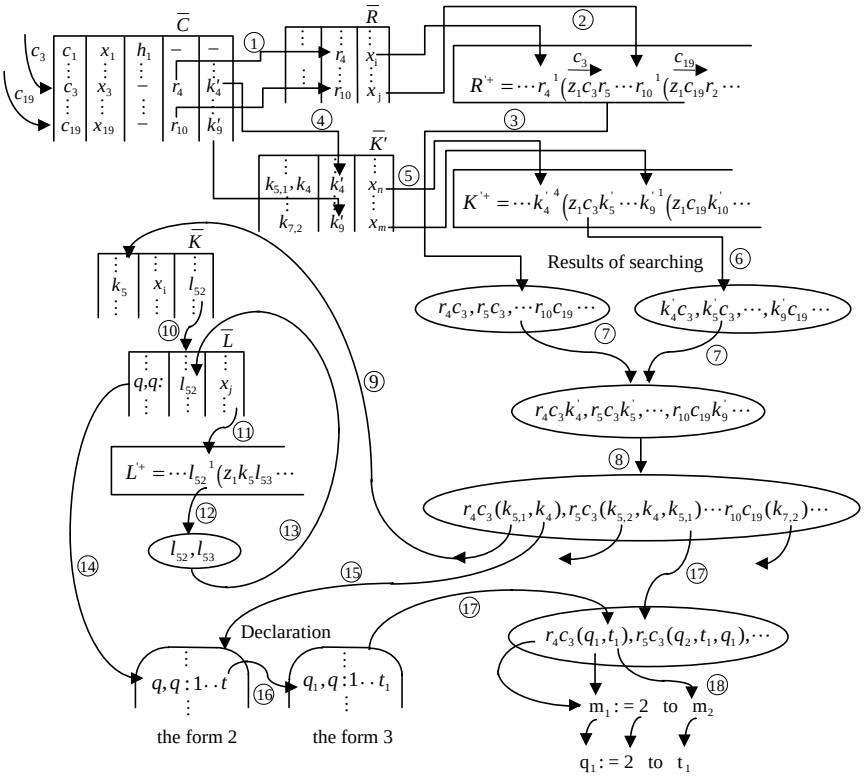

Figure 6. Synthesis of the declaration and operation codes

On the basis of Table $\overline{C}$, where there is the assignment $c_3 \leftrightarrow r_4, k_4'$, the computer finds out that the first pattern of opcode in the sequence of opcodes attached to the small part of algorithm with the semantics c_3 is r_4 and the sequence of operands which should be inserted into r_4 is k_4'. Having the first opcode pattern r_4 for c_3 the computer derives from the expression R'^{+} the following sequence of opcode patterns:

$$r_4c_3,\ r_5c_3,\ r_6c_3 \tag{7}$$

Similarly as above, from the expression $\dot{K}'^{+}$ the following sequence is derived:

$$k_4'c_3, k_5'c_3, k_3'c_3 \tag{8}$$

The sequences (7) and (8) are put together, what results in the following sequence:

$$R'(c_3) = r_4c_3k_4', r_5c_3k_5', r_6c_3k_3' \tag{9}$$

Next the symbols k_i' in (9) are replaced, on the basis of Table $\overline{K}'$ ($k_{i_1}k_{i_2} \leftrightarrow k_j'$), by the symbols k_i being names of structural components of the declaration. As the result the sequence $\breve{R}(c_3)$ is obtained:

$$\breve{R}(c_3) = r_4c_3(k_{5,1}, k_4), r_5c_3(k_{5,2}k_4, k_{5,1}), r_6c_3(k_6, k_{5,2}) \qquad (10)$$

Symbols k_i appearing in brackets are written down in Table $\overline{K}$, where to every symbol k_i the first symbol l_j in the sequence $k_i = l_{j_1}l_{j_2}l_{j_3}$ is assigned. The sequences of symbols l_j attached to symbols k_i are searched for in the symbolic expression L'^{+}. The structure of L'^{+} has the form:

$$L'^{+} = {}^{0}(\ldots l_{52}{}^{1}(z_1k_5l_{53}{}^{2}(z_3k_5l_{52})^{2})^{1}), l_{54}{}^{1}(z_1k_6l_4\ldots \qquad (11)$$

The sequences of symbols l_j derived from L'^{+} are translated, on the basis of Table $\overline{L}$, into words and terms in Pascal. As the result the following fragment of the form 2 of declaration is obtained:

const $t = y \leftrightarrow k_4$
  var q, q: $1..t \leftrightarrow k_5$
    q: array [$1..t$] of integer $\leftrightarrow k_6$
    q: integer $\leftrightarrow k_7$

Symbols t and q obtain indexes in the form integers (1, 2, 3, ..) in ordering determined by symbols k_i appearing in the sequences $\breve{R}(c_3)$ (10) and $\breve{R}(c_4)$. As the result the following form 3 of declaration is obtained:

```
const

  t1 = y;  ↔ k4    {... c6 ...}
  var                                                    (12)
  q1, q2:  1..t 1;  ↔ k5(k5,1, k5,2)
     q3: array [1..t1] of integer; ↔ k6 {... c7 ...}
     q4: integer  ↔ k7
```

Now the symbols k_i appearing in the sequence (10) are replaced by symbols t_1 and q_1, q_2, q_3, respectively, as follows:

$$\breve{R}(c_3) \rightarrow \dot{R}(c_3) = r_4c_3(q_1, t_1), r_5c_3(q_2, t_1, q_1), r_6c_3(q_3, q_2) \qquad (13)$$

Then the symbol r_i appearing in the sequence (13) are replaced, in accordance with Table $\overline{R}$, by patterns of opcodes. Next the symbols m_j appearing in patterns of opcodes are replaced by the real operands, q_i or t_j, appearing in the sequence (13). According to this the following transformation is realized:

$r_4 \rightarrow \boxed{m_1 := 2 \text{ to } m_2} \rightarrow q_1 := 2 \text{ to } t_1$

$$r_4 \rightarrow \boxed{m_1 := m_2 \text{ downto } m_3} \rightarrow q_2 := t_1 \text{ downto } q_1 \qquad (14)$$
$$r_6 \rightarrow \boxed{m_1\,[m_2 - 1] > m_1[m_2]} \rightarrow q_3[q_2 - 1] > q_3[q_2]$$

The obtained opcodes with real operands are inserted into algorithm structure derived from the group e_3 of structural components f_i. These opcodes are inserted in the places denoted by symbols op or opp.

In similar way as above the declaration associated with opcodes included in the small functional parts of algorithm with semantic meaning c_{19} and c_{20} is constructed. From the symbolic expressions R'^{+} (5) and $\dot{K}'^{+}$ (6) the following sequences are derived

$$\breve{R}(c_{19}) = r_{10}c_{19}(k_7), r_2c_{19}(k_{11}[k_8], k_4), r_{11}c_{19}(k_{10,1}, k_7, k_4), r_{12}c_{19}(d_{i_1}, d_{i_2}, k_{10,1})$$
$$\breve{R}(c_{20}) = r_3c_{20}(k_9, k_{11}[k_8]), r_{13}c_{20}(k_7, k_9, k_{11}[k_8]) \qquad (15)$$

Symbols k_i included in brackets are translated into structural components of declaration, on the basis of Table $\overline{K}$, Table $\overline{L}$ and the symbolic expression L'^{+} (11). As the result, a fragment of the form 2 of declaration associated with small functional parts of the algorithm denoted by symbols c_{19} and c_{20} is obtained. The obtained fragment has the form as follows:

```
const
    t = y          k4    {... c6 ...}
type
    o = 1..t…… k8
var                                                    (16)
    q: array [o]  of integer    k9[k8] {... c7 ...}
    q: o                        k11[k8]
    q: integer                  k7      {... c22 ...}
    q: real                     k10,1   {... c23 ...}
```

The form 2 of the declaration (16) obtained for c_{19} and c_{20} is compared with the form 3 of the declaration (12) obtained for c_3 and c_4 to construct one common declaration. Notice that some declaration in a program are defined by comments. During knowledge acquisition the computer assigns to each comment a symbol c_w of its semantics. For example, the places of comments in the declaration (12) and (16) have been denoted by symbol c_w in brackets. The comments included in declarations are used by the computer for constructing the common declaration. For constructing the common declaration some rules are applied. At first, the sequence of symbol k_i appearing in description of the form 2 of the declaration (16) is compared with the form 3 of the declaration 12. According to this, the following sequences are compared:

k_1, $k_4(c_6)$, k_3, k_5, $k_6(c_7)$, k_7 for (12)

k_1, $k_4(c_6)$, k_2, k_8, k_3, $k_9(c_7)$, k_{11}, $k_7(c_{22})$, $k_{10,2}(c_{23})$ for (16)

where symbols k_1, k_2, k_3 denote words **const**, **type**, **var**, respectively. On the basis of some rules the following common sequence is obtained:

$$k_1,\ k_4(c_6),\ k_3,\ k_6(c_7),\ k_5,\ k_{7,1},\ k_{7,2}(c_{22}),\ k_{10,1}(c_{23}) \qquad (17)$$

The common declaration derived from the sequence (17) obtains the following form:

```
const
    t1 = y                          k4
type    o1 = 1..t 1   –  k8
var                                                          (18)
    q1, q2: 1..t1                   k5
    q3: array [1..t1]  of integer   k6
    q4: integer                     k7,1
    q5: integer                     k7,2
    q6: real                        k10,1
```

After inserting the derived operation codes into the algorithm structure (4) the computer obtains the following form of new algorithm:

```
begin
   for  q1 := 2  to t1 do     {... c3 ...}
     begin
        for  q2 :=  t1 downto q1 do
           if  q3 [q2-1] > q3 [q2] then
              begin          {... c4 ...}
                 q4 := q3 [q2-1];
                 q3 [q2-1] := q3 [q2];
                 q3 [q2] := q4;
              end
     end
   begin                            {... c19 ...}
      q5 := 0
      for  q1 := 1  to t1 do
         begin                      {... c20 ...}
            read  (q3 [q1])
            q5 := q5 + q3 [q1]
         end
      q6 := q5/t1
      writeln ('average is': q6)
   end
end
```

This algorithm together with the declaration (18) create a program, which has been derived from pieces of knowledge acquired by the computer from

some programs g_1 and g_3. This program sorts the list of numbers in increasing order and computes the average of those numbers.

5 Conclusions

In this paper a method of synthesis of a new program from small pieces of knowledge acquired from another different programs is introduced. The presented the automatic programming system can be applied in any microcomputer. It is very helpful for a man who solves diverse scientific problems using the same formal tool, e.g. using symbolic expressions representing graphs. In this case the same pieces of knowledge will be appeared in many programs from which the knowledge is acquired to the KB but sequence of these pieces in every program will be different. Under notion "small piece of knowledge" we mean small functional part of algorithm indicated by the comment. The presented automatic programming system is peculiarly very useful in real-time systems when we have to solve urgent problem, but we have not time to write a program to solve this problem by the computer.

References

1. Cooke D.E., Gates A., "On the Development of a Method to Synthesize Programs from Requirement Specifications", International Journal of Software Engineering and Knowledge Engineering 1 (March 1991), pp. 21-38, World Scientific Publishing Co., New Jersey, 1991.
2. Kazimierczak J., "Acquisition and Representation of Knowledge on the Level of Programming Language for Automatic Programming", Proceedings of the ACM Computer Science Conference'93, Indianapolis, pp. 221-228, ACM Press, New York 1993.
3. Kazimierczak J., "Knowledge Representation on the Level of Natural Language for Purposes of Automatic Programming", Proceedings of the 7th Intern. Conference "Software Engineering and Knowledge Engineering", SEKE'95, Rockville, Maryland, USA, KSI Press, Skokie 1995.
4. Kazimierczak J., "Representation of Knowledge for Purposes of Automatic Programming", Proceedings of the 9th Int. Symp. on Artificial Intelligence "ISAI 1996", pp. 240-249, Cancun, Mexico, ITESM Press, Monterrey 1996.
5. Kazimierczak J., "Splitting Natural Language Sentences into Groups of Words for Purposes of Automatic Programming", Report PRE, No. 37/92, I-6, Wroclaw University of Technology Press, Wroclaw 1992.
6. Kazimierczak J., "Acquisition of Knowledge and its Representation to Achieve the Ability of Computer to Automatic Programming", Proceedings of the Intern. Conf. on Artificial Intelligence, Volume II, pp. 819-826, Las Vegas, Nevada, 2002, CSREA Press.
7. Rich Ch., Waters R.C., "The Programmer's Apprentice", Addison-Wesley Publishing Company, Reading, Massachusetts 1990.