PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 34 (48), 1983, pp. 37-47

DISCUSSING GRAPH THEORY WITH A COMPUTER III, MAN–MACHINE THEOREM PROVING

Dragoš Cvetković, Irena Pevac

Abstract. The interactive programming system "Graph" for the classification and extension of knowledge in the field of graph theory has recently been implemented at the University of Belgrade, Faculty of Electrical Engineereing. System "Graph" consists of a computerized graph theory bibliography, a system for graph theoretic alorithms and a mechanical theorem prover. This paper describes the theorem prover of the system "Graph". The main features of the prover are: formalization of graph theory by a first order calculus, well organized files of definitions and lemmas, both interactive (natural deduction) and non-interactive (resolution) work, usage of analogies, man-machine communication by natural language (including sentences to be proved), possibility to use graph theoretic alogithms to prove statements about concrete graphs or to check conjectures on particular graphs including random graphs.

1. Introduction. The interactive programming system "Graph"for the classification and extension of knowledge in the field of graph theory, announced in [**4**], has recently been implemented at University of Belgrade, Faculty of Electrical Engineering. Parts of the system are described in papers [**5, 6, 7, 8, 9, 11, 12**].

System "Graph" consists of a computerized bibliography of graph theory (BIBLI), a subsystem (ALGOR) with the graph theoretic algorithms implemented, and a theorem prover (THEOR). The purpose of the system is to help a qualified researcher in the field of graph theory or its applications.

The system is implemented on a PDP 11/34 computer with two disc drives (RK 05), a typewriter and a graphical video display with a light pen. Because of the small memory space (32K) the use of overlay was necessary. Programming language is FORTRAN IV and the whole package contains about 25000 instructions and about 700 subroutines.

The whole system is supported by such useful modules as memory manager (MEMAN), which controles the main memory and gives the space to subroutines are active in the moment, file manager (FILMAN) whose specific feature is that it is possible to store an arbitrary long record in a file, moduo for displaying messages etc.

AMS Subject Classification (1980): Primary 68 G 15; 03 B 35, Secondary 05 C 99.

The communication with the system is carried out in English. The syntax of commands is simple, and the system can help the user if he does not know their exact form.

In this paper we shall describe the interactive theorem prover contained in the part THEOR. The system contains also a noninteractive theorem prover, based on resolution, with some newly developed induction rules added, which are described in [**11**].

The strategy in building this system was to make it highly interactive. With respect to this we followed some ideas from [**10**]. This enables the user to tell the system everything he knows about the statement which is being proved. As a contrast to the similar systems described in the literature, we did not want to force the system to prove what the user already knows. Of course, we incorporated techniques standard for interactive provers. Our hopes for effectiveness of the prover are based on features specific for graph theory: carefully chosen formalization of graph theory, well organized files of definitions and lemmas, usage of graph theoretic algorithms. There are, as well, some non-specific features such as: communication in a natural language (the sentence which is proved is given by the user in English), and usage of analogies.

Interactive theorem prover is based on splitting procedures described in [**1, 2, 3**].

Experiments with the prover are in due course. Improvements can include some blocks which would automatically do some parts instead of the user.

2. Formalization of graph theory. The graph theory is formalized by a special first order predicate calculus, called "arithmetical graph theory, or briefly AGT. AGT is obtained by extending the formal arithmetics.

There are point, line, and integer variables. They are represented by strings of at most five symbols, first one being always a letter, while the others are digits. (Hence, digits are suffixes of letters and are used instead of indices). The letter indicates the type of the variable according to the following scheme:

X, Y Z—point variables;

U, V W—line variables;

K, L, M, N—integer variables.

Similarly, the strings of the type described above, where the first letter is:

G, H denote graph names;

O denote constants;

F denote function names;

A denote operations over graphs.

Finally, the strings where the letter is one of P, Q, R, S, T denote predicate names. The corresponding predicates are n-placed where $n = 0, 1, 2, 3, 4$, respectively. This convention does not concern predicates carried from arithmetics. Predicates of type $X\sigma Y$ where the sign σ stands for $=, \neq, <, \leq, >, \geq$, are written in standard fashion in formulas, instead of, say, $R(X,Y)$ as the other 2-placed predicates.

All the constants, relations, operations and axioms with the same meaning are carried from the formal arithmetics. AGT has also a basic predicate $S1(X,Y,U)$ with the meaning "points X and Y are joined by the line U" and has constants $O1$ and $O2$ which denote the number of points and the number of lines, respectively. It is assumed that $1 \leq X,Y \leq O1$ and $1 \leq U \leq O2$.

There are no variables for graphs, as we are dealing with the first order theory. If we want to mention the graph considered, we can add the name of the graph as a suffix of a predicate name. So we get the predicate $S1G(X,Y,U)$ with the meaning "points X and Y are joined by the line U in the graph G". The same could be done for constants or functions. So, $O1G$ and $O2G$ denote the number of points in the graph G, and the number of lines in the graph G, respectively.

More generally, the following suffixes could be added to the predicate names:

1. B, 2. $B\$E$, 3. $B\$\$EC$, where B and C are the names of graphs and E is a string denoting a graph operation (unary or binary).

The suffixes are added to point out that the predicate is referred to: 1. the graph with the name B; 2. the graph that is obtained by applying the unary operation E to the graph with the name B; 3. the graph that is obtained by a binary operation E from graphs with the names B and C.

The previously mentioned suffixes could also be added to the constant or function names with the analogous meaning.

All these variables and symbols are given as the user sees them in the formula, but they are internally coded by some integers. These codes are not mentioned here.

The axioms of formal arithmetic are extended by the axioms for $S1(X,\, Y,\, U)$:

1° $(\exists X)(\exists Y) S1(X,\, Y,\, U)$,

2° $S1(X,\, Y,\, U) \wedge S1(X1,\, Y1,\, U) \Rightarrow (X = X1 \wedge Y = Y1) \vee (X = Y1 \wedge Y = X1)$,

3° $\rceil S1(X,\, Y,\, U)$,

4° $S1(X,\, Y,\, U) \wedge S1(X,\, Y,\, V) \Rightarrow U = V$,

5° $S1(X,\, Y,\, U) \Rightarrow S1(Y,\, X,\, U)$.

There are two axiom schemes. If a point variable X is an argument of a predicate whose name has a suffix D, (where D is one of previously defined suffixes), then $1 \leq X \leq O1D$. Similarly, for a line variable we have $0 \leq U \leq O2D$. The user is permitted to introduce a new specific predicate, besides $S1(X,\, Y,\, U)$. This, of course, implies further extensions of the axiom set. That will be discussed latter with more details.

Naturally, the formalization of graph theory by AGT is not the only one possible. The great advantage of AGT is that we can formalize almost all the notions from the "proper graph theory" (i. e. excluding weighted graphs, spectra, etc) within a first order calculus.

3. The knowledge organization. While the system is working, it communicates with the files of: definitions, axioms, lemmas, and logically valid formulas, where the knowledge that the system possesses is situated.

We introduce a new definition by telling the system the semantic category that determines the concept that will be defined, and by typing the definition in the language "proper GTCL" (Graph Theoretical Computer Language). "Proper GTCL" is a part of the natural English language simplified and formalized for the use in graph theory. The syntax of the "proper GTCL" and the corresponding parser are described in [**12**].

The definition is, of course, a sentence where "iff" is the central logical conective. A definiendum is the new concept (predicate) and a definiens consists of concepts (predicates) that were defined before.

Finally, the definition is translated into a formula of AGT. The translation procedure is described in [**5**] and [**7**].

For instance, if we tell the system a new concept: *adjacent points*, with the definition in "proper GTCL": Points X and Y are adjacent iff there exists a line U, such that points X and Y are joined by the line U", the corresponding translation to the first order calculus formula is the following:

$$R1(X,Y) \Leftrightarrow (\exists U) S1(X,Y,U).$$

The name of the new predicate is generated by the system and arguments of this predicate are free variables in the definiens.

Let us mention some other examples of definitions.

Incident point and line: "Point X and line U are incident iff there exists a point Y such that points X and Y are joined by the line U;

$$R2(X,U) \Leftrightarrow (\exists Y) S1(X,Y,U).$$

Points joined by a walk: "Points X and Y are joined by a walk iff there exists a number K such that points X and Y are joined by a walk of length K";

$$R4(X,Y) \Leftrightarrow (\exists K) S2(X,Y,K).$$

(For the definition of the predicate $S2(X,Y,K)$ see below.)

Connected graph: Graph is connected if for all X, Y points X and Y are joined by a walk";

$$P12 \Leftrightarrow (\forall X)(\forall Y) R4(X,Y).$$

Isolated point: "Point X is isolated iff for all points Y points X and Y are not adjacent";

$$Q3(X) \Leftrightarrow (\forall Y) \rceil R1(X,Y).$$

Vertices adjacent in the complement of a graph: "Points X and Y are adjacent in the complement of a graph G iff points X and Y are not adjacent in the graph G and point X is not equal to point Y";

$$R1G\$A1(X,Y) \Leftrightarrow \rceil R1G(X,Y) \wedge X \neq Y.$$

This is an example where the defined adjacency relation refers to the graph that is obtained by a unary graph operation. Once the adjacency relation for a graph obtained by a unary or binary operation is introduced, other notions, which were previously defined or will be defined afterwards, for graphs in general, are automatically extended. So, it is not necessary to define them again for the new graph. If the predicate name appears with suffixes $B\$E$ or $B\$\$EC$, where B and C are the names of graphs and E denotes a unary or binary graph operation, the system is able to generate the corresponding definition using analogy principle.

For instance, if the predicate $Q3H7\$A1(Z)$ appears (its meaning is "Z is an isolated point in the complement of the graph $H7$"), the system will find the definition of an isolated point in the graph G whose formula is

$$Q3G(X) \Leftrightarrow (\forall Y)\rceil R1G(X,Y).$$

Without changing the definition file it will generate the corresponding definition by analogy:

$$Q3H7\$A1(X) \Leftrightarrow (\forall Y)\rceil R1H7\$A1(X,Y).$$

This is done by interchanging the predicate suffixes from the above definition.

Evidently, the same analogy principle is used when the suffix with the name of a graph does not fit to the graph name used in the definition. Let us note, that the name of the graph has the role of a hidden free variable.

The definition file is created in a deductive way. There are basic and derived predicates, the latter being defined by previously defined predicates.

Among basic predicates we have $X = Y$ and $S1(X,Y,U)$, but the user can introduce other basic predicates. For instance, the predicate $S2(X,Y,K)$ mentioned in the definiens of the concept *points joined by a walk* is introduced as a basic predicate, but its actual meaning is specified by the following axioms:

$$S2(X,Y,0) \Leftrightarrow X = Y, \quad S2(X,Y,1) \Leftrightarrow R1(X,Y),$$
$$S2(X,Y,K+1) \Leftrightarrow (\exists Z)(S2(X,Z,K) \wedge (\exists V)S1(Z,Y,V)).$$

Every definition is provided with two sets of pointers. The first set consists of the numbers of definitions where the predicates from the definens are defined and the second one contains the numbers of definitions in which the definiendum of this definition is used in the formula of the definiens. The two sets of pointers provide a faster access to the specifies definition and help direct the process of substituting of definitions.

As we can see, the system is able to "learn" the graph theory. Although the elementary notions of graph theory are introduced into the system, the user can teach it various variants of graph theory as he feels is useful. Different users can develop the definition file towards different areas of graph theory following their own interests.

Using the so developed deinition file the system is able to accept any sentence written in "proper GTCL" which contains the concepts already defined.

Sentences are memorized in the system GRAPH under the names which begin with one of symbols P, Q, R, S and which are given by the user. Sentences are introduced into the system by the command

SET "name"="text",

where "name" means the name of the sentence and text of the sentence is in "proper GTCL".

If the sentence is syntactically correct and contains the concepts known to the system, it will be translated into a formula of AGT. Under the name given the system will memorize the English text of the sentence (appropriately coded), the formula of AGT and a set of pointers to the definitions of predicates occurring in the formula.

A sentence defined in the system can be sent to several files, including the files of axioms and lemmas.

The axiom file contains the sentences that involve basic predicates of AGT. Limitations coming from axiom schemes for point and line variables (see previous section) are generated by the system when necessary.

The file of lemmas contains the sentences that accelerate the proving process. There is a fixed part of this file containing some useful graph theoretical lemmas (for example, symmetry and transitivity of the predicate $R4(X, Y)$ and so on) and this part is protected against user's modifications. On the other hand, the user can add (or delete) some other lemmas if he feels that it would accelerate the proving process.

The user is also permitted to get the definition of some concept if he wishes. A definition can be cancelled but that causes the loss of all definitions which use the definition cancelled.

The file of logically valid formulas differs from the above mentioned files, in which the formulas express some relationship over concrete predicates of AGT. Here, the relationship concerns any predicates or even formulas.

4. Description of the interactive prover. After the sentence is introduced into the system under a name P, we begin the interactive proof by the command:

CREATE [TREE] T PROOF OF [SENTENCE] P.

The words in brackets may be omitted.

The system creates the tree that is the proof of the sentence P. The sentence that is the goal of the proof is split into the subgoals and they further are split into the new subgoals etc. So, we generate the proof tree which is memorized in the system. The proof tree is a rooted tree. The root indicates the currently considered subgoal. The user can move the root by commands as described below. The user is also permitted o tell the system that the considered subgoal is true, and the comment why it is so, is memorized. There is also a resolution based prover incorporated into the system that is described in [**11**] and it can be applied to any subgoal. Of course, the proofs is finished when all the subgolas are proved.

There are the following blocks available to the user for further processing of the current subgoal:

1° Finding subgoals,

2° Case analysis (by P),

3° Forward chaining (by the transition goal P),

4° Simplification of the formula,

5° Extension of the formula,

6° Modification of the formula to an equivalent form,

7° Sending the subgoal to the resolution based prover,

8° The user tells the system that the current subgoal is true,

9° Moving of the root of the tree up, down, to the left, to the right, to the specific point,

10° Displaying the proved and the unproved subgoals,

11° Omitting a subtree of the generated tree,

12° Presenting a current subgoal in the formula form or in English where the sentence is written in "proper GTCL" mentioned earlier,

13° Printing the whole proof tree with all subgoals as sentences in a nice form,

14° Displaying the graphical version of the proof tree.

Some of the blocks will be now explained with more details.

1° *Finding subgoals*. Essentially, this block splits a formula of the form $A \wedge B$ into two subgoals: A and B. This includes the following cases:

$$\begin{array}{ccccccccc} & A \Leftrightarrow B & & & A \Rightarrow B \wedge C & & & A \vee B \Rightarrow C & \\ & / \quad \backslash & & / & & \backslash & / & & \backslash \\ A \Rightarrow B & & B \Rightarrow A & A \Rightarrow B & & A \Rightarrow C & A \Rightarrow C & & B \Rightarrow C \end{array}$$

and some others, similarly, as described in [**2**].

2° *Case analysis*, the current subgoal denoted A is converted into two subgoals:

$$\begin{array}{ccc} & A & \\ / & & \backslash \\ P \Rightarrow A & & \neg P \Rightarrow A \end{array}$$

where P is a newly introduced sentence given by the user. The further improvement of the system includes inserting the heuristics for automatic generating of the sentence P.

3° *Forward chaining by the transition goal P.* The current subgoal of the form $A \Leftarrow B$ is converted into two subgoals:

$$\begin{array}{ccc} & A \Rightarrow B & \\ / & & \backslash \\ A \Rightarrow P & & P \Rightarrow B \end{array}$$

where the sentence P is given by the user. As in previous case the automatic generation of P is planned.

4° *Simplification of the subgoal.* The current subgoal of the form $A \Rightarrow B$ is simplified by omitting the hypothesis A and dealing further with conclusion B. Formula A is moved to the temporary file and it can be added as a conjunct to a hypothesis of some subgoal which is situated below the subgoal $A \Rightarrow B$.

5° *Extending the subgoal.* The sentence P from the temporary file is added to the hypothesis of the current subgoal of the form $A \Rightarrow B$. The situation when this is permitted is just opposite to 4°.

6° *Modification of the subgoal using the equivalent transformations.*

1°° The current subgoal can be transformed to an equivalent form, using the definition file. This can be done by substituting for a predicate by its definiens. This includes the cases when analogy is used, as discussed in Section 3. The opposite case we substitute the definiendum of some definition for a subformula of the current subgoal, provided this subformula "equals" the definiens of the definition. The two formulas "equal" if they coincide, neglecting the arguments of the predicates, the predicate suffixes and superfluous brackets.

2°° The current subgoal can be transformed using an axiom of the form $F_1 \Leftrightarrow F_2$. If one side of the axiom "equals" a subformula of the formula, in the sense described above, it can be substituted by the other side. Of course, the analogy is also included.

3°° The current subgoal can be transformed using a lemma in the same way as described in 2°°

4°° The modification of the current subgoal by means ob a logically valid formula (LVF) is performed if LVF is in the form of an equivalence. Each subformula of the subgoal is compared with both sides of such a LVF. If they are of the same structure, the subformula is replaced by another formula corresponding to the other side of the LVF. For example, De Morgan's law $\rceil(P \wedge P2) \Leftrightarrow \rceil P1 \vee \rceil P2$ would convert the formula

$$(\forall X)(S2(X,Y,K) \wedge \neg(Q1(X) \wedge Q1(Y)))$$

into the formula

$$(\forall X)(S2(X,Y,K) \wedge (\neg Q1(X) \vee \neg Q1(Y))).$$

The coincidence of the structures of a subformula and a side of a LVF is reduced to establishing an isomorphism of the trees representing their structures in a usual way.

The skolemization, in familiar sense, is not implemented, although the effects of skolemization can be achieved by the existing commands. We believe that it contributes to the greater naturallity of the proof procedure.

Example. Experience with the "Graph" theorem prover will be described in another paper. Here we give only an example. The sentence:

"If graph G is not connected, then complement of graph G is connected" which represents a well-known theorem from graph theory, is sent to the prover.

After some dialogue with the system the proof is completed. The printout is given in Appendix.

5. Some special features. The possibility of checking some subgoal using the model is a specific part of this prover. As mentioned in the introduced the system GRAPH also contains the part ALGOR with some graph theoretic algorithms implemented. So, it gives the option of interacting the part THEOR with the part ALGOR. Some subgoal can be verified on a random of a particular graph, first one being generated by the system and the other by the user. If the counterexample is found we conclude that the main goal is not true, or that some subgoals performed by the user's interaction (case analysis, transition goal) are not true.

In the counterexample is not found the user must continue the proof of the main goal.

If the proofs of the theorems of graph theory a subgoal is very often a statement about a concrete graph. If that is the case, the subgoal is proved by activating the corresponding algorithm in the part ALGOR.

Appendix. This appendix represents an abreviated dialog with the system "Graph" which has led to the proof of sentence named below by S.

```
RUN GRAPH
Hello, system "Graph" is at your disposal.
If you do not know how to continue type: INFORM .
Type your next instruction, please.
----------------------------------------------------------------
        This last message, by which the system indicates that
it has performed the previous request of the user, will be
omitted below.Some commands are commented by reffering to
the proof tree given below. Predicates used are defined in
the Section 3.
----------------------------------------------------------------
SET S = "IF GRAPH G IS NOT CONNECTED THEN COMPLEMENT
OF GRAPH G IS CONNECTED" .
CREATE T PROOF OF S .
MODIFY T BY DEFINITION .  (Goal 1 is replaced by subgoal 2)
MOVE ROOT OF T DOWN .  (Root is placed on subgoal 2)
SIMPLIFY T .  (Subgoal 3 is produced)
MOVE ROOT OF T DOWN .  (Root moved to subgoal 3)
DELETE QUANTIFIERS OF T .  (Subgoal 4 is produced)
SET S1 = "POINT X AND POINT Y ARE JOINED BY A WALK IN G" .  (A
sentence enters the system under the name S1)
MOVE ROOT OF T DOWN .  (Root moved to subgoal 4)
MODIFY T BY CASE S1 .  (Subgoal 4 replaced by subgoals 5 and 6)
MOVE ROOT OF T DOWN .  (Root moved to subgoal 5)
SET S2 = ROOT OF T .  (Subgoal 5 extracted from the tree as a
sentence S2)
TYPE S2 .  (The system types the sentence in English; last two
commands are not necessary for the proof)
"IF X AND Y ARE JOINED BY A WALK IN G THEN X AND Y ARE
JOINED BY A WALK IN COMPLEMENT OF G" .
SET S3 = "THERE EXISTS Z SUCH THAT [X AND Z ARE NOT
JOINED BY A WALK IN G AND Y AND Z ARE NOT JOINED BY A WALK
IN G]" .
MODIFY T BY CASE S3 .  (Subgoal 5 replaced by subgoals 7 and 8)
MOVE ROOT OF T DOWN .  (Root moved to subgoal 7)
EXTEND T .  (Subgoal 9 produced)
MOVE ROOT OF T DOWN .  (Root moved to subgoal 9)
SET S4 = ROOT OF T .
PROVE S4 .
I proved sentence S4.
SET ROOT OF T TO BE TRUE .
Tell me why it is true.
PROVED BY RESOLUTION
----------------------------------------------------------------
Next few commands are not necessary for the proof.
----------------------------------------------------------------
MOVE ROOT OF T UP .  (Root moved to subgoal 7)
MOVE ROOT OF T TO THE RIGHT .  (Root moved to subgoal 8)
TYPE ROOT OF T .
(EXT Z)(NOT.R4G(X,Z).AND.NOT.R4G(Y,Z))=>R4G$A1(X,Y)
----------------------------------------------------------------
        After some additional dialog, not reproduced here, the
proof is completed. By the next command the proof tree is typed.
----------------------------------------------------------------
```

```
TYPE T .
 1. NOT.P12G=>P12G$A1
   2. NOT.P12G=>(ALL X)(ALL Y)R4G$A1(X,Y)
     3. (ALL X)(ALL Y)R4G$A1(X,Y)
       4. R4G$A1(X,Y)
         5. R4G(X,Y)=>R4G$A1(X,Y)
           7. R4G(X,Y)=>(EXT Z)(NOT.R4G(X,Z).AND.NOT.R4G(Y,Z))
             9. NOT.P12G.AND.R4G(X,Y)=>(EXT Z)(NOT.R4G(X,Z).AND.NOT.R4G(
                Y,Z))  PROVED BY RESOLUTION
           8. (EXT Z)(NOT.R4G(X,Z).AND.NOT.R4G(Y,Z))=>R4G$A1(X,Y)
            10. (EXT Z)(NOT.R4G(X,Z).AND.NOT.R4G(Y,Z))=>(EXT Z1)(R4G$A1(
                X,Z1).AND.R4G$A1(Y,Z1))  TRUE IF SUBGOAL 6 IS TRUE
            11. (EXT Z1)(R4G$A1(X,Z1).AND.R4G$A1(Y,Z1))=>R4G$A1(X,Y)
              12. (EXT Z1)(R4G$A1(X,Z1).AND.R4G$A1(Z1,Y))=>R4G$A1(X,Y)
                13. (EXT Z1)R4G$A1(X,Y)=>R4G$A1(X,Y)
                  14. R4G$A1(X,Y)=>R4G$A1(X,Y)  TRUE
         6. NOT.R4G(X,Y)=>R4G$A1(X,Y)
          15. NOT.R4G(X,Y)=>NOT.R1G(X,Y)
            17. NOT.(EXT K)S2G(X,Y,K)=>NOT.R1G(X,Y)
              18. NOT.(EXT K)S2G(X,Y,K)=>NOT.S2G(X,Y,1)
                20. S2G(X,Y,1)=>(EXT K)(S2G(X,Y,K))  TRUE
              19. NOT.S2G(X,Y,1)=>NOT.R1G(X,Y)
                21. R1G(X,Y)=>S2G(X,Y,1)
                  22. R1G(X,Y)=>R1G(X,Y)  TRUE
          16. NOT.R1G(X,Y)=>R4G$A1(X,Y)
            23. X=Y=>(NOT.R1G(X,Y)=>R4G$A1(X,Y))
              25. X=Y.AND.NOT.R1G(X,Y)=>R4G$A1(X,Y)  TRUE
            24. X#Y=>(NOT.R1G(X,Y)=>R4G$A1(X,Y))
              26. X#Y.AND.NOT.R1G(X,Y)=>R4G$A1(X,Y)
                27. R1G$A1(X,Y)=>R4G$A1(X,Y)
                  28. R1G$A1(X,Y)=>(EXT K)S2G$A1(X,Y,K)
                    29. R1G$A1(X,Y)=>S2G$A1(X,Y,1)
                      31. R1G$A1(X,Y)=>R1G$A1(X,Y)  TRUE
                    30. S2G$A1(X,Y,1)=>(EXT K)S2G$A1(X,Y,K)  TRUE
```

Points of the tree are represented by the corresponding subgoals, i.e. by some formulas. First formula represents the original sentence S. “Sons” are printed two characters to the right and below their “father”. The system can also print a proof tree in such a way that formulas are replaced by the corresponding sentences in English.

REFERENCES

[1] W. W. Bledsoe, Bruell, *A man-machine theorem-proving system*, Artificial Intelligence **5** (1974), 51–72.

[2] W. W. Bledsoe, *Spliting and reduction heuristics in automatic theorem proving*, Artificial Intelligence **2** (1971), 55–77.

[3] W. W. Bledsoe, *Non resolution theorem proving*, Artifical Intelligence **9** (1977), 1–35.

[4] D. M. Cvetković *A project for using computers in further development of graph theory The Theory and Applications of Graphs*, Proc. 4. th Internet. Conf. Theory Appl. of Graphs, Kalamazoo, 1980, Wiley 1981, 285– 296.

[5] D. Cvetković, *Translating mathematical text into the language of formulas of first order calculus by means of a computer*, (Serbo-Croatian, English summary), Proc. 15th Yugoslav International Symposium of Computer Technology and Problems of Informatics, Ljubljana, 1981, 3, 108, 1–3.

[6] D. Cvetković, I. Pevac, *Generating sentences equivalent to a given sentence*, (Serbo-Croatian, English summary), Proc. 15th Yugoslav Internat. Symposium of Computer Technology and Problems of Informatics, Ljubljana, 1981, 3, 107, 1–3.

[7] D. M. Cvetković, M. Ćipranić, *Interactive programming system for manipulation with sentences*, (Serbo-Croatian, English summary), Proc. of the XXVI Yugoslav ETAN Conf. Subotica 1982., Vol III, 263–269.

[8] D. M. Cvetković, L. L. Kraus, S. K. Simić, *Discussing graph theory with a computer I, Implementation of graph theoretic algorithms*, Univ. Beograd, Publ. Elektrotehn. Fak., Ser. Mat. Fiz. N^o **716**–N^o**734** (1981), 100-104.

[9] D. M. Cvetković, *Discussing graph theory with a computer II, Theorems suggested by the computer*, Publ. Inst. Math. (Beograd) **33** (47) (1983), 29–34.

[10] R. W. Erickson, D. R. Musser, *The affirm theorem prover: proof forests and management of large proofs*, Proc. 5th Conference on Automated Deduction, Les Arcs 1980, 220–231.

[11] P. Hotomski, *A way of building in the induction rule into procedures of automatic theorem proving by resolution* (in Russian), Publ. Inst. Math. (Beograd) **33**(47), (1983) 89–96.

[12] V. Stojković, D. Cvetković, *Parsing sentences of a subset of English formalized for the use in graph theory*(Serbo-Croatian, English summary), Proc. XXV Yigoslav ETAN Conf., Mostar 1981, vol. III, 271–278.

Elektrotehnički fakultet
Beograd

Matematički institut
Knez Mihailova 35
Beograd

(Received 10 09 1982)