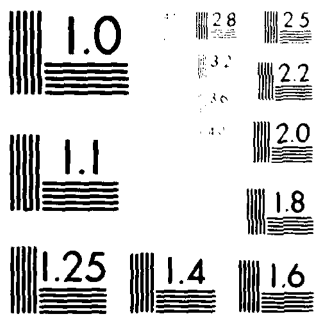
1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

FTD-ID(RS)T-0626-80

(4)

FOREIGN TECHNOLOGY DIVISION

PROGRAMMING AND PROVING CORRECTNESS OF THE SCANNING ALGORITHM WITH BACKTRACK

by

Juraj Wiedermann

FTD-ID(RS)T-0626-80

EDITED TRANSLATION

14 FTD-ID(RS)T-0626-80 11 19 Sep~~tember 1~~980

MICROFICHE NR: FTD-80-C-001029

6 PROGRAMMING AND PROVING CORRECTNESS OF THE SCANNING ALGORITHM WITH BACKTRACK

By: Juraj Wiedermann 10

~~English pages: 12~~

21 Edited trans. 31

~~Source~~: Informacne Systemy, Nr. 4, 1977, (Yugoslavia) n4 p371-382 1977, pp. 371-382.

Country of origin: Yugoslavia
Translated by: SCITRAN
F33657-78-D-0619
Requester: FTD/TQT
Approved for public release; distribution unlimited.

12 14

Accession For	
NTIS GRA&I	☒
DDC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist.	Avail and/or special
A	

141600 [illegible]

PROGRAMMING AND PROVING CORRECTNESS OF THE SCANNING ALGORITHM WITH BACKTRACK

Juraj Wiedermann
Computer Research Center, Bratislava

Abstract: The systematic construction of a program in which its correctness is demonstrated in parallel to the development of the program is explained in the example of the scanning algorithm with backtrack. The total correctness of the program is proven by the method of intermittent assertions.

Key words: Programming; Program correctness; Systematic construction of programs; Scanning algorithm with backtrack.

1. Introduction

When solving combinatorial problems in practice, we frequently encounter a situation in which we have a given (finite) set A of candidates to resolve the problem and we must choose from this set those candidates that satisfy the specific conditions p for the solution. This means that the solution to our problem is the set $B = \{x \in A \mid p(x)\}$.

We will call the algorithms that solve this problem the scanning algorithms since to find all the solutions, they must scan the entire set of candidates (on the assumption that we do not know any of the relationships between the elements in set A).

Later on, we will look at a simple program flow chart which is the solution to a given problem and we will prove its correctness.

The scanning problem as we have defined it in the introduction is quite universal in many situations. Since we are limited here to a special problem where we need not scan the set of candidates by the exhaustive technique of "one after the other" and can sometimes "omit" certain elements because we know that they cannot be the solution to the problem. We will call these algorithms scanning algorithms with backtrack and using them, we will look at a universal programming chart and prove its total correctness.

We will prove total correctness with the method called intermittent assertions [6].

Since the scanning algorithm with backtrack is a special instance of the universal scanning algorithm, we will see that this function is also applicable to proving the correctness of these algorithms; proving the correctness of the scanning algorithm with backtrack is a special "refined" proof of the correctness of the universal scanning algorithm.

With this, we also hope to show at the same time that the method of intermittent assertions is applicable for the so-called systematic construction of programs where its correctness is proven in parallel to the development of the program.

In the conclusion we will look at the problem of "27 cards" as an application of the universal scheme of the scanning algorithm in a concrete example.

2. The universal scanning algorithm

The following program which solves the universal scanning problem is simple and obvious:

```
                program SEARCH;
                begin
                        input (H);
start:                  B = ∅;
loop:                   while H ≠ ∅ do begin
                                x ← H; H: = H − {x};
                                if p(x) then B: = B ∪ {x} fi
                        end;
finish:                 output (B)
                end
```

The instruction "x ← H" means "select an arbitrary element from the set H and assign the variables of x to it".

At the beginning, this program "opens" the set of candidates into sets (variables) of H and so long as the set H is not an empty set, it selects one element after another and tests whether this element is or is not the solution; if not, it then moves on to the set B.

The following theorem explains the total correctness of the SEARCH program:

THEOREM 1: If at times H = A at the start signal, then sometimes $B = \{x \in A \mid p(x)\}$ at the finish signal.

We will note that this theorem does, in fact, explain the total correctness of the SEARCH program; that is, when the entry condition at the beginning of the program is such that the program ends (the theorem implies that the solution to the program is sometimes achieved at the finish signal which is at the end of the program) and, consequently, the output condition is fulfilled at the close.

In order for us to be able to use the method of intermittent assertions, we must generally look through each cycle of the program at the validity of any lemmas that describe the effect of that cycle and imply (generally in combination with other lemmas) the validity of the main theorem which explains the total correctness of the program.

The behavior of a single cycle in our program is described by the following lemma:

LEMMA 1. If when $H = H'$ and $B = B'$ at the start, then when $H = \emptyset$ and $B = B' \cup \{x \in H' \mid p(x)\}$ at the start loop.

The assumption $H = H'$ and $B = B'$ are lemmas that enable us to look for sequential values in the sets H and B at the conclusion of the lemma where the values of H and B can change in the interim.

Stated informally, Lemma 1 asserts that the cycle with the start loop is sometimes the end — the original value H' of the set H sometimes changes to the value $H = \emptyset$ — and consequently, the set $\{x \in H' \mid p(x)\}$ increases towards the original value B' in the set B; that is, all the solutions from this set (H') that we have not "disproven".

Lemma 1 is proven with full inductive regard for the value of the set H':

Let $H= H'$ and $B = B'$ at times in the start loop;

1. Let $H' = \emptyset$. Then, it holds true that $H = \emptyset$ and $B = B' \cup \{x \in \emptyset \mid p(x)\} =$ $= B'$ at the start loop which must be proven.
2. We will assume that the lemma is correct for all values in the set $F \subset H'$.
3. Since $H = H' \neq \emptyset$ holds after one cycle pass , there is also an $x \in H'$ such that $H = H' - \{x\}$ and $B = B' \cup \{y \in H' - \{x\} \mid p(y)\}$ in the start loop. Therefore, $H' - \{x\} \subset H'$ and we can use the inductive assumption so that when $H = \emptyset$ and $B = B' \cup \{y \in \{x\} \mid p(y)\} \cup$ $\cup \{y \in H' - \{x\} \mid p(y)\} = B' \cup \{y \in H' \mid p(y)\}$ in the start loop which proves the lemma.

It is essentially true that a program as simple as the SEARCH program might never be proven in practice. We make special note of this in order to show in a simple program how total correctness of a program is proven by the method of intermittent assertions as since later on in proving the correctness of the scanning algorithm with backtrack, we can use some of the concepts and experience acquired by proving the correctness of the universal scanning algorithm.

2. The scanning algorithm with backtrack

On the assumption that all elements in the set of candidates A are explicitly given (we are not aware of the order or how they are generated), the preceding program for universal scanning essentially has no effect whatsoever. Its memory and temporal make-up are unsuitable — we must retain the entire set A in memory at one time and hold on to all its elements. If the set A is large (as would be the case in combinatorial problems) the entire program is intolerably inefficient.

It is often the case, however, that the set A is not explicitly given; that is, that not all its elements are generated one after the other or that we know the order of how all the elements are generated.

We will assume a set of $A = A_1 \times A_2 \times \ldots \times A_n$; the solution to the scanning problem then is all the n-tuples (vectors) of $\bar{x}=(x_1, x_2, \ldots x_n) \in A$ so that $p(\bar{x})$ is valid.

Then we are able to program the scanning algorithm in an essentially more effect manner (at least, for the present, from the memory standpoint); we need not remember the entire set A all at once but only the individual components $A_1, A_2, \ldots A_n$ of the Cartesian product and that all elements of the set A are generated sequentially from them. It is simplest to program how n is tested against itself in insert cycles where a single component of the vector $\bar{x}$ is generated in each cycle as well as during the innermost cycle where the entire vector $\bar{x}$ is generated which is the solution:

```
        program SIMPLEBACKTRACK;
        begin
                input ($X_1, \ldots X_n$);
start:          $H_1 := X_1$; $B := \emptyset$;
                while $H_1 \neq \emptyset$ do begin
                    $x_1 \leftarrow H_1$; $H_1 := H_1 - \{x_1\}$;
                    $H_2 := X_2$;
                    while $H_2 \neq \emptyset$ do begin
                        $x_2 \leftarrow H_2$; $H_2 := H_2 - \{x_2\}$;
                        ⋮
                        $H_n := X_n$;
                        while $H_n \neq \emptyset$ do begin
                            $x_n \leftarrow H_n$; $H_n := H_n - \{x_n\}$;
                            if $p(x_1, x_2, \ldots x_n)$
                              then $B := B \cup \{x_1, \ldots x_n\}$
                            fi
                        end
                        ⋮
                    end
                end;
finish:         output ($B$)
        end;
```

Thus when $X_1 = A_1$, $X_2 = A_2, \ldots, X_n = A_n$ at the start signal, then it can be proven that it will be true that $B = \{\bar{x} \in A \mid p(\bar{x})\}$ at the finish signal.

The correctness of this progam cannot be proven, however, since this program is still unattractive. Aside from the fact that we must know the concrete value of n in advance in order to make up a program at all, it is also inefficient from the time standpoint as is the preceding SEARCH program for universal scanning.

The reason for this time inefficiency is that in the i-th cycle,

$1 \leq i \leq n$ generates further components x_i of the vector $\bar{x}$ without convincing us that the currently discovered components $x_1, \ldots x_{i-1}$ can be supplemented in any solution at all.

Let us name the predicates $p(x_1, \ldots x_i)$ such that $p(x_1, \ldots x_n) \supset p(x_1, \ldots x_i)$ where $i = 1, \ldots, n-1$ and satisfaction of the predicate $p(x_1, \ldots x_i)$ is a necessary condition for the predicate $p(\bar{x})$ to be fulfilled at all.

What we arrange in this sense is the preceding program and we "roll" the cycle back to itself and we get the following program:

```
            program BACKTRACK;
            begin
                 input $(X_1, \ldots X_n)$;
start:           $H_1 := X_1$; $B :\neq \emptyset$; $i := 1$;
L1:              while $i \leq 1$ do begin
L2:                   while $H_2 \neq \emptyset$ do begin
                           $x_i \leftarrow H_i$; $H_i := H_i - \{x_i\}$;
                           if $p(x_1, \ldots x_i)$
                              then if $i = n$
                                      then $B := B \cup \{\bar{x}\}$
                                      else $i := i + 1$; $H_i := X_i$
                                   fi
                           fi
                      end;
                      $i := i - 1$
                 end;
finish:          output $(B)$
            end
```

From this program, we can see that after generating one component of x_i, we can predict all that fullfill the necessary condition of the solution — the predicate $p(x_1, \ldots x_i)$ and only when it has been fulfilled (and not just another solution) do we "plunge" into a deeper cycle; otherwise, it is apparent that no choice of $x_{i+1}, \ldots x_n$ can fulfill the condition $p(\bar{x})$ — this means that all n-th which begin $x_1, \ldots x_i$ can be omitted from the scanning process — which we get so that we proceed with selection of another $x_i \in H_i$.

If we exhaust all the elements at the i-th level, then we select again at the i-1 level and continue on through the generation of the components x_{i-1} at this level. From there, we come back to the algorithm — the scanning algorithm with backtrack.

We prove the total correctness of the BACKTRACK program as we prove the validity of the following theorem:

THEOREM 2: If $X = A$ at the start signal, then $B = \{\bar{x} \in A \mid p(\bar{x})\}$ at the finish signal.

We can prove Theorem 2 by means of the following lemma:

LEMMA 2. When $H_i = H_i'$, $B = B'$ and $1 \leq i = i' \leq n$ at the L2 signal then $H_i = \emptyset$, $B = B' \cup \{\bar{y} \in \{x_1\} \times \ldots \times \{x_{i-1}\} \times H_i' \times X_{i+1} \times \ldots \times X_n \mid p(\bar{y})\}$ and $i = i'$ at the L2 loop.

When the assumption $X = A$ of Theorem 2 is fulfilled at the start signal, we also get $H_1 = A_1$, $B = \emptyset$ and $i = 1$ at the L2 signal so that according to Lemma 2, when $H_1 = \emptyset$, $B = \{\bar{y} \in A \mid p(\bar{y})\}$ and $i = 1$ at the L2 signal. Since $H_i = \emptyset$, we can drop the cycle with the L2 loop, the value of i will drop by one and we can proceed to the L1 signal with a value of $i = 0$ and, from there, on to the finish signal at which the output condition of $B = \{\bar{y} \in A \mid p(\bar{y})\}$ which is the assertion of Theorem 2 remains valid.

We can see here the similarity between Theorem 1, Lemma 1 and Theorem 2 and Lemma 2. Theorem 2 and Lemma 2 merely reflect the "more refined" structure of the set $A = A_1 \times \ldots \times A_n$.

We can offer a proof of Lemma 2 with an inductive view of i'.

1. When $i = n$, let $H_n = H_n'$ and $B = B'$ at the L2 loop. Looking inductively at the magnitude of the set H_n', we can show the validity of the lemma in the following case:

1.1 If $H_n' = \emptyset$, the fact that the assertion of the lemma is fulfilled at the L2 signal is trivial.

1.2. We will assume that the lemma is valid for all subsets $F_n \subset H_n'$.

1.3. If $H_n = H_n' \neq \emptyset$, then it is true after one cycle pass at the n-th level that:
there exists an $x_n \in H_n'$ such that $H_n = H_n' - \{x_n\}$, $B = B' \cup \{\bar{y} \in \{x_1\} \times \ldots \times \{x_n\} \mid p(\bar{y})\} = B''$. Since $H_n' - \{x_n\} \subset H_n'$, we can use the inductive assumption 1.2 so that when $H_n = \emptyset$ and $B = B'' \cup \{\bar{y} \in \{x_1\} \times \ldots \times \{x_{n-1}\} \times (H_n' - \{x_n\}) \mid p(\bar{y})\} = B' \cup \{\bar{y} \in \{x_1\} \times \ldots \times \{x_{n-1}\} \times H_n' \mid p(\bar{y})\}$ and $i = n$ at the L2 signal;
Q.E.D.

2. We will assume that the lemma holds true for all j, $1 \leq i < j \leq n$.

3. Looking inductively at the magnitude of the set H_i', we can show the validity of the lemma for when $i = i'$ also.

Therefore, let $H_i = H_i'$, $B = B'$ and $1 \leq i = i' < n$ at the L2 signal.

3.1. If $H_i' = \emptyset$, the lemma is, of course, fulfilled.

3.2. We will assume that the lemma holds true for all subsets $F_i \subset H_i'$.

3.3. When $H_i = H_i' \neq \emptyset$, then after the first two instructions in the cycle it holds true that there exists an $x_i \in H_i'$ so that $H_i = H_i' - \{x_i\}$.

We can further break it down into two cases:

a) if it is not true that $p(x_1, \dots x_i)$, we can go back to the L2 signal with a value of $i = i'$, $H = H_i' - \{x_i\}$ and $B = B' \cup \{\bar{y} \in \{x_1\} \times \dots \times \{x_i\} \times X_{i+1} \times \dots X_n \mid p(\bar{y})\} = B'' = B' \cup C$. From the assumption that $p(x_1, \dots x_i)$ is not true, it follows that no n-tuples beginning at $x_1, \dots, x_i$ can be the solution; this means that the set $C = \emptyset$ and that we can use it without everything to add to the set B'. Since $H_i' - \{x_i\} \subset H_i'$, we can use the inductive assumption 3.2 so that $H_i = \emptyset$ and $B = B'' \cup \{\bar{y} \in \{x_1\} \times \dots \times \{x_{i-1}\} \times (H_i' - \{x_i\}) \times X_{i+1} \times \dots X_n \mid p(\bar{y})\}$ at the L2 signal and also $i = i'$ at the L2 signal.

We can get a confirmation of the lemma by applying the preceding expression to the set B.

b) If $p(x_1, \dots x_i)$ holds true and since $i < n$, the value of i increases by one for a value of $j = i + 1$, we also reach the L2 signal with values of $H_j = X_j$, $B = B'$ and $1 < j = i + 1 \leq n$. Therefore, $1 \leq i < j \leq n$ according to inductive assumption 2 when $H_j = \emptyset$, $B = B' \cup \{\bar{y} \in \{x_1\} \times \dots \times \{x_{j-1}\} \times X_j \times \dots \times X_n \mid p(\mathbf{y})\}$ and $j = i + 1$ at L2. Since $H_j = \emptyset$ at L2, the value of j is reduced to the original value i and since $i \geqslant 1$, we move from the L1 signal to the L2 signal with /the values/ $i = i'$, $H_i = H_i' - \{x_i\}$ and $B = B''$. Since $H_i' - \{x_i\} \subset H_i'$, we get a confirmation of the lemma according to the inductive assumption 3.2 similar to that in the preceding situation a).

By proving Lemma 2, we have completed the proof of the total correctness of the BACKTRACK program.

On its own level, this program represents an effective flow plan for the scanning algorithm with backtrack. This plan also provides for a relatively direct "approach" to solving a concrete problem. Therefore, it is necessary only to find (and program) the correct interpretation of the set A and the predicate p. The effectiveness of the resultant program is critically dependent on the effectiveness of running a test of **if** $p(x_1, \dots x_i)$ **then** ....

It is apparanent that the sooner we recognize that some partial solution of $x_1, \dots x_i$ cannot be expanded (completed) to a full solution, the more elements of the set A we can eliminate all at once from the scanning process and the faster we will achieve the goal.

From the nature of the combinatorial problem to be solved, we can sometimes see that a simple transformation from one solution to the problem can result in some other solutions (the solutions are isomorphous).

For this reason, it is advantageous not to generate these other isomorphic solutions and to "skip over" them. The test for this can also be hidden in the predicate p. Further detailing of this problem as well as the questions of effective implementation of scanning algorithms with backtrack is the subject of reference /1/.

4. The problem of 27 cards

The problem of 27 cards is as follows /2/: we have a playing board with 27 fields and 3 sets of cards with values of from 1 through 9. Our job is to find all the possible positions of all the cards on the game board so that between every two adjacent fields on the game board on which there are cards of some value k, there are cards to the right of the field with some other value.

The exercise may obviously be generalized to a problem of 3n cards.

Let x_i stand for the position of the first card of three with values of i, i = 1 ... n on the game board. From the statement of the problem it follows that the position of the first card also determines the position of the other two cards — they must be in positions $x_i + i + 1$ and $x_i + 2i + 2$ and also that $1 \leq x_i \leq 3n - 2i - 2$ as otherwise, some card would fall outside the playing area. The problem of our sets A_i then is the playing intervals $\langle 1, 3n-2i-2 \rangle$ where i = 1, ...n.

The predicate $p(x_1, \dots x_i)$ is now interpreted as correct so long as it places cards with a value of k in the positions x_k, $x_k + k + 1$ and $x_k + 2k + 2$ given that k = 1,...i; otherwise it is incorrect. The solution to our problem will be all n's of $x_1, \dots x_n$ such that $p(x_1, \dots x_n)$ holds true.

The partially stated BACKTRACK program plan which has been adapted to solve the problem of 27 cards now looks as follows:

```
program CARDS;
begin input ($n$);
    $x_1 := 3n - 4$; $i := 1$;
    while $i \geq 1$ do begin
        while $x_i > 0$ do begin
            $x_i := x_i - 1$;
            if $p(x_1 + 1, \dots x_i + 1)$
              then if $i = n$
                    then print ($x_1, \dots x_n$)
                    else $i := i + 1$; $x_i := 3n - 2i - 2$
              fi
            fi
        end;
        $i := i - 1$
    end
end
```

We can see that we can do without the sets X_i since we are assured by the sequential reduction in the value of x_i (which has been initialized to the value of $3n-2i-2$ prior to each plunge into the cycle) by one that we will cover all the possible values that x_i can acquire.

Thus, we are not "collecting" all the solutions in set B so that we print them directly (the representative area — "listing" — plays the part of set B).

With the full "extension" of the CARDS program to some concrete programing language (such as Pascal) we can also program the predicate p. We can do this as a Boolean function, for example which has a game field and memory coded into itself with an occupied field. We will recall that the efficiency of the entire program to a great extent depends directly on the effective implementation of this predicate since the test of its accuracy is determined in each cycle. This problem is not solely concerned with the basic function of the scanning algorithm with backtrack and for this reason, we will not deal with it further.

5. Conclusion

In the foregoing text, we have shown how to prove the total correctness of programs by the intermittent assertions technique. While using this method, total correctness of a program is expressed in the form of a theorem:

THEOREM: When the opening condition is fulfilled at the point of entry into the program, the solution is achieved at the end of the program and at that point, the output condition is fulfilled.

This theorem states that the program is partially correct and, over and above this, that it ends.

In order to prove the validity of this theorem for our program, we must find an intermittent assertion at a specified point in the program which asserts something about the chosen variables in the program that also holds true when the solution to the problem passes through. The connection between these assertions can be described by means of lemmas from which proof of the validity of the main theorem must come.

In general, we will need one lemma which describes the intended behavior of the cycle for each cycle in the program. This lemma generally asserts that when the cycle ends, it adds what still remains to be completed to what has currently been executed in the cycle.

The proof of these lemmas is generally worked out through complete induction with the elements (variables) of a certain set which does not contain any indeterminate non-declining series (the so-called "wellfounded set" — see [5] or [4]).

The only, and therefore probably the greatest problem remains of how to formulate the assertion of the theorem so that it expresses the total correctness of the program and how to formulate the assertion of the lemma so that it is possible to prove the theorem from then and also so that we can prove the lemma independently.

This validates the "golden rule of proving programs", that is, the better we think out the program, the easier it is for us to prove its correctness. When we recognize the fact that the more intelligible a program is, the simpler and shorter it is, we come to the next instruction on how to proceed with proving programs:

We start with the suggestion of a program that solves a given problem on the level of abstraction that we understand well and then, we can move easily to formulate the assertion of the theorem and the secondary lemma and to prove them. In a further step, we drop to a lower level of abstraction so that we make certain instructions in the program more specific and we can refine its structure. Then we come to the program which differs only "slightly" from the original and therefore, the proof of its correctness would have a resemblance "in its coarser features" to the proof of the preceding program. This means that the "skeleton" of the new proof is already made up and the proof requires only the "refinement" of certain assertions and arguments so that it is appropriate to the new situation. Thus, we proceed along with the gradual refinement of the program and proving its correctness and we do not work towards a meaningful program which will solve a given problem and has already been formulated in a specific and meaningful programming language.

It seems that in practice it is sufficient to "attain" proof of the program's correctness when the program is at the "penultimate" level of abstraction; that is, it is not necessary to come up with a proof of correctness for the final program in a meaningful languange except when the program is at the highest abstract level which, to some extent, represents the most universal plan in the solution to our problem — and the one from which we can achieve our meaningful program more or less directly. During the final phase of firming up the program, only the final simple functions which add to the program additional secondary variables but which do not alter the control of the program are made more specific. At the programming level, the experienced programmer assumes this function and substantive errors introduced into the program during this phase are found either by a good collator (differences in types, parameters, indexes that "run past" the acceptable dimensions, and the like) or they are quickly removed by the programmer during the debugging process. The descriptive complexity of proof at this final level would increase considerably and be greatly augmented by the probability of introducing error into the proof as well.

This reality supports (in addition to our own experience) the empirical reconnaisance of programs [3] where it has been shown that, in a number of cases, programs have been incorrect in spite of the fact that their correctness has been proven. It was also shown, however, that error was introduced into the program right at the final stage of making the program concrete and that the proof "did not find" it but rather that the error per se was removed during the first failure of the program.

From the above considerations, it is particularly apparent that there is no need to prove programs in standard practice whose statement at a "sufficiently" high degree of abstraction is at the same time a concrete program in a meaningful language as well — the so-called "baby" programs (looking for the maximum element in a field, testing the equality of two fields, and the like, for example).

It is apparent that the method of constructing proof of correctness in parallel to the structure of a program is useful in most of the known methods for proving the correctness of programs. It also indicates [6] that proving the total correctness of programs is, in many cases, done most simply by the technique of intermittent assertions rather than by means of the "traditional" methods (Floyd's method of invariants and derived techniques — [4, 5]).

Our proof of the correctness of the BACKTRACK program can serve as an example of this if we compare it with the proof of the similar but recursive program in [7] which was shown by the invariants technique.

In [6] it is shown again and again that proof by the technique of intermittent assertions can never be as "cumbersone" as proof by means of invariants since the invariants technique is only a special instance of the intermittent assertions technique.

Acknowledgements. We would like to thank Dr. P. Ruzick and Dr. P. Vodov who, through their interest in the problems of this article, contributed immensely to getting it into its present form.

BIBLIOGRAPHY

1. Bitner, J., Reingold, E.: Backtrack programming Techniques. Comm. ACM, 18, 1975, Nr 2, pp 651-655.

2. Gardner, M.: Mathematical Games. Some new and dramatic demonstrations of number theorems with playing cards. Scientific American, Vol. 231, 1974, No 5, pp 122-125.

3. Gerhart, S., Yelowitz, L.: Observations of fallibility in applications of modern programming methodologies. IEEE Transactions on Software Engineering, Vol. SE-2, 1976, No 3, pp 195 - 207.

4. Gruska, J., Privara, I.: "Dokazovanie spravnosti programov" /Proving the Correctness of Programs/ In: Zbornik seminara "SOFSEM 76" /Anthology from the "SOFSEM-76" Seminar/, Computer Research Center, Bratislava, 1976, pp 331-375.

5. Manna, Z.: Mathematical theory of computations (Computer Science Series), McGraw-Hill, New York, 1974.

6. Manna, Z., Waldinger, R.: Is "sometime" sometimes better than "always"? Intermittent assertions in proving program correctness. Report No. STAN-CS-76-558, Stanford University, June 1976

7. Privara, I.: "Teoria a aplikacia formalnej semantiky programovacich jazykov" /The Theory and Application of Formal Semantics in Programming Languages/ (Vyskumna sprava "Efektivnost' algoritmov programovacich systemov 1.") /Research report, "The Effectiveness of Algorithms in Programming Systems, 1"/. Computer Research Center, Bratislava, 1976, pp 75 - 117.

PROGRAMMING AND PROVING CORRECTNESS OF THE BACKTRACK ALGORITHM

Solving the combinatorial problems in practice we often find ourselves in a situation where we have a certain (final) set of candidates for the problem solution and we have to choose from this set those candidates that fulfil a certain condition of the solution. Algorithms solving this problem are called the scanning algorithms, because for finding all the solutions they have to scan the whole set of candidates. The example of the efficient scanning algorithm — backtrack algorithm — shows the systematic construction of program where, together with the development of the program, its correctness is proved. The total correctness of the program is proved by the method of intermittent assertions.