Joint NCC & IIS Bull., Comp. Science, 5 (1996), 1–22

On a symbolic method of verification for definite iteration over data structures*

V.A. Nepomniaschy

A verification method is proposed for definite iteration over different data structures. The method is based on a replacement operation which expresses the definite iteration effect in a symbolic form and belongs to a specification language. The method includes a proof rule for the iteration without invariants and inductive proof principles for proving verification conditions which contain the replacement operation. As a case study, a parallel replacement operation for arrays is considered in order to simplify the proof of verification conditions.

1. Introduction

Formal program verification which means the proof of consistency between programs and their specifications is successfully developed. The axiomatic style of verification is based on the Hoare method [7] which consists of the following stages: constructing the pre-, post-conditions and loop invariants; deriving verification conditions with the help of proof rules and proving them. The construction of loop invariants is a difficult stage of the verification process.

The functional style of verification proposed by Mills and others [1, 9, 12] assumes that each loop is annotated with a function expressing the loop effect. The functions are closely related to loop invariants but differences can be noticed [4]. As before, the construction of the functions associated with loops is a difficult problem.

Loops can be divided in two groups called definite and indefinite iterations. Typical examples are Pascal for- and while-loops. Definite iteration has the advantage over indefinite one because of its termination provided the loop body terminates. Definite iteration is iteration over all elements of a list, set, file, array, tree or other data structure. It is often used in application programs [19].

The verification styles mentioned above are oriented to indefinite iteration. To verify definite iteration we can at first transform it to indefinite

*This work is partially supported by grant INTAS-RFBR 95-0378 from International Association for the promotion of cooperation with scientists from the New Independent States of the former Soviet Union (INTAS) and Russian Foundation for Basic Research (RFBR).

one and then use the approaches mentioned above, but we lose their benefits which we achieve by using definite iteration. So, it is of interest to simplify the verification of definite iteration. In [2, 6, 8] advantages of for-loops over unordered and linear ordered sets are discussed, and proof rules which take into account the specific character of the for-loops are proposed. In [19] the functional method for verifying definite iteration is described. In both approaches verification of definite iteration remains a difficult problem because the construction of loop invariants or the functions associated with loops is needed.

In [13, 14, 15, 18] we proposed a symbolic method of verifying loops over unordered and linear ordered sets which is different from the mentioned approaches. This method was suitable to loops which had the statement of assignment to array elements as the loop body. The main idea of the method is to use the symbols of invariants instead of the invariants in verification conditions and a special technique based on the loop properties for proving the verification conditions. Thus, the symbolic method allows us to verify the for-loops without loop invariants or their analogs.

The purpose of this paper is to develop the symbolic method of verification for definite iteration without restrictions on the loop bodies. The method is based on a replacement operation introduced in the specification language which represents the effect of the iteration by means of a symbolic form. The iteration invariant can be expressed with the help of the replacement operation. To prove verification conditions containing the replacement operation, a proof technique is proposed which includes axioms for this operation and inductive proof principles. In order to simplify the proof of verification conditions for loop bodies with arrays, a parallel replacement operation is considered. The use of the method is demonstrated by some examples.

The rest of this paper consists of 8 sections. In Section 2 the notion of data structures is defined and useful functions over the structures are introduced. Definite iteration over data structures and its axiomatic semantics are described in Section 3. In Section 4 the replacement operation is defined and a proof rule using the operation instead of a loop invariant is given. Inductive proof principles for proving assertions containing the replacement operation are presented in Sections 5 and 6. A case of study of loop bodies with arrays is considered in Sections 7 and 8. In conclusion, results and perspectives of the symbolic verification method are discussed. Proofs of all theorems are given in Appendix.

2. Data structures

We introduce the following notation. Boolean operations are denoted by symbols $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), $\neg$ (negation), $\leftrightarrow$ (equivalence). We suppose that all free variables are bound by universal quantifiers in axioms, theorems and other formulas. Let $\{s_1, s_2, \ldots, s_n\}$ be the multiset which consists of elements $s_1, \ldots, s_n$. The membership of s in the multiset T is denoted by $s \in T$. Let $T_1 - T_2$ be the difference of multisets T_1 and T_2. For the function $f(x)$ we denote $f^0(x) = x$, $f^i(x) = f(f^{i-1}(x))$ $(i = 1, 2, \ldots)$.

Let us remind the notion of data structures which contain a finite number of elements [19]. Let $memb(S)$ be the multiset of elements of the structure S, and $|memb(S)|$ be the power of the multiset $memb(S)$. For a structure S the following three operations are defined: $empty(S)$ is a predicate whose value is true if $memb(S)$ is empty and false otherwise; $choo(S)$ is a function which returns an element s of $memb(S)$; $rest(S)$ is a function which returns a structure S' of the same type as S such that $memb(S') = memb(S) - \{choo(S)\}$. The functions $choo(S)$ and $rest(S)$ will be undefined if and only if $empty(S)$. Typical examples of the structures are sets, sequences, lists, strings, arrays, files and trees.

We introduce a number of useful functions related to a structure S in the case of $\neg empty(S)$. We denote $s_i = choo(rest^{i-1}(S))$ for $i = 1, \ldots, n$ provided $\neg empty(rest^{n-1}(S))$ and $empty(rest^n(S))$. So, $memb(S) = \{s_1, s_2, \ldots, s_n\}$. Here $last(S)$ is a partial function such that $last(S) = s_n$. $next(S, s)$ is a partial function such that $next(S, s_{i-1}) = s_i$ for $i = 2, \ldots, n$. $next(S, s)$ will be undefined for $s \notin memb(S)$ or $s = last(S)$. For elements of $memb(S)$ we will use the order relation such that $s_i < s_j \leftrightarrow i < j$.

Let $str(s)$ denote a structure S which contains the only element s. The following axiom defines the structure $str(s)$.

Ax1. $\neg empty(str(s)) \wedge empty(rest(str(s))) \wedge choo(str(s)) = s$.

Let $M = [m_1, \ldots, m_k]$ denote a vector which consists of elements m_i $(1 \le i \le k)$ ordered by the relation $<$ such that $m_i < m_j \leftrightarrow i < j$. We will use $pred(m_j)$ $(j = 1, \ldots, k)$ to denote the set $\{m_i \mid 1 \le i < j\}$ if $j > 1$ and the *empty* set if $j = 1$. We will consider the vector $M = [m_1, \ldots, m_k]$ as a structure such that $choo(M) = m_1$, $rest(M) = [m_2, \ldots, m_k]$ (if $k \ge 2$), $empty(rest(M))$ (if $k = 1$). We consider $m \in M$ to be a shorthand for $m \in memb(M)$. Let $con(M_1, M_2)$ be the concatenation operation of vectors M_1 and M_2.

For a structure S we assume that $vec(S) = [s_1, \ldots, s_n]$ if $\neg empty(S)$, $memb(S) = \{s_1, \ldots, s_n\}$ and $s_i = choo(rest^{i-1}(S))$ $(i = 1, \ldots, n)$. The following axioms define the function $vec(S)$ for all cases.

Ax2. $empty(S) \leftrightarrow empty(vec(S))$.

Ax3. $\neg empty(S) \rightarrow choo(vec(S)) = choo(S) \wedge$
$rest(vec(S)) = vec(rest(S))$.

For structures S_1 and S_2 let us define a concatenation operation $con(S_1, S_2)$ by the following axioms.

Ax4. $empty(S_1) \to con(S_1, S_2) = S_2$.

Ax5. $\neg empty(S_1) \to choo(con(S_1, S_2)) = choo(S_1) \wedge$
$rest(con(S_1, S_2)) = con(rest(S_1), S_2)$.

We consider $con(S, s)$, $con(s, S)$, $con(S_1, S_2, S_3)$ to be a shorthand for $con(S, str(s))$, $con(str(s), S)$, $con(con(S_1, S_2), S_3)$ respectively. Notice that the axioms Ax4 and Ax5 hold for vectors S_1 and S_2. Hence the concatenation operation for structures generalizes the same operation for vectors. It should be noted that the axioms Ax1–Ax5 imply the following theorems expressing some important properties of the concatenation operation for structures.

Th1. $\neg empty(S) \to con(choo(S), rest(S)) = S$,

Th2. $con(vec(S_1), vec(S_2)) = vec(con(S_1, S_2))$.

For a structure S we introduce a useful function $head(S)$ which returns a structure such that $vec(head(S)) = [s_1, \ldots, s_{n-1}]$ provided $vec(S) = [s_1, \ldots, s_n]$. The function is defined by the following axioms.

Ax6. $|memb(S)| \leq 1 \leftrightarrow empty(head(S))$.

Ax7. $\neg empty(head(S)) \to (choo(head(S)) = choo(S) \wedge$
$rest(head(S)) = head(rest(S)))$.

It follows from the axioms that an important property symmetric to Th1 is satisfied.

Th3. $\neg empty(S) \to con(head(S), last(S)) = S$.

3. Definite iteration over structures

We recall the notion of definite iteration over structures from [19]. Let us consider the statement

$$\textbf{for } x \textbf{ in } S \textbf{ do } v := body(v, x) \textbf{ end} \tag{1}$$

where S is the structure, x is the variable called the loop parameter, v is the data vector of the loop body ($x \notin v$) and $v := body(v, x)$ represents the loop body computation. We suppose that the loop body uses variables from v (and x), does not change the loop parameter x and iterates over all elements of the structure S. So, the loop body terminates for every $x \in memb(S)$.

Operational semantics of iteration (1) is defined as follows. Let v_0 be the vector of initial values of variables from the vector v. The result of the iteration is $v = v_0$ if $empty(S)$. If $\neg empty(S)$ and $vec(S) = [s_1, \ldots, s_n]$, the loop body iterates sequentially for x defined as $s_1, s_2, \ldots, s_n$.

To describe the axiomatic semantics of iteration (1), we introduce the following notation. Let P, Q, inv and $prog$ denote a pre-condition, a post-condition, an invariant, and a program fragment, respectively.

$\{P\}$ *prog* $\{Q\}$ denotes partial correctness of the program *prog* with respect to the pre-condition P and the post-condition Q. Let $R(y \leftarrow exp)$ (or $R(exp1 \leftarrow exp)$) be a result of substitution of an expression exp for all occurrences of a variable y (or an expression $exp1$) into a formula R. Let $R(vec \leftarrow vexp)$ denotes a result of the synchronous substitution of components of an expression vector $vexp$ for all occurrences of corresponding components of a vector vec into a formula R. Axiomatic semantics of iteration (1) is given by the following proof rule.

rl1. $cond1 \wedge cond2 \wedge cond3 \vdash$
$\{P\}prog\{inv\}$ **for** x **in** S **do** $v := body(v, x)$ **end** $\{Q\}$

where the post-condition Q does not depend on the loop parameter x,
cond1:$\{P\}prog\{(\neg empty(S) \rightarrow inv(x \leftarrow choo(S))) \wedge (empty(S) \rightarrow inv)\}$,
cond2:$\{inv \wedge x \in memb(S)\}v := body(v,\ x)\ \{(x \neq last(S) \rightarrow inv(x \leftarrow next(S, x))) \wedge (x = last(S) \rightarrow Q)\}$,
cond3: $inv \wedge empty(S) \rightarrow Q$.

Let PROOF denote the standard system of proof rules for usual statements including while-loop and assignment to variables which have a type of the loop parameter. The system for Pascal is presented in [7]. The following theorem justifies the proof rule rl1.

Th4. The proof rule rl1 is derived in the standard system PROOF.

4. Specifying the iteration by replacement operation

We associate a function $body(v,\ x)$ with the right part of the body of iteration (1) such that the body has the same form $v := body(v,\ x)$. To present the effect of iteration (1), let us define a replacement operation $rep(v,\ S,\ body)$ to be a vector v_n such that $v_0 = v$; $n = 0$ provided $empty(S)$; $v_i = body(v_{i-1}, s_i)$ for all $i = 1, \ldots, n$ provided $\neg empty(S)$ and $vec(S) = [s_1, \ldots, s_n]$. The following axioms define the replacement operation for all cases.

Ax8. $empty(S) \rightarrow rep(v,\ S,\ body) = v$.

Ax9. $\neg empty(S) \rightarrow rep(v,\ S,\ body) =$
$rep(body(v,\ choo(S)),\ rest(S),\ body)$.

Important properties of the replacement operation are expressed by the following theorems.

Th5. $rep(v,\ con(S_1,\ S_2),\ body) = rep(rep(v,\ S_1,\ body),\ S_2,\ body)$.

Th6. $\neg empty(S) \rightarrow rep(v, S, body) =$
$body(rep(v, head(S), body), last(S))$.

The replacement operation allows us to eliminate the loop invariant from the proof rule rl1 for iteration (1). Indeed, let us consider the following proof rule.

rl2. $\{P\}prog\{Q(v \leftarrow rep(v,\ S,\ body))\} \vdash$
$\{P\}prog;\ \mathbf{for}\ x\ \mathbf{in}\ \ S\ \mathbf{do}\ v := body(v,\ x)\ \mathbf{end}\ \{Q\}$

where the post-condition Q does not depend on the loop parameter x, the variables from the vector v are unchanged in the body part of the $rep(v,\ S,\ body)$ since substitutions for the occurrences of the variables are not performed when the rule is used. The proof rule is justified by the following theorem which can be proved with the help of the theorem Th4.

Th7. The proof rule rl2 is derived in the standard system PROOF.

5. Backward induction principle

Verification conditions including the replacement operation are generated by means of the proof rule rl2. To prove the verification conditions, we need a special technique. We present the technique based on principles of induction by $|memb(S)|$. In this section a backward induction principle is described. The principle allows us to prove a property of the replacement operation over a structure S if the property for the structure $rest(S)$ is assumed.

Let $prop(rep(v,\ S,\ body))$ denote a property expressed by a first-order logic formula with the only free variable S. The formula is constructed from the replacement operation $rep(v, S, body)$, function symbols, variables and constants by means of Boolean operations, first-order quantifiers and substitution of constants for variables from v.

Induction principle 1. The property $prop(rep(v,\ S,\ body))$ holds for each structure S if the following two conditions hold for each structure S:

1) $empty(S) \rightarrow prop(rep(v,\ S,\ body))$.

2) $\neg empty(S) \wedge prop(rep(v,\ rest(S),\ body)) \rightarrow prop(rep(v,\ S,\ body))$.

The following corollary is evident from the induction principle 1 and the axioms Ax8 and Ax9.

Corollary 1. *The property $prop(rep(v,\ S,\ body))$ holds for each structure S if the following two conditions hold for each structure S:*

1) $empty(S) \rightarrow prop(rep(v,\ S,\ body) \leftarrow v)$.

2) $\neg empty(S) \wedge prop(rep(v,\ rest(S),\ body)) \rightarrow prop(rep(v,\ S,\ body) \leftarrow rep(body(v,\ choo(S)),\ rest(S),\ body))$.

To illustrate the use of the backward induction principle we consider a simple example from [19].

Example 1. String concatenation.

The following annotated program concatenates strings Y_0 and Y_1 where Y_0 is an initial value of Y.

$$\{P\}\ \textbf{for}\ x\ \textbf{in}\ Y_1\ \textbf{do}\ Y := con(Y,\ x)\ \textbf{end}\ \{Q\} \tag{2}$$

where $P : Y = Y_0$, $Q : Y = con(Y_0,\ Y_1)$. The following verification condition is generated by means of the proof rule rl2 when the program *prog* is *empty*.

$$Y = Y_0 \rightarrow rep(Y,\ Y_1,\ con) = con(Y_0,\ Y_1). \tag{3}$$

We will prove that the following property is equivalent to condition (3).

$$prop(rep(Y_0,\ Y_1,\ con)) : \forall Y_0 rep(Y_0,\ Y_1,\ con) = con(Y_0,\ Y_1). \tag{4}$$

We apply corollary 1. If $empty(Y_1)$, then the condition $\forall Y_0\ Y_0 = con(Y_0, Y_1)$ is obviously true. Suppose that

$$\neg empty(Y_1) \wedge \forall Y_0 rep(Y_0,\ rest(Y_1),\ con) = con(Y_0,\ rest(Y_1)). \tag{5}$$

It remains to show that

$$\forall Y_0 rep(con(Y_0,\ choo(Y_1)),\ rest(Y_1),\ con) = con(Y_0,\ Y_1). \tag{6}$$

By condition (5), condition (6) is equivalent to

$$\forall Y_0 con(con(Y_0,\ choo(Y_1)),\ rest(Y_1)) = con(Y_0,\ Y_1). \tag{7}$$

Condition (7) follows from Theorem Th1 and the standard property of string concatenation

$$con(con(Y_1,\ Y_2),\ Y_3) = con(Y_1,\ con(Y_2,\ Y_3)). \tag{8}$$

6. Forward induction principle

In this section we present a forward induction principle. The principle allows us to prove a property of the replacement operation over a structure S if the property for the structure $head(S)$ is assumed.

Induction principle 2. The property $prop(rep(v, S, body))$ holds for each structure S if the following two conditions hold for each structure S:

1) $empty(S) \rightarrow prop(rep(v,\ S,\ body))$.
2) $\neg empty(S) \wedge prop(rep(v,\ head(S),\ body)) \rightarrow prop(rep(v,\ S,\ body))$.

The following corollary is evident from the induction principle 2, axiom Ax8 and theorem Th6.

Corollary 2. *The property $prop(rep(v,\ S,\ body))$ holds for each structure S if the following two conditions hold for each structure S:*
1) $empty(S) \rightarrow prop(rep(v,\ S,\ body) \leftarrow v)$.
2) $\neg empty(S) \wedge prop(rep(v,\ head(S),\ body)) \rightarrow$
$prop(rep(v,\ S,\ body) \leftarrow body(rep(v,\ head(S),\ body),\ last(S)))$.

We consider an example from [19] in order to illustrate the use of the forward induction principle. Let M_Z denote the projection of a vector M of values of variables $Z, \ldots$ on the variable Z.

Example 2. Copying an ordered file with insertion.

To specify a copying program we introduce the following notation. Let F and G be the files considered as structures; Ω denotes the empty file; $ord(F)$ is a predicate whose value is true if F was sorted in ascending order $\leq$ of elements and false otherwise. We assume that $ord(\Omega)$ and $\omega < y$ for each defined element y and the undefined element ω. Here $del(F,\ y)$ is a function which returns a file resulted from the file F by eliminating the first occurrence of the element y. If the element y is not contained in the file F, then $del(F,\ y) = F$; $hd(F,\ y)$ is a function which returns a file resulted from the file F by eliminating its tail which begins with the first occurence of the element y; $tl(F,\ y)$ is a function which returns a file resulted from the file F by eliminating its head which ends with the first occurence of the element y. If the element y is not contained in the file F, then $hd(F,\ y) = tl(F,\ y) = F$. Here, $y > F$ is a predicate whose value is true, if $empty(F)$ or $\forall x \in memb(F)$ $y > x$ and false otherwise.

The following annotated program copies the sorted file F to the file G inserting an element w in its proper place.

$\{P\}$ $ins := false$; $G := \Omega$; **for** x **in** F **do** $(G,\ ins) := body(G,\ ins,\ x)$ **end**; **if** $\neg ins$ **then** $G := con(G,\ w)$ $\{Q\}$

where ins is a Boolean variable,

$$body(G, ins, x) = \textbf{if } w \leq x \wedge \neg ins \textbf{ then } (con(G, w, x), true) \textbf{ else } (con(G, x), ins),$$

$P(F) = ord(F)$, $Q(F,\ G) = (del(G,\ w) = F \wedge ord(G) \wedge w \in memb(G))$.

Two following verification conditions are generated by means of the proof rule rl2 and the standard system PROOF. We consider $rep(F)$ to be a shorthand for $rep((\Omega,\ false),\ F,\ body)$.

VC1: $P(F) \wedge \neg rep_{ins}(F) \rightarrow Q(F,\ con(rep_G(F),\ w))$,
VC2: $P(F) \wedge rep_{ins}(F) \rightarrow Q(F,\ rep_G(F))$.

At first, we will prove the property $prop(rep(F)) = prop1 \wedge prop2$ where

$$\begin{aligned} prop1 &= (\neg rep_{ins}(F) \rightarrow rep_G(F) = F \wedge w > F), \\ prop2 &= (rep_{ins}(F) \rightarrow del(rep_G(F),\ w) = F \wedge w > hd(rep_G(F),\ w) \wedge \\ &\quad w \in memb(rep_G(F)) \wedge w \leq choo(tl(rep_G(F),\ w))). \end{aligned}$$

The property $prop1$ specifies the case when the variable ins remains false, w exceeds all elements of the file F, and F is copied to the file G. The property $prop2$ specifies another case when the variable ins becomes true and the file F is copied to the file G with insertion of the element w in its proper place.

We apply Corollary 2 in order to prove the property $prop(rep(F))$. If $empty(F)$, then the condition $prop(rep(F) \leftarrow (\Omega,\ false)) = (\Omega = F \wedge w > F)$ is obviously true. Suppose that $\neg empty(F) \wedge prop(rep(head(F)))$. The property $prop(rep(F) \leftarrow body(rep(head(F)),\ last(F)))$ is proved by case analysis. Let us consider the most complicated case in which we prove the property

$$prop2(rep(F) \leftarrow body(rep(head(F)),\ last(F))) \tag{9}$$

provided $rep_{ins}(head(F))$. Property (9) is equivalent to

$$\begin{aligned} &body_{ins}(rep(head(F)),\ last(F)) \rightarrow del(G',\ w) \\ &= F \wedge w > hd(G',\ w) \wedge w \in memb(G') \wedge w \leq choo(tl(G',\ w)) \end{aligned} \tag{10}$$

where $G' = body_G(rep(head(F)),\ last(F))$. By the definition of the body,

$$body_{ins}(rep(head(F)),\ last(f)) = rep_{ins}(head(F))$$

and

$$G' = con(rep_G(head(f)),\ last(F)).$$

It follows from $prop2(rep(head(F)))$ that

$$\begin{aligned} &del(rep_G(head(F)),\ w) = head(F) \wedge w > hd(rep_G(head(F)),\ w) \wedge \\ &\quad w \in memb(rep_G(head(F))) \wedge w \leq choo(tl(rep_G(head(F)),\ w)). \end{aligned} \tag{11}$$

It follows from this and Theorem Th3 that

$$\begin{aligned} del(G',\ w) &= del(con(rep_G(head(F)),\ last(F)),\ w) \\ &= con(del(rep_G(head(F)),\ w),\ last(F)) \\ &= con(head(F),\ last(F)) = F. \end{aligned}$$

It follows from (11) that

$$hd(G',\ w) = hd(con(rep_G(head(F)), last(F)),\ w) = hd(rep_G(head(F)),\ w),$$

hence $w > hd(G',\ w)$.

It follows from (11) that

$$w \leq choo(tl(rep_G(head(F)),\ w)),$$

therefore $\neg empty(tl(rep_G(head(F)),\ w))$ and

$$choo(tl(G',\ w)) = choo(tl(rep_G(head(F)),\ w)).$$

So, condition (10) is true.

To prove the verification conditions VC1 and VC2, we apply the property $prop(rep(F))$. The conclusion of the condition VC1 is equivalent to

$$del(con(F,\ w),\ w) = F\ \wedge ord(con(F,\ w)) \wedge w \in memb(con(F,\ w)). \quad (12)$$

Condition (12) is evident from $P(F)$ and $prop1(rep(F))$. The conclusion of the condition VC2 is equivalent to

$$del(rep_G(F),\ w) = F \wedge ord(rep_G(F)) \wedge w \in memb(rep_G(F)). \quad (13)$$

It remains to show that $ord(rep_G(F))$ since the rest terms of condition (13) are evident from $prop2(rep(F))$. It follows from $w \in memb(rep_G(F))$ that

$$rep_G(F) = con(hd(rep_G(F),\ w),\ w,\ tl(rep_G(F),\ w)).$$

It follows from $ord(F)$ and $del(rep_G(F),\ w) = F$ that $ord(hd(rep_G(F),\ w))$ and $ord(tl(rep_G(F),\ w))$. So, by the property $prop2(rep(F))$, if follows from $w \leq choo(tl(rep_G(F),\ w))$ and $w > hd(rep_G(F),\ w)$ that $ord(rep_G(F))$.

Note that in [19] a mistake has been found in a version of the program with the help of the functional method. Formal verification of the correct program is not described in [19].

7. Case of study: arrays in loop bodies

At first, we recall the known notion $upd(A, ind, exp)$ which denotes an array resulted from the array A by replacing its element indexed by ind with the value of the expression exp. A notion $upd(A,\ IND,\ EXP)$ where $IND = [ind_1,\ \ldots,\ ind_m]$, $EXP = [exp_1,\ \ldots,\ exp_m]$ is its generalization. The notion denotes an array obtained from the array A by the sequential replacement of its ind_j-th element with the value of the expression exp_j for all $j = 1,\ \ldots,\ m$. The following axioms define this notion.

Ax10. $upd(A,\ IND,\ EXP)[ind] = A[ind]$ provided $ind \notin IND$.

Ax11. For all $j = 1, \ldots, m$ $upd(A, IND, EXP)[ind_j] = exp_j$ provided $\forall k\ (j < k \leq m \rightarrow ind_k \neq ind_j)$.

In the rest of this paper we assume that the iteration body contains a vector of variables consisted of a variable x, an array A and a vector v of other variables. So, iteration (1) have the form

$$\textbf{for } x \textbf{ in } S \textbf{ do } (A, v) := body(A, v, x) \textbf{ end}.$$

We also assume that $body_A(A, v, x)$ can be represented by $upd(A, IND, EXP)$ for appropriate vectors $IND(x)$ and $EXP(A, v, x)$ where if $A[ind]$ is in $exp_j(A, v, x)(1 \leq j \leq m)$, then ind has the form $r_i(x)(1 \leq i \leq t)$. So, we impose a restriction on IND and EXP such that ind_j and r_i do not depend on variables from v. Notice that the representation of $body_A$ by upd is natural, since such a loop body usually contains the statements of the form $A[ind] := exp$ which can be jointly represented by the statement $A := upd(A, IND, EXP)$.

To express the effect of iteration (14) in a special case, we will define a parallel replacement operation $rep(\tilde{A}, v, S, body)$ with respect to the array A as a special case of the replacement operation for which the reasoning technique can be simplifyed. The operation $rep(\tilde{A}, v, S, body)$ is defined to be a pair (A_n, v_n) such that $A_0 = A$, $v_0 = v$; $n = 0$ provided $empty(S)$; $A_j = upd(A_{j-1}, IND(s_j), EXP(A, v_{j-1}, s_j))$, $v_j = body_v(A_{j-1}, v_{j-1}, s_j)$ for all $j = 1, \ldots, n$ provided $\neg empty(S)$ and $vec(S) = [s_1, \ldots, s_n]$. Thus, the definition differs from the replacement operation definition (see Section 4) in that EXP included in upd depends on the initial value of the array A. The following axioms define the parallel replacement operation.

Ax12. $empty(S) \rightarrow rep(\tilde{A}, v, S, body) = (A, v)$.

Ax13. $\neg empty(S) \rightarrow rep_A(\tilde{A}, v, S, body) = upd(rep_A(\tilde{A}, v, head(S), body), IND(last(S)), EXP(A, rep_v(\tilde{A}, v, head(S), body), last(S)))$.

Ax14. $\neg\ empty(S) \rightarrow rep_v(\tilde{A}, v, S, body) = body_v(rep(\tilde{A}, v, head(S), body), last(S))$.

The parallel replacement operation is correct, if it coincides with the replacement operation. A useful sufficient condition of correctness of the parallel replacement operation gives the following theorem where $IND(T) = \{ind(s) \mid s \in T,\ ind \in IND\}$.

Th8. $rep(\tilde{A}, v, S, body) = rep(A, v, S, body)$, if for every $j = 2, \ldots, n$ and $i = 1, \ldots, t$, $r_i(s_j) \notin IND(pred(s_j))$.

Notice that the condition of the theorem holds for $j = 1$ because $IND(pred(s_1))$ is the empty set.

8. Computation of parallel replacement operation

If Th8 holds, the replacement operation can be replaced by the parallel replacement operation with respect to an array. To compute $rep_A(\tilde{A}, v, S, body)$, a recursive procedure can be used. The procedure given by axiom Ax13 reduces the operation for the structure S to the same operation for the structure $head(S)$. In this section we present another procedure to compute the parallel replacement operation which is simpler than the recursive one and shows advantages of the operation.

We introduce the following notation. A set $IND(S) = \{ind_j(s) \mid s \in memb(S),\ 1 \le j \le m\}$ is called a replacement domain. The set $IND(S)$ is empty if $empty(S)$. Let us define a maximal occurrence function $moc(S, k)$. The function for an element k of the replacement domain $IND(S)$ returns an element s of S generating k and a number j of a suitable component ind_j. So, $ind_j(s) = k$. The function $moc(S, k)$ will be undefined for $k \notin IND(S)$. So, the function $moc(S, k)$ will be undefined for every k, if $empty(S)$. If the element k is generated by different elements of the structure S, then the maximal element from these elements is choosen. Next, the maximal number of the component of IND which generates k is selected.

The following theorem gives a procedure for computing the parallel replacement operation with respect to an array A.

Th9. $rep_A(\tilde{A}, v, S, body)[k] = A[k]$ if $k \notin IND(S)$.
$rep_A(\tilde{A}, v, S, body)[k] = exp_j(A, rep_v(\tilde{A}, v, head^{n-i+1}(S), body), s_i)$ if $k \in IND(S)$, $vec(S) = [s_1, \ldots, s_n]$ and $moc(S, k) = (s_i, j)$.

Notice that

$$vec(head^{n-i}(S)) = [s_1, \ldots, s_i],$$

therefore

$$\neg empty(head^{n-i}(S))$$

and

$$head^{n-i+1}(S)$$

will be defined.

We consider the following example to illustrate the use of theorem Th9.

Example 3. The array inversion.

The following annotated program inverts an array $A[1..n]$.
$\{P\}$ **for** k **in** S **do** $(A[k],\ A[n+1-k]) := (A[n+1-k],\ A[k])$ **end** $\{Q\}$
where $S = [1, 2, \ldots, trunc(n \setminus 2)]$, $trunc(s)$ is an integer nearest to s, A_0 is an initial value of the array A, $P(A) = n \ge 1 \wedge A[1..n] = A_0[1..n]$,
$Q(A) = \forall i(1 \le i \le n \to A[i] = A_0[n+1-i])$.

The loop body can be represented in the form

$$A := upd(A,\ IND,\ EXP)$$

where

$$\begin{aligned} IND &= (ind_1,\ ind_2), \\ ind_1(k) &= k, \\ ind_2(A,\ k) &= n+1-k, \\ EXP &= (exp_1,\ exp_2), \\ exp_1(A,\ k) &= A[n+1-k], \\ exp_2(A,\ k) &= A[k]. \end{aligned}$$

Therefore,

$$\begin{aligned} exp_1(A,\ k) &= A[r_1(k)], \\ r_1(k) &= n+1-k, \\ exp_2(A,\ k) &= A[r_2(k)], \\ r_2(k) &= k. \end{aligned}$$

To prove the condition of Theorem Th8, at first we fix an integer s such that $s \in S$ and $s > 1$. Then

$$\begin{aligned} pred(s) &= \{1, \ldots, s-1\}, \\ IND(pred(s)) &= \{1,\ \ldots,\ s-1,\ n-s+2,\ \ldots,\ n\}. \end{aligned}$$

It follows from $2s \leq n$ that

$$s < n-s+2,\ r_1(s) = n+1-s \notin IND(pred(s))$$

and

$$r_2(s) = s \notin IND(pred(s)).$$

Therefore, by theorem Th8, $rep(\tilde{A}, S,\ body) = rep(A,\ S,\ body)$.

The verification condition $P(A) \rightarrow Q(rep(\tilde{A},\ S,\ body))$ is generated by the proof rule rl2 when the program *prog* is empty. The conclusion of the condition has the form

$$\forall i(1 \leq i \leq n \rightarrow rep(\tilde{A},\ S,\ body)[i] = A_0[n+1-i]). \qquad (14)$$

To prove (15), we fix an integer $i (1 \leq i \leq n)$ and consider three cases:

1) $i \in S$. Then $i \in IND(S)$, $moc(S,\ i) = (i,\ 1)$ since i is only generated by $ind_1(k)$. By theorem Th9, $rep(\tilde{A},\ S,\ body)[i] = A[n+1-i]$.

$A[n+1-i] = A_0[n+1-i]$ follows from $P(A)$.

2) $i \notin S$ and $i \in IND(S)$. Then $moc(S, i) = (n+1-i, 2)$ since i is only generated by $ind_2(k)$. By theorem Th9, $rep(\tilde{A},\ S,\ body)[i] = A[n+1-i]$ as in the first case.

3) $i \notin IND(S)$. Notice that n is an odd number. Otherwise, there exists $t \geq 1$ such that $n = 2t$. Then $S = [1, \ldots, t]$, $IND(S) = \{1, \ldots, t, t+1, \ldots, 2t\}$. Therefore, we have a contradiction with $1 \leq i \leq n$. So, there exists $t \geq 1$ such that $n = 2t - 1$. Then $trunc(n \setminus 2) = t - 1$, $IND(S) = \{1, \ldots, t-1, t+1, \ldots, 2t-1\}$. Therefore, $i = t$ and, by theorem Th9, $rep(\tilde{A}, S, body)[i] = A[t]$. It remains to notice that $n + 1 - i = t$ and $A[t] = A_0[t]$ follows from $P(A)$.

It should be noted that the proof of correctness of the array inversion program with the help of theorem Th9 can be realized without induction. The structure S can be a set $\{1, 2, \ldots, trunc(n \setminus 2)\}$ in the program, since the result of the parallel replacement operation does not depend on the order of elements of S.

9. Conclusion

The paper presents a new symbolic method for verification of definite iteration over different data structures. The symbolic method differing substantially from the axiomatic and functional methods has some features related with these methods.

For definite iteration the symbolic method uses a proof rule which has the form inherent in the axiomatic method, however, without invariants. To justify the proof rule, the axiomatic method is applied. It might be good to combine the axiomatic and symbolic methods, especially for programs which include inner and outer loops. It will be better for inner loops to use the symbolic method because the replacement operation is simpler for them than for outer ones. For outer loops the axiomatic method is better since invariants of the loops are simpler than invariants of inner ones.

The functional method in a semiformal form has been used in industrial projects as a part of the Cleanroom method [5, 11]. Problems of the application of the functional method to arrays have been noted in [10]. The symbolic method, like the functional one, makes use of a functional representation for the iteration body and for the iteration as the replacement operation. Therefore, the resemblance of Axioms Ax8 and Ax9 for the replacement operation to theorem 2 [19] can be noted.

A technique of proving verification conditions which include the replacement operation is of considerable importance in the symbolic method. The technique is based on backward and forward induction principles which are rather flexible because they allows one to vary the property *prop* from postconditions(as in example 1) to taking into account other relations between variables (as in example 2). For arrays the change of the replacement operation by the parallel one plays a large part in the proof technique, since it allows us to simplify inductive reasoning. Notice that the parallel replacement operation has been introduced for a special case of arrays in [13] and has been generalized for them in [15, 18]. A variant of the parallel replacement operation has been used for modelling synchronous computations in [3] which are represented by statements equivalent to for-loops over sets with vector assignments as the loop bodies. The statements are expressed by universal quantifiers bounded by sets which are given by Boolean expressions.

Both of the axiomatic and symbolic methods are used for automatic program verification by means of the problem-oriented system SPECTRUM [16, 17]. The parallel replacement operation is applied to Pascal for-loops with a vector assignment to array elements as the loop body which are contained in linear algebra programs [16].

The symbolic method of verification is promising for applications. To extend the range of its applications in program verification systems, it is helpful to spread the parallel replacement operation to data structures such as files, records, and pointers. It is of interest to extend the symbolic method to definite iterations with the bodies which include the goto-statement. The method for such iterations is considered in a special case in [15]. For applications of the symbolic method it is helpful to develop a proof technique which uses the specific features of problem domains. The induction principles developed in framework of the symbolic method can be also useful for the functional method.

References

[1] S.K. Basu, J. Misra, *Proving loop programs*, in IEEE Trans. on Software Engineering, Vol. 1, No. 1, 1975, 76–86.

[2] S.K. Basu, J. Misra, *Some classes of naturally provable programs*, in Proc. 2nd Int. Conf. on Software Engineering, IEEE Press, 1976, 400–406.

[3] K.M. Chandy, J. Misra, *Parallel Program Design*, Addison-Wesley, 1988.

[4] D.D. Dunlop, V.R. Basili, *Generalizing specifications for uniformly implemented loops*, in ACM Trans. on Programming Languages and Systems, Vol. 7, No. 1, 1985, 137–158.

[5] M. Dyer, *Designing software for provable correctness*, in Information and Software Technology, Vol. 30, No. 6, 1988.

[6] D. Gries, N. Gehani, *Some ideas on data types in high-level languages*, in Communications of the ACM, Vol. 20, No. 6, 1977, 414–420.

[7] C.A.R. Hoare, *An axiomatic basis of computer programming*, in Communications of the ACM, Vol. 12, No. 10, 1969, 576–580.

[8] C.A.R. Hoare, *A note on the for statement*, in BIT, Vol. 12, No. 3, 1972, 334–341.

[9] R.C. Linger, H.D. Mills. B.I. Witt, *Structured Programming: Theory. And Practice*, Addison-Wesley, 1979.

[10] H.D. Mills, *Structured programming: retrospect and prospect*, in IEEE Software, Vol. 3, No. 6, 1986, 58–67.

[11] H.D. Mills, M. Dyer, R.C. Linger, *Cleanroom software engineering*, in IEEE Software, Vol. 4, No. 5, 1987, 19–24.

[12] J. Misra, *Some aspects of the verification of loop computations*, in IEEE Trans. on Software Engineering, Vol. 4, No. 6, 1978, 478–486.

[13] V. A. Nepomniaschy, *Proving correctness of linear algebra programs*, in Programming, Vol. 4, 1982, 63–72 (in Russian).

[14] V.A. Nepomniaschy, *Loop invariant elimination in program verification*, in Programming, Vol. 3, 1985, 3–13 (in Russian).

[15] V.A. Nepomniaschy, *On problem-oriented program verification*, in Programming, Vol. 1, 1986, 3–13 (in Russian).

[16] V.A. Nepomniaschy, A.A. Sulimov, *A problem-oriented verification system and its application to linear algebra programs*, in Theoretical Computer Science, Vol. 119, 1993, 173–185.

[17] V.A. Nepomniaschy, A.A. Sulimov, *Problem-oriented means of program specification and verification in project SPECTRUM*, in Proc. Intern. Symp. on Design and Implementation of Symbolic Computation Systems (DISCO 93), Austria, Sept. 1993, Lecture Notes in Computer Science, Vol. 722, 1993, 374–378.

[18] V.A. Nepomniaschy, *Array program verification*, in System Informatics, Novosibirsk, Nauka, 1993, 68–98.

[19] A.M. Stavely, *Verifying definite iteration over data structures*, in IEEE Trans. on Software Engineering, Vol. 21, No. 6, 1995, 506–514.

Appendix. Proofs of the theorems.

Th1. $\neg empty(S) \rightarrow con(choo(S),\ rest(S)) = S.$

Proof. From $\neg empty(S)$ and axioms Ax1, Ax4, Ax5 it follows that

$$choo(con(choo(S),\ rest(S))) = choo(S)$$

and

$$rest(con(choo(S),\ rest(S))) = rest(S).$$

So, theorem Th1 is evident from this.

Th2. $con(vec(S_1),\ vec(S_2)) = vec(con(S_1,\ S_2)).$

Proof. We will use the induction by $|memb(S_1)| = n$. If $n = 0$, then $empty(S_1)$. From this and axioms Ax2, Ax4, it follows that

$$con(vec(S_1),\ vec(S_2)) = vec(S_2) = vec(con(S_1,\ S_2)).$$

Let us consider the case $n \neq 0$. So, $\neg empty(S_1)$. We will use the following inductive hypothesis.

$$con(vec(rest(S_1)),\ vec(S_2)) = vec(con(rest(S_1),\ S_2)). \quad (1)$$

From axioms Ax3 and Ax5, it follows that

$$choo(con(vec(S_1),\ vec(S_2)) = choo(S_1) = choo(vec(con(S_1,\ S_2))).$$

From axioms Ax3, Ax5 and (1), it follows that

$$\begin{aligned} rest(con(vec(S_1), vec(S_2))) &= vec(con(rest(S_1), S_2)) \\ &= rest(vec(con(S_1, S_2))). \end{aligned}$$

Th3. $\neg empty(S) \rightarrow con(head(S),\ last(S)) = S.$

Proof. We will use the induction by $|head(S)| = n$.

If $n = 0$, then $empty(head(S))$. From this, $\neg empty(S)$ and axiom Ax6, it follows $|memb(S)| = 1$ and $S = str(last(S))$. By axiom Ax4, we have

$$con(head(S),\ last(S)) = str(last(S)).$$

Let us consider the case $n \neq 0$. So, $\neg empty(head(S))$. We will use the following inductive hypothesis

$$con(head(rest(S)),\ last(rest(S))) = rest(S). \quad (2)$$

From axioms Ax5 and Ax7, it follows that

$$choo(con(head(S),\ last(S))) = choo(head(S)) = choo(S)).$$

From (2) and axioms Ax5 and Ax7, it follows that

$$\begin{aligned} rest(con(head(S),\ last(S))) &= con(rest(head(S)), last(S)) \\ &= con(head(rest(S)), last(rest(S))) \\ &= rest(S). \end{aligned}$$

Th4. The proof rule rl1 is derived in the standard system PROOF.

Proof. It should be noted that the iteration

for x **in** S **do** $v := body(v,\ x)$ **end**

is equivalent to the following program *prog*1

```
if ¬empty(S) then
begin x := choo(S);  while x ≠ last(S) do
                     begin v := body(v, x); x := next(S, x) end
      v := body(v, x)
end.
```

If $empty(S)$, then the proof rule rl1 has the form

$$cond1 \wedge cond3 \vdash \{P\}\ prog\ \{inv\}\{Q\}$$

where $cond1 : \{P\}\ prog\ \{inv\}$, $cond3 : inv \rightarrow Q$, since the condition *cond*2 is true because $x \notin memb(S)$. So, the rule rl1 is derived in the system PROOF.

Let us consider the case $\neg empty(S)$. The annotated program

$$\{P\}\ prog;\ prog1\ \{Q\}$$

equivalent to the conclusion of the rule rl1 is denoted by *prog*2. The following assertions are generated by means of the proof rule from the system PROOF for the while-loop with the invariant $inv \wedge x \in memb(S)$ which is applied to the program *prog*2.

$$\{P\}\ prog;\ x := choo(S)\{inv \wedge x \in memb(S)\}, \tag{3}$$

$$\begin{aligned} &\{inv \wedge x \in memb(S) \wedge x \neq last(S)\}\ v := body(v,\ x); \\ &x := next(S,\ x)\{inv \wedge x \in memb(S)\}, \end{aligned} \tag{4}$$

$$\{inv \wedge x \in memb(S) \wedge x = last(S)\}\ v := body(v,\ x)\{Q\}. \tag{5}$$

It remains to derive the assertions from the conditions *cond*1 – *cond*3 of the rule rl1. The assertion (3) is evident from *cond*1. The assertions (4) and (5) are evident from *cond*2.

Th5. $rep(v,\ con(S_1,\ S_2),\ body) = rep(rep(v,\ S_1,\ body),\ S_2,\ body).$

Proof. Let us consider a generalization of Theorem Th5 of the form

$$\forall v\ rep(v,\ con(S_1,\ S_2),\ body) = rep(rep(v,\ S_1,\ body),\ S_2,\ body). \qquad (6)$$

We will use the induction by $|memb(S_1)| = n$ to prove (6).

If $n = 0$, then $empty(S_1)$. From this and axioms Ax4, Ax8 it follows (6).

Let us consider the case $n \neq 0$.

So, $\neg empty(S_1)$ and $\neg empty(con(S_1, S_2))$. From axiom Ax5, it follows that

$$choo(con(S_1,\ S_2)) = choo(S_1)$$

and

$$rest(con(S_1,\ S_2)) = con(rest(S_1),\ S_2).$$

From this and axiom Ax9, it follows that

$$\begin{aligned} &rep(v,\ con(S_1,\ S_2),\ body) \\ &= rep(body(v,\ choo(S_1)),\ con(rest(S_1),\ S_2),\ body). \end{aligned} \qquad (7)$$

An inductive hypothesis is obtained by the substitution of $rest(S_1)$ for all occurrences of S_1 in (6). From (7), the inductive hypothesis and axiom Ax9, (6) is valid.

Th6. $\neg empty(S) \rightarrow rep(v,\ S,\ body)$
$= body(rep(v,\ head(S),\ body),\ last(S))$.

Proof. From $\neg empty(S)$ and theorems Th3, Th5, it follows that

$$\begin{aligned} rep(v,\ S,\ body) &= rep(v,\ con(head(S),\ last(S)),\ body) \\ &= rep(rep(v,\ head(S),\ body),\ str(last(S)),\ body). \end{aligned}$$

From axiom Ax1, it follows that

$$\neg empty(str(last(S))),\ choo(str(last(S))) = last(S)$$

and $empty(rest(str(last(S))))$. It remains to apply axioms Ax9 and Ax8.

Th7. The proof rule rl2 is derived in the standard system PROOF.

Proof. For a structure S and a variable x such that

$$\neg empty(S),\ x \in memb(S),\ vec(S) = [s_1,\ \ldots,\ s_n],$$

the following function is defined. S_x is a function which returns a vector $[x_1,\ \ldots,\ x_m]$ such that $x_1 = x$, $x_m = last(S)$ and $x_{i+1} = next(S,\ x_i)$ for $i = 1,\ \ldots,\ m-1$. If $empty(S)$, then the function S_x returns such a vector that $empty(S_x)$. The function S_x will be undefined if and only if $x \notin memb(S)$ and $\neg empty(S)$.

Let us assume that the following premise of the rule rl2 is satisfied.

$$\{P\}\ prog\ \{Q(v \leftarrow rep(v,\ S,\ body))\}. \tag{8}$$

To prove the conclusion of the rule rl2, we apply the rule rl1 justified by theorem Th4 to it with $inv = Q(v \leftarrow rep(v,\ S_x,\ body))$. The conditions $cond1 - cond3$ from the premise of the rule rl1 are proved by the case analysis.

If $empty(S)$, then $empty(S_x)$. By axiom Ax8,

$$rep(v,\ S,\ body) = rep(v,\ S_x,\ body) = v$$

and $inv = Q$. So, from this and (8) it follows that the conditions $cond1 - cond3$ hold.

Let us consider the case $\neg empty(S)$. It should be noted that

$$inv(x \leftarrow choo(S)) = Q(v \leftarrow rep(v,\ S,\ body))$$

because $S_x = S$ for $x = choo(S)$. So, $cond1$ follows from (8). If $x \in memb(S)$ and $x \neq last(S)$, then

$$\begin{aligned} & inv(x \leftarrow next(S,\ x),\ v \leftarrow body(v,\ x)) \\ & = Q(v \leftarrow rep(body(v;\ x),\ rest(S_x),\ body)) \\ & = inv \end{aligned}$$

because

$$x = choo(S_x),\ S_x(x \leftarrow next(S,\ x)) = rest(S_x)$$

and we use axiom Ax9. If $x = last(S)$, then

$$rep(v,\ S_x,\ body) = body(v,\ x)$$

and

$$Q(v \leftarrow body(v,\ x)) = inv$$

because $empty(rest(S_x))$ and axioms Ax8, Ax9 are used. So, the condition $cond2$ is derived in the system PROOF. The condition $cond3$ is evident.

Th8. $rep(\tilde{A},\ v,\ S,\ body) = rep(A,\ v,\ S,\ body)$, if for every $j = 2, \ldots, n$ and $i = 1, \ldots, t$, $r_i(s_j) \notin IND(pred(s_j))$.

Proof. It is sufficient to prove

$$EXP(A_{j-1},\ v_{j-1},\ s_j) = EXP(A,\ v_{j-1},\ s_j) \text{ for all } j = 1, \ldots, n. \tag{9}$$

Assertion (9) follows from

$$A_{j-1}[r_i(s_j)] = A[r_i(s_j)] \text{ for all } j = 1, \ldots, n \text{ and } i = 1, \ldots, t. \tag{10}$$

Let us consider a generalization of (10) of the form

$$A_k[r_i(s_j)] = A[r_i(s_j)] \text{ for all } j = 1, \ldots, n, \quad i = 1, \ldots, t, \quad k = 0, 1, \ldots, j-1. \tag{11}$$

To prove (11) we will use the induction by k.

If $k = 0$, then (11) is true because $A_0 = A$.

If $0 < k < j$, then

$$A_k[r_i(s_j)] = upd(A_{k-1}, IND(s_k), EXP(A, v_{k-1}, s_k))[r_i(s_j)] = A_{k-1}[r_i(s_j)]$$

because $s_k \in pred(s_j)$, $IND(s_k) \subseteq IND(pred(s_j))$, $r_i(s_j) \notin IND(s_k)$ and axiom Ax10 is used. It remains to apply the inductive hypothesis $A_{k-1}[r_i(s_j)] = A[r_i(s_j)]$.

Th9. $rep_A(\tilde{A}, v, S, body)[k] = A[k]$, if $k \notin IND(S)$;
$rep_A(\tilde{A}, v, S, body)[k] = exp_j(A, rep_v(\tilde{A}, v, head^{n-i+1}(S), body), s_i)$,
if $k \in IND(S)$, $vec(S) = [s_1, \ldots, s_n]$ and $moc(S, k) = (s_i, j)$.

Proof. We will use the induction by $|memb(S)| = n$.

If $n = 0$, then $empty(S)$ and $IND(S)$ is the empty set. From this and Ax12, it follows that $k \notin IND(S)$ and theorem 9.

Let us consider the case $n \neq 0$. So, $\neg empty(S)$. By theorem Th3,

$$S = con(head(S), last(S))$$

and

$$IND(S) = IND(head(S)) \cup IND(last(S)).$$

From axiom Ax13, it follows that

$$\begin{aligned} &rep_A(\tilde{A}, v, S, body)[k] = upd(rep_A(\tilde{A}, v, head(S), body), \\ &IND(last(S)), EXP(A, rep_v(\tilde{A}, v, head(S), body), last(S)))[k]. \end{aligned} \tag{12}$$

theorem Th9 is proved by the case analysis. Three cases are possible.

1. $k \notin IND(S)$. Then $k \notin IND(last(S))$. From this, (12) and axiom Ax10, it follows that

$$rep_A(\tilde{A}, v, S, body)[k] = rep_A(\tilde{A}, v, head(S), body)[k]. \tag{13}$$

It remains to apply the inductive hypothesis because

$$|memb(head(S))| = n - 1 \text{ and } k \notin IND(head(S)).$$

2. $k \in IND(last(S))$. Then there exists j such that $1 \leq j \leq m$, $ind_j(s_n) = k$, and $\forall l(m \geq l > j \rightarrow ind_l(s_n) \neq k)$. From this, (12) and Ax11, it follows the conclusion of theorem Th9 for $i = n$ of the form

$$rep_A(\tilde{A},\ v,\ S,\ body)[k] = exp_j(A,\ rep_v(\tilde{A},\ v,\ head(S),\ body),\ s_n).$$

3. $k \in IND(head(S)) \wedge k \notin IND(last(S))$. Then from (12) and axiom Ax10 it follows (13). It should be noted that $moc(S,\ k) = moc(head(S),\ k)$, because $k \notin IND(last(S))$. Let us assume $moc(head(S),\ k) = (s_i,\ j)$ where $1 \le i < n,\ 1 \le j \le m$. It remains to apply the inductive hypothesis of the form

$$\begin{aligned} &rep_A(\tilde{A}, v, head(S), body)[k] \\ &= exp_j(A, rep_v(\tilde{A}, v, head^{n-i}(head(S)), body), s_i) \end{aligned}$$

because $|memb(head(S))| = n-1$. Theorem Th9 follows from this and (13).