KYBERNETIKA — VOLUME 15 (1979), NUMBER 2

Intermittent-Assertion Method as a Structural Induction

KAREL VOSÁTKA

This paper formulates intermittent-assertion method for program verification as a structural induction. It analyses the inductive mechanism of the proof. The relation of the method to other known methods for program verification is studied. The paper also formulates verification conditions which represent the proof intermittent-assertion method.

1. INTRODUCTION

The intermittent-assertion method (IAM) is a method for proving total correctness of programs. It uses assertions, which are placed at some points in program structure. As opposed to invariants, they need not hold always at that point but only sometimes. This technique was suggested by Burstall [1] and later it was described by Manna and Waldinger [2]. In both of them ([1] and [2]) the IAM was not described formally. It was only illustrated with a number of examples. In the paper [2] the authors also showed that a correctness proof using the invariant assertion method (Floyd [3]) or the subgoal induction method (Morris and Wegbreit [4]) can be expressed by using IAM. They noted that the reverse is not always possible. This paper should aid better understanding of connections between these methods.

We shall show easily that IAM can be expressed by means of structural induction of Burstall [5]. Structural induction is used to prove properties of data structures. We shall distinguish two types of proofs according to induction on the domain of the program:

In the first type of proof, the inductive structure is derived from the structure of program itself.

In the second type of proof, the inductive structure is derived from the specifications of the program. The natural structure of the domain or the structure of expected function of program can be used.

We shall show that the first type can be expressed by using subgoal induction method and then by using invariant method, too. What the IAM contributes to is just the second type of proofs.

It is advantageous for the reader to know the paper [2], but it is not requirement.

2. INTERMITTENT-ASSERTION METHOD

Let P be a program, $J(x_0)$ be an input specification and $K(x_0, x)$ be an output specification. x is a program variable (or vector of variables) and x_0 is its input value. We are to prove the total correctness of program P with respect to J and K.

We shall place cutpoints to the program P : at the entrance and at each exit of P and at least one inside of each loop.

Intermittent assertion at some cutpoint A is written in the form:

$$\textbf{\textit{Sometime}}\ (\text{At } A \text{ and } T(x))$$

and it is read:

"Sometime the computation is at the cutpoint A and $T(x)$ holds."

If we denote the cutpoint at the entrance of P by A and the cutpoint at the exit by B (for simplicity we suppose only one exit of P) we shall express the total correctnes of P as a theorem:

Theorem.

$$\text{If } \textbf{\textit{Sometime}}\ (\text{At } A \text{ and } J(x_0)) \qquad (2.1)$$

$$\text{then } \textbf{\textit{Sometime}}\ (\text{At } B \text{ and } K(x_0, x))\,.$$

Note that the theorem involves also the request for termination of P, because it says that the computation will occur sometime at the end of program.

The theorem is proved with the help of lemmas which are formulated for each loop and which describe intended behaviour of the loops.

Let C be a cutpoint of a loop, then

Lemma.

$$\text{If } \textbf{\textit{Sometime}}\ (\text{At } C \text{ and } a = x \text{ and } Q1(a)) \qquad (2.2)$$

$$\text{then } \textbf{\textit{Sometime}}\ (\text{At } C \text{ and } Q2(a, x))\,.$$

expresses a relation between two values of variable x at two states of computation at C. It is possible to prove the theorem with the help of relation between the values of x at the first and the last passage through C.

The proof of the lemma involves complete induction over a well-founded set. Thus the essential condition for the proof of the theorem is a good choice of a lemma and of a well-founded set for each loop. We shall show that this choice determines the types of proof as was told in the introduction.

3. STRUCTURAL INDUCTION

Let $(S, \prec)$ be a partially ordered set and let it be *well-founded*; which means that it contains no infinite decreasing sequence $a_0 \succ a_1 \succ a_2 \succ \dots$ of its elements. Thus each subset S' of S contains a minimal element (we call it *$\prec$-least element of S'*).

Let φ be a total predicate over S. Then φ is *inductive on* S iff the following condition holds:

$$(\forall a \in S)\,\{[(\forall b \in S \text{ such that } b \prec a)\,\varphi(b)] \supset \varphi(a)\}\,.$$

Then by *structural induction* (see Manna [6]):

If φ is inductive on S then $(\forall c \in S)\,\varphi(c)$.

If we are to prove some property Φ of elements of S it will do to verify if Φ is inductive on S. But if it is not the case, then we must find a predicate φ that is inductive on S and show that

$$(\forall a \in S)\,(\exists c \in S)\,(\varphi(c) \supset \Phi(a))$$

holds (we shall say that φ *is sufficient for Φ on S*).

Now let us return to the IAM.

We shall consider the following simple one-loop abstract program P:

(3.1)
```
A: input (x_0) ;
   x := f(x_0) ;
C: while t(x) do x := g(x) ;
   x := h(x) ;
B: output (x) .
```

If $P(x)$ is a function of P, we shall express the Theorem (2.1) as a property $V(x_0)$ of the input value x_0 of the variable x:

$$V(x_0) \equiv [J(x_0) \supset P(x_0) \text{ is defined} \wedge K(x_0, P(x_0))]\,. \tag{3.2}$$

Similarly we shall express the Lemma (2.2) as a property of value x at C:

$$\varphi(x) \equiv [Q1(x) \supset (\exists n)\, Q2(x, g^n(x))]\,, \tag{3.3}$$

where g is a function of one passage through the loop and g^n means n-th power of g.

We shall call the range of values of x at the cutpoint of the loop as a *domain of the loop*. If we find a well-founded set on the domain D of the loop and a predicate φ of the form (3.3) and if we verify that φ is inductive on D and that it is sufficient for V on the domain of P, then the total correctness of P with respect to J and K is proved.

We can verify the latter by proving:

$$(3.4) \qquad (\forall x_0)\,[\varphi(f(x_0)) \supset V(x_0)]\,.$$

To verify the former, we need an inductive structure of the domain D of the loop so that the program decomposes D according to that structure. A well-founded set will define such structure.

More formally, perform a decomposition of the set D (*well-founded decomposition*) into subsets D_p according to a well-founded set $(S, \prec_s)$, where

$$D_p = \{x : x \in D \wedge r(x) = p\}$$

and $p \in S$ and $r : D \to S$ is a representation.

We can define an ordering of D (*well-founded ordering*):

$$b \prec a \quad \text{iff} \quad b \in D_p\,, \quad a \in D_q \quad \text{and} \quad p, q \in S \quad \text{and} \quad p \prec_s q\,.$$

Thus the set $(D, \prec)$ will be well-founded and we are to verify:

$$(3.5) \qquad (\forall a \in D)\,\{[(\forall b \in D \text{ such that } b \prec a)\,\varphi(b)] \supset \varphi(a)\}\,.$$

4. THE FIRST TYPE OF PROOF

The inductive structure of the domain is derived from the structure of the program itself. See again the program (3.1). A well-founded ordering of D is looked for so that $g(x) \prec x$ can hold for each $x \in D$. If there is such well-founded ordering, the computation will terminate, because there is no infinite decreasing sequence $x \succ g(x) \succ \succ g^2(x) \succ \ldots$ We can compare this approach with the proof of termination by Floyd [2]:

The computation will terminate, if for a well-founded set $(S, \prec_s)$ and an expression $E(x)$, the following holds:

$$(\forall x \in D)\,[E(x) \in S \wedge (t(x) \supset E(g(x)) \prec_s E(x))]\,.$$

It offers directly a well-founded decomposition of D into the following subsets:

$$D_p = \{x : x \in D \wedge E(x) = p\}\,.$$

We can see that $\prec$-least elements of D are just those for which $\neg\, t(x)$ holds.

The following conditions are clearly sufficient for verification that φ is inductive on D:

(4.1) $$(\forall x \in D)(\neg t(x) \supset \varphi(x)),$$

(4.2) $$(\forall x \in D)(t(x) \supset g(x) \prec x),$$

(4.3) $$(\forall x \in D)(t(x) \wedge \varphi(g(x)) \supset \varphi(x)).$$

The condition (4.2) is stated for justification of that the $\prec$ ordering was well chosen.

We have four verification conditions for the total correctness of P now: (3.4), (4.1), (4.2) and (4.3).

As examples of such proof, see the proofs of the programs for finding a zero of an array or for the greatest common divisor of two numbers in [2]. The latter proof will be analysed in detail in the seventh section (Example 1).

5. FORMAL EQUIVALENCE TO OTHER TECHNIQUES

We shall show that the first type of the proof IAM (of course only with respect to the proof of partial correctness) is formally equivalent to subgoal induction method and thus also to invariant method.

Suppose that the computation will always terminate. Then the well-founded decomposition of D is easily found:

$$D_i = \{x : x \in D \text{ for which the computation terminates after } i \text{ iterations}\}.$$

The decomposition is defined according to well-founded set $(N, <)$: natural numbers with "less than" ordering. Of course, $g(x) \prec x$ always holds and we can leave out the (4.2) verification condition.

Subgoal induction is structural induction on the cartesian product $D \times H$ (see Reynolds and Yeh [7]), where D is a domain of the loop and H is a range of output values of the loop. Well-founded ordering of $D \times H$ is defined:

$$\langle x_1, y_1 \rangle \prec\!\cdot \langle x_2, y_2 \rangle \quad \text{iff} \quad x_1 \in D_i, \quad x_2 \in D_j, \quad y_1, y_2 \in H \quad \text{and} \quad i < j,$$

where D_i, D_j was defined above. Output values y_1, y_2 are $g^i(x_1)$, $g^j(x_2)$, respectively, as it directly follows.

Let ψ be a total (subgoal) predicate over $D \times H$, ψ will be inductive on $(D \times H, \prec\!\cdot)$ if

(5.1) $$(\forall x \in D)(\neg t(x) \supset \psi(x, x)),$$

(5.2) $$(\forall x \in D)(t(x) \wedge \psi(g(x), y) \supset \psi(x, y))$$

hold.

If the predicate φ of the form (3.3) is inductive on $(D, \prec)$, then the following predicate ψ will be inductive on $(D \times H, \prec.)$:

$$\psi(x, y) \equiv (\exists n)(\exists k)\,[(Q1(x) \supset Q2(x, g^n(x)) \wedge \neg\, t(g^{n+k}(x))) \wedge y = g^{n+k}(x)]\,.$$

The conjuct $(\exists k)\ \neg\, t(g^{n+k}(x))$ means that there is a last value of x at C so that the output condition of the loop holds. Then the validity of (5.1) and (5.2) follows from (4.1) and (4.3).

On the contrary, if ψ is inductive then $\varphi(x) \equiv (\exists n)\,\psi(x, g^n(x))$ is also inductive.

The condition (3.4) of sufficiency can be transformed in a similar way. Let z be an output value of x in P. We will write an output-input relation

$$W(x_0, z) \equiv V(x_0) \wedge P(x_0) = z\,. \tag{5.3}$$

Then ψ is sufficient for W iff

$$(\forall x_0)\,(\psi(f(x_0), y) \supset W(x_0, h(y)))\,.$$

The mutual transformation of subgoal induction and invariant method is well known (see Morris and Wegbreit [4] or Reynolds and Yeh [7]).

Remember $J(x_0)$ to be an input and $K(x_0, x)$ an output specification. On the assumption of termination $W(x, z) \equiv (J(x) \supset K(x, z))$, as follows directly from (3.2) and (5.3). Then an *invariant* of the loop at cutpoint C that is adequate to J and K is

$$I(x_0, x) \equiv (\forall y)\,(\psi(x, y) \supset W(x_0, h(y)))\,.$$

On the contrary:

$$\psi(x, y) \equiv (\forall x_0)\,(I(x_0, x) \supset W(x_0, h(y)))\,.$$

The transformation of the IAM to the invariant method is as follows:

$$I(x_0, x) \equiv (\forall y)\,[(\exists n)(\exists k)[(Q1(x) \supset Q2(x, g^n(x)) \wedge \neg\, t(g^{n+k}(x))) \wedge y = \\ = g^{n+k}(x)] \supset W(x_0, h(y))]\,.$$

The opposite transformation:

$$\varphi(x) \equiv (\exists n)(\forall x_0)\,[I(x_0, x) \supset (J(x_0) \supset K(x_0, h(g^n(x))))]\,.$$

Because

$$\neg\, t(g^n(x)) \wedge I(x_0, g^n(x)) \supset K(x, h(g^n(x)))\,,$$

we can strengthen

$$\bar{\varphi}(x) \equiv (\exists n)(\forall x_0)\,[I(x_0, x) \wedge J(x_0) \supset \neg\, t(g^n(x)) \wedge I(x_0, g^n(x))]\,.$$

Note that $\bar{\varphi}$, when rewritten to intermittent-assertion form, is just the lemma that is stated by Manna and Waldinger [2] for that transformation.

Let us have all the assumptions stated in Section 3: a one-loop program P, a property $V(x_0)$ and a predicate $\varphi(x)$ of the forms (3.1), (3.2) and (3.3), respectively. The predicate φ is sufficient for V ((3.4) holds) and it should be inductive on $(D, \prec)$, where D is a domain of the loop again but $\prec$ is an ordering that does not follow the program structure. It is derived from the specifications J and K, specially given for the loop at φ.

A. The ordering of D can be derived from input data structure, which should be decomposed by the computation.

For instance, let us have a set S^* of strings over S as a domain, with ordering:

$$tail(x) \prec x \text{ for each } x \in S^*, \quad x \neq \Lambda \quad \text{(empty string)}.$$

Now φ is inductive on $(S^*, \prec)$ if all the following conditions hold:

$$\begin{gathered}(\forall x \in S^*)\,(x = \Lambda \supset \varphi(x)),\\ (\forall x \in S^*)\,(x \neq \Lambda \supset g(x) \prec x),\\ (\forall x \in S^*)\,(x \neq \Lambda \wedge \varphi(g(x)) \supset \varphi(x)).\end{gathered}$$

As an example of such proof, see the proof by IAM of an imaginary sequential operating system presented in [2].

B. The ordering of D can be derived from a structure of the expected function of the loop.

For instance, let φ specify $(\exists n)\, g^n(x) = F(x)$ and function F is defined recursively

$$F(x) \equiv \text{if } B(x) \text{ then } G(F(S(x))) \text{ else } H(x)$$

for a predicate B and some functions G, S, H.

A decomposition of D is again looked for so that $S(x) \prec x$ for each $x \in D$. Such ordering is well-founded iff the function $F(x)$ is defined for each $x \in D$.

The verification conditions will be now:

$$\begin{gathered}(\forall x \in D)\,(\neg B(x) \supset \varphi(x)),\\ (\forall x \in D)\,(B(x) \supset g(x) \prec x),\\ (\forall x \in D)\,(B(x) \wedge \varphi(g(x)) \supset \varphi(x)).\end{gathered}$$

As examples of such proofs see the proofs of the programs for counting the tips of a tree or for computing the Ackermann function in [2]. The latter will be analysed in detail in the next section (Example 2).

C. As a special instance, the function of the loop can be specified by another iterative program. The decomposition is done according to the structure of the second program. This is the case of proving equivalence of two iterative programs.

Let us give some remarks.

In proofs of second type, the domain of proof is decomposed according to the structure of specifications. But it may happen that such a decomposition merges into a decomposition according to the structure of program. Then the proof merges into the proof of the first type.

If the proof differs from a proof of the first type then there is no direct way how to express it by using subgoal induction or invariant method.

In practice, the predicate φ that is proved in this way can be weaker, because the proof uses only the properties of the own problem and does not depend on technical details of the program. That is why such a proof is advantageous especially for unstructured program segments.

7. PROGRAMS WITH MORE COMPLEX STRUCTURE

In this section we shall try to generalize the above stated verification conditions, better to say, to find rules for its synthesis.

For the simplicity, we confine ourselves only to structured programs with only while loops, if-then-else branching and sequential concatenation of statements.

Suppose that each loop has a cutpoint, domain of values at the cutpoint and a predicate which is total over the domain and which specifies the behaviour of the loop (its function). If we prove that the predicate is inductive on the domain we can see the loop as a single command and we do not need to deal with its structure from now on.

We can decompose the whole program into individual levels of abstraction. At each level we can distinguish only sequential structure or branching. An arbitrary program segment in an arbitrary level can be divided into finite number of paths according to branching.

Some properties are to be proved on a domain of a segment. Such proof requires finding functions of each path of that segment and on this occasion it uses all specifications of loops (already proved predicates) lying on the path. We can shift such predicates back (by backward substitution) to apply them for data of the domain of the segment. We shall call the conjunction of all the shifted predicates along a path as a *precondition of the path.*

The proof of the whole program will be performed level by level from the bottom up.

Let us prove that a predicate φ is inductive on $(D, \prec)$ where D is a domain of a loop at a certain level and $\prec$ ordering was defined according to the loop structure (the first type of proof). Let the tail of the loop be divided into i paths. Let $t_1, t_2, \ldots, t_i$

be their conditions, $g_1, g_2, \ldots, g_i$ be their functions and let $\Phi_1, \Phi_2, \ldots, \Phi_i$ be their preconditions. Now we can state the verification conditions for that level:

$$(\forall x \in D)\,(\neg t_1(x) \wedge \neg t_2(x) \wedge \ldots \wedge \neg t_i(x) \supset \varphi(x))$$

and for each $k = 1, 2, \ldots, i$

$$(\forall x \in D)\,(t_k(x) \supset g_k(x) \prec x)\,,$$
$$(\forall x \in D)\,(t_k(x) \wedge \Phi_k(x) \wedge \varphi(g_k(x)) \supset \varphi(x))\,.$$

At the highest level, let the program have just j paths, their conditions be $t_1, t_2, \ldots, t_j$ and their preconditions be $\Phi_1, \Phi_2, \ldots, \Phi_j$. It must be verified for each path that its precondition is sufficient for the required property V:

$$(\forall x_0)\,(t_1(x_0) \wedge \Phi_1(x_0) \supset V(x_0))\,,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$(\forall x_0)\,(t_j(x_0) \wedge \Phi_j(x_0) \supset V(x_0))\,.$$

Example 1. (Manna and Waldinger [2]) The following program computes the greatest common divisor of two positive integers, defined as follows:
$gcd(x, y) = \max\{u : u \mid x \text{ and } u \mid y\}$.

```
A:  input(x, y);
C1:   while $x \neq y$ do begin
        C2: while $x > y$ do $x := x - y$;
        C3: while $x < y$ do $y := y - x$;
              end;
B:  output(y)
```

We want to prove

Theorem.

If ***Sometime*** (At A and $x = a,\ y = b$ and $a, b > 0$)

then ***Sometime*** (At B and $y = gcd(a, b)$)

with the help of following lemmas:

Lemma 1.

If ***Sometime*** (At $C1$ and $x = a_1,\ y = b_1$ and $x, y > 0$)

then ***Sometime*** (At $C1$ and $x = y$ and $y = gcd(a_1, b_1)$).

Lemma 2.

$$\text{If } \textbf{\textit{Sometime}}\ (\text{At } C2 \text{ and } x = a_2,\ y = b_2 \text{ and } x, y > 0,\ x > y)$$
$$\text{then } \textbf{\textit{Sometime}}\ (\text{At } C2 \text{ and } x < a_2,\ y = b_2,\ x \leqq y,\ x, y > 0$$
$$\text{and } gcd(x, y) = gcd(a_2, b_2)) .$$

Lemma 3.

$$\text{If } \textbf{\textit{Sometime}}\ (\text{At } C3 \text{ and } x = a_3,\ y = b_3,\ x, y > 0,\ x < y)$$
$$\text{then } \textbf{\textit{Sometime}}\ (\text{At } C3 \text{ and } x = a_3,\ y < b_3,\ x \geqq y,\ x, y > 0$$
$$\text{and } gcd(x, y) = gcd(a_3, b_3)) .$$

If we transform them to our formalism we shall get

$$V(x, y) , \quad \varphi\ 1(x, y) , \quad \varphi\ 2(x, y) , \quad \varphi\ 3(x, y) .$$

We can see that the domain of the whole program and of each loops is a set of ordered pairs $\langle x, y \rangle$.

Prove the bottom level. We can define a well-founded orderings, as follows

$$\langle x_1, y_1 \rangle \prec_2 \langle x_2, y_2 \rangle \quad \text{iff} \quad x_1 < x_2 ,$$
$$\langle x_1, y_1 \rangle \prec_3 \langle x_2, y_2 \rangle \quad \text{iff} \quad y_1 < y_2 ,$$

for the domain of the second and the third loop, respectively. Verification conditions:

$$(\forall \langle x, y \rangle)\,(x \leqq y \supset \varphi 2(x, y)) ,$$
$$(\forall \langle x, y \rangle)\,(x > y \supset (\langle x - y, y \rangle \prec_2 \langle x, y \rangle) ,$$
$$(\forall \langle x, y \rangle)\,(x > y \wedge \varphi 2(x - y, y) \supset \varphi 2(x, y)) ;$$
$$(\forall \langle x, y \rangle)\,(x \geqq y \supset \varphi 3(x, y)) ,$$
$$(\forall \langle x, y \rangle)\,(x < y \supset \langle x, y - x \rangle \prec_3 \langle x, y \rangle) ,$$
$$(\forall \langle x, y \rangle)\,(x < y \wedge \varphi 3(x, y - x) \supset \varphi 3(x, y))$$

are easy to verify. So $\varphi 2$, $\varphi 3$ are inductive on $(D, \prec_2)$, $(D, \prec_3)$, respectively.

Now prove the higher level. The tail of the outer loop involves just three paths: either the first inner loop is gone through or the second inner loop or both.

Let as denote the function of the first and the second inner loop as $G2 = \langle G2_x, G2_y \rangle$ and $G3 = \langle G3_x, G3_y \rangle$, respectively. (You can see that $\varphi 2$ determines that

$(\exists\, n)\; G2_x(x, y) = x - n\,.\,y$ and $G2_y(x, y) = y$, similarly $\varphi 3$ determines $G3$.) The conditions of the three paths are:

$$t_1(x, y) \equiv (x > y \wedge G2_x(x, y) \geqq G2_y(x, y))\,,$$
$$t_2(x, y) \equiv x < y\,,$$
$$t_3(x, y) \equiv (x > y \wedge G2_x(x, y) < G2_y(x, y))\,,$$

their preconditions:

$$\Phi_1(x, y) \equiv \varphi 2(x, y)\,,$$
$$\Phi_2(x, y) \equiv \varphi 3(x, y)\,,$$
$$\Phi_3(x, y) \equiv \varphi 2(x, y) \wedge \varphi 3(G2(x, y))$$

and their functions:

$$g_1(x, y) = G2(x, y)\,,$$
$$g_2(x, y) = G3(x, y)\,,$$
$$g_3(x, y) = G3(G2(x, y))\,.$$

If $\prec_1$ is an ordering (defined below) of the domain D then the verification conditions are:

(7.1) $(\forall\langle x, y\rangle)\,(x = y \supset \varphi 1(x, y))\,,$

(7.2) $(\forall\langle x, y\rangle)\,(t_1(x, y) \supset g_1(x, y) \prec_1 \langle x, y\rangle)\,,$

(7.3) $(\forall\langle x, y\rangle)\,(t_1(x, y) \wedge \Phi_1(x, y) \wedge \varphi 1(g_1(x, y)) \supset \varphi 1(x, y))\,,$

(7.4) $(\forall\langle x, y\rangle)\,(t_2(x, y) \supset g_2(x, y) \prec_1 \langle x, y\rangle)\,,$
$(\forall\langle x, y\rangle)\,(t_2(x, y) \wedge \Phi_2(x, y) \wedge \varphi 1(g_2(x, y)) \supset \varphi 1(x, y))\,,$

(7.5) $(\forall\langle x, y\rangle)\,(t_3(x, y) \supset g_3(x, y) \prec_1 \langle x, y\rangle)\,,$
$(\forall\langle x, y\rangle)\,(t_3(x, y) \wedge \Phi_3(x, y) \wedge \varphi 1(g_3(x, y)) \supset \varphi 1(x, y))\,.$

Note that from $\Phi_1 \equiv \varphi 2$ (see Lemma 2) $G2_x \leqq G2_y$ holds. Then from the conjunction $t_1 \wedge \Phi_1$, it follows $G2_x = G2_y$ and thus the same for g_1. Then it follows directly from (7.1) that $\varphi 1(g_1(x, y))$ is true. Thus $\varphi 1(g_1(x, y))$ is redundant in (7.3) and so the condition (7.2), too.

We can define $\prec_1$ ordering in that case only to fit (7.4) and (7.5):

$$\langle x_1, y_1\rangle \prec_1 \langle x_2, y_2\rangle \quad \text{iff} \quad y_1 < y_2\,.$$

That is why the IAM-approach is more effective than the Floyd's technique for proving termination (see this example at [2]) in some cases.

For the highest level, it will do to prove

$$(\forall\langle x, y\rangle)\,(\varphi 1(x, y) \supset V(x, y)),$$

what is straightforward.

Let us return to a level of complex program, where we want to prove that some predicate is inductive on the domain by a proof of the second type. The proof will go through the tail of the loop just only according to a structure given by its specification. See the next example:

Example 2. (Manna and Waldinger [2]) The computation of the *Ackermann function*:

$$\begin{aligned} A(x, y) \equiv {} & \text{if } x = 0 \text{ then } y + 1 \\ & \quad \text{else if } y = 0 \text{ then } A(x - 1, 1) \\ & \quad\quad \text{else } A(x - 1, A(x, y - 1)) \end{aligned}$$

The program uses an array stack s:

```
A:  input($x_0$, $y_0$) ;
    $s[1] := x_0$ ;
    $s[2] := y_0$ ;
    $i := 2$ ;
C:  while $i \neq 1$ do if $s[i - 1] = 0$
                        then begin $s[i - 1] := s[i] + 1$ ;
                                   $i := i - 1$
                             end
                        else if $s[i] = 0$
                             then begin $s[i - 1] := s[i - 1] - 1$ ;
                                        $s[i] := 1$
                                  end
                             else begin $s[i + 1] := s[i] - 1$ ;
                                        $s[i] := s[i - 1]$ ;
                                        $s[i - 1] := s[i - 1] - 1$ ;
                                        $i := i + 1$
                                  end;
B:  output($s[1]$)
```

We want to prove

Theorem.

$$\text{If } \textit{\textbf{Sometime}} \text{ (At } A \text{ and } x_0, y_0 > 0)$$
$$\text{then } \textit{\textbf{Sometime}} \text{ (At } B \text{ and } s[1]' = A(x_0, y_0)).$$

with the help of the following lemma.

Lemma.

If ***Sometime*** (At C and $i = index$, $index \geqq 2$, $s[1 : index - 2] =$ $= stack$, $s[index - 1] = a$, $s[index] = b$)

then ***Sometime*** (At C and $i = index - 1$, $s[1 : index - 2] =$ $= stack$ and $s[index - 1] = A(a, b)$) .

If we transform them to our formalism we will get $V(x_0, y_0)$ and $\varphi(s, i)$.

The domain of the whole program is a set of ordered pairs $\langle x_0, y_0 \rangle$ and the domain D of the loop is a set of ordered pairs $\langle s, i \rangle$, where i is a positive integer and s is an array stack of size i.

We can then easily prove that φ is sufficient for V:

$$(\forall \langle x_0, y_0 \rangle)\,(\varphi(s_0, 2) \supset V(x_0, y_0))\,,$$

where $s_0[1] = x_0$ and $s_0[2] = y_0$.

The domain D of the loop is decomposed according to the structure of $A(s[i - 1], s[i])$:

$$\langle s_1, i_1 \rangle \prec \langle s_2, i_2 \rangle \quad \text{iff} \quad \langle s_1[i_1 - 1], s_1[i_1] \rangle \prec_{\mathrm{L}} \langle s_2[i_2 - 1], s_2[i_2] \rangle\,,$$

where $\prec_{\mathrm{L}}$ is the lexicographic ordering.

Recursive definition of $A(s[i - 1], s[i])$ involves three paths. Their conditions are

$$B_1(s, i) \equiv s[i - 1] = 0\,,$$
$$B_2(s, i) \equiv s[i - 1] \neq 0 \wedge s[i] = 0\,,$$
$$B_3(s, i) \equiv s[i - 1] \neq 0 \wedge s[i] \neq 0\,.$$

Note that B_1 is a condition for $\prec$-least elements of D.

Let g_1, g_2, g_3 denote the functions of the paths, which correspond to the conditions B_1, B_2, B_3, respectively. Let G denote the function of the loop which is determined by the predicate φ.

Now we can write the verification conditions:

$$(\forall \langle s, i \rangle)\,(B_1(s, i) \supset \varphi(s, i))\,,$$
$$(\forall \langle s, i \rangle)\,(B_2(s, i) \supset g_2(s, i) \prec \langle s, i \rangle)\,,$$
$$(\forall \langle s, i \rangle)\,(B_2(s, i) \wedge \varphi(g_2(s, i)) \supset \varphi(s, i))\,,$$
$$(\forall \langle s, i \rangle)\,(B_3(s, i) \supset g_3(s, i) \prec \langle s, i \rangle)\,,$$
$$(\forall \langle s, i \rangle)\,(B_3(s, i) \supset G(g_3(s, i)) \prec \langle s, i \rangle)\,,$$
$$(\forall \langle s, i \rangle)\,(B_3(s, i) \wedge \varphi(g_3(s, i)) \wedge \varphi(G(g_3(s, i))) \supset \varphi(s, i))\,.$$

In the last condition, the two applications of the inductive hypothesis answer to the double recursive call in the definition of the Ackermann function for the condition B_3.

8. CONCLUSION

We can see that intermittent-assertion method is more powerful than other known methods for program verification, because it involves the others and even something more. Its proof need not depend on the program structure but only on the problem itself. That is the main advantage of the method.

Our expressing of IAM as a structural induction served only for an analysis but not for a practical hand-verification. Intermittent assertions are more simple and more intelligible.

The presented rules for synthesis of verification conditions could serve for the control of mechanical proofs.

(Received August 11, 1978.)

ACKNOWLEDGMENT

I wish to thank Dr. V. Rajlich ior his critical reading of this paper.

REFERENCES

[1] R. M. Burstall: Program Proving as Hand Simulation with a Little Induction. Information Processing 74, North-Holland Publ. Comp., 1974, 308—312.

[2] Z. Manna, R. Waldinger: Is "sometime" sometimes better than "always"? Intermittent assertions in proving program correctness. Stanford Artif. Intel. Lab. STAN-CS-76-558, June 1976 (also in CACM *21* (1978), 2, 159—179).

[3] R. W. Floyd: Assigning meanings to programs. Proceedings of Symposium in Applied Math., American Math. Soc. 1967, 19—32.

[4] J. H. Morris Jr., Ben Wegbreit: Subgoal induction. CACM *20* (1977), 4, 209—220.

[5] R. M. Burstall: Proving properties of programs by structural induction. The Comp. Jour. *12* (1969), 1, 41—48.

[6] Z. Manna: Mathematical Theory of Computation. McGrow-Hill Book Comp., New York 1974.

[7] C. Reynolds, R. T. Yeh: Induction as the basis for program verification. IEEE Tran. on Software Engineering *SE-2* (1976), 4, 244—252.

RNDr. Karel Vosátka, Výzkumný ústav matematických strojů (Research Institute for Mathematical Machines), Loretánské nám. 3, 118 55 Praha 1. Czechoslovakia.