Joint NCC & IIS Bull., Comp. Science, 15 (2001), 103–123

Symbolic verification method for definite iterations over tuples of data structures*

V. A. Nepomniaschy

The symbolic method for verifying definite iterations over hierarchical data structures [12] is extended to allow tuples of data structures and exit from the iteration body under a condition. Transformations of these generalized iterations to the standard ones are proposed and justified. Useful properties of the transformations are given. A technique for generating verification conditions and their proving, including induction principles, is described. For iterations over files, a problem-oriented proving technique is presented too. Examples which illustrate application of the symbolic method to file program verification are considered.

1. Introduction

The axiomatic and functional styles of program verification include the following stages: program annotation through the construction of pre -, post-conditions and loop invariants or functions expressing the loop effect; deriving verification conditions with the help of proof rules and proving them [5, 7]. In both approaches, the loop annotation remains a difficult problem especially for programs over complex data structures [8, 13].

A natural method to attack the problem is to use simple loops [1], i. e., definite iterations similar to for-loops. Although the reduction of for-loops to while-loops is often used for verification, attempts to use the specific character of for-loops in the framework of the axiomatic approach should be noted [2–4, 6]. In the framework of the functional approach, a general form of a definite iteration as an iteration over all elements of a structure, such as a list, set, file, array, tree, has been proposed in [14].

A symbolic method of verifying for-loops, that had the statement of assignment to array elements as the loop body, has been proposed in [9, 10]. This method is based on inserting a replacement operation into the annotation language. The replacement operation represents the loop effect in a symbolic form and allows us to formulate a proof rule for these loops without invariants. Using the proof rule, verification conditions including the replacement operation are generated. To prove the verification conditions, a special technique which uses theorems about properties of the replacement operation is developed. In [11], we have extended the symbolic method to the

*Supported by the Russian Foundation for Basic Research under Grant 00-01-00909.

definite iteration over data structures without restrictions on the loop bodies. The symbolic method has been developed for the definite iteration over hierarchical data structures in [12]. The symbolic method is more appropriate for so-called flat loops [1] which have no embedded loops. Moreover, definite iterations from [14] do not accept exit from the iteration body. This raises the problem how to extend the area of application of the symbolic method. One approach to tackling this problem is extending the notion of flat loops and accepting exit from the loop body under a condition.

The purpose of this paper is to introduce a new statement, namely, a definite iteration over tuples of data structures, which accepts exit from the iteration body, and to extend the symbolic method to the statement. The definite iteration over hierarchical data structures [12] is described in Section 2. In Section 3, the iteration over tuples of data structures is defined, and theorems about its reduction to the iteration over hierarchical data structures and about properties of these structures are proved. A reduction of an iteration over tuples of data structures with the exit statement to the iteration described in Section 2 is justified in Section 4. A replacement operation and its properties are given in Section 5, where a technique for deriving and proving verification conditions is considered. Definite iterations over files are described as a case of study in Section 6 along with recursive procedures for computing the replacement operation. Verification of two file programs which perform merging (with cleaning) and intersection of ordered files, is exemplified in Section 7. Results and prospects of the symbolic verification method are discussed in Section 8.

2. Iteration over hierarchical data structures

We introduce the following notation. Let $\{s_1, \ldots, s_n\}$ be a multiset consisting of elements $s_1, \ldots, s_n$, $U_1 - U_2$ be the difference between multisets U_1 and U_2, $U_1 \bigcup U_2$ be the union of multisets, and $|U|$ be the power of a finite multiset U. Let $[v_1, \ldots, v_m]$ denote a vector consisting of elements $v_i (1 \leq i \leq m)$. A concatenation operation $con(V_1, V_2)$ is defined in the usual fashion for vectors V_1 and V_2.

Let us remind the notion of a data structure [14]. Let $memb(S)$ be a finite multiset of elements of a structure S, $empty(S)$ be a predicate "$memb(S)$ is empty", $choo(S)$ be a function which returns an element of $memb(S)$, $rest(S)$ be a function which returns a structure S' such that $memb(S') = memb(S) - \{choo(S)\}$. The functions $choo(S)$ and $rest(S)$ will be undefined if and only if $empty(S)$. This definition, abstracting from the way of determination of the functions $choo(S)$ and $rest(S)$, is quite flexible.

For example, if a tree is defined as a data structure, a tree traversal method is fixed. So, such different methods result in different data structures.

Let us remind the definition of useful functions related to the structure S [11]. Let $vec(S)$ denote a vector $[s_1, \ldots, s_n]$ such that $s_i = choo(rest^{i-1}(S))$ $(i = 1, \ldots, n)$ in the case of $\neg empty(S)$ and $memb(S) = \{s_1, \ldots, s_n\}$. The vector $vec(S)$ is empty if $empty(S)$. Structures S_1 and S_2 are called equivalent when $vec(S_1) = vec(S_2)$. The functions $head(S)$ and $last(S)$ will be undefined in the case of $empty(S)$. A function $head(S)$ returns a structure such that $vec(head(S)) = [s_1, \ldots, s_{n-1}]$, if $vec(S) = [s_1, \ldots, s_n]$ and $n \geq 2$. If $n = 1$, then $empty(head(S))$. Let $last(S)$ be a partial function such that $last(S) = s_n$, if $vec(S) = [s_1, \ldots, s_n]$. Let $str(s)$ denote a structure S that contains the only element s. A concatenation operation $con(S_1, S_2)$ is defined in [11] so that $con(vec(S_1), vec(S_2)) = vec(con(S_1, S_2))$. Let $con(s, S) = con(str(s), S)$ and $con(S, s) = con(S, str(s))$. In the case of $\neg empty(S)$, $con(choo(S), rest(S)) = con(head(S), last(S)) = S$ [11]. Moreover, the property $head(rest(S)) = rest(head(S))$ provided $\neg empty(rest(S))$ results from [11].

Our aim is to introduce parameters in the definition of the structure S. To this end, we determine rules for construction of a hierarchical structure S from the given structures $S_1, \ldots, S_m$. We will use $T(S_1, \ldots, S_m)$ to denote a term constructed from data structures S_i $(i = 1, \ldots, m)$ with the help of the functions *choo*, *last*, *rest*, *head*, *str*, *con*. For a term T which represents a data structure, we denote the function $|memb(T)|$ by $lng(T)$. The function can be calculated by the following rules:

$$\begin{array}{lcl} lng(S_i) & = & |memb(S_i)|, \\ lng(con(T_1, T_2)) & = & lng(T_1) + lng(T_2), \\ lng(rest(T)) & = & lng(head(T)) \quad = \quad lng(T) - 1, \\ lng(str(s)) & = & 1. \end{array}$$

Let a hierarchical data structure $S = STR(S_1, \ldots, S_m)$ be defined by the functions $choo(S)$ and $rest(S)$ constructed with the help of conditional $if - then - else$, superposition and Boolean operations from the following components:

- terms not containing $S_1, \ldots, S_m$;
- the predicate $empty(S_i)$ and the functions $choo(S_i)$, $rest(S_i)$, $last(S_i)$, $head(S_i)$ $(i = 1, \ldots, m)$;
- terms of the form $STR(T_1, \ldots, T_m)$ such that

$$\sum_{i=1}^{m} lng(T_i) < \sum_{i=1}^{m} lng(S_i);$$

– an undefined element ω.

Note that the undefined value ω of the functions $choo(S)$ and $rest(S)$ means $empty(S)$. This definition of hierarchical structures allows us to apply more conveniently the following induction principle 1 to proving the properties of the structures.

Let $prop(STR(T_1, \ldots, T_m))$ denote a property expressed by a first-order logic formula only with free variables $S_1, ..., S_m$, where terms $T_i(S_1, \ldots, S_m)$ are defined in Section 2. The formula is constructed from the term $STR(T_1, \ldots, T_m)$, functional symbols, variables and constants by means of Boolean operations and first-order quantifiers. The functional symbols include $memb$, $empty$, vec, $choo$, $rest$, $last$, $head$, str, con.

Induction principle 1. The property $prop(STR(S_1, \ldots, S_m))$ holds for all structures $S_1, \ldots, S_m$, if there exists an integer $c \geq 0$ such that the following conditions hold:

(1) for all structures $S_1, \ldots, S_m$ such that $\sum_{i=1}^{m} lng(S_i) \leq c$, the property $prop(STR(S_1, \ldots, S_m))$ holds;

(2) for all structures $S_1, \ldots, S_m$ such that $\sum_{i=1}^{m} lng(S_i) > c$, there exist terms $T_1, \ldots, T_m$ for which

$$\sum_{i=1}^{m} lng(T_i) < \sum_{i=1}^{m} lng(S_i)$$

and

$$prop(STR(T_1, \ldots, T_m)) \rightarrow prop(STR(S_1, \ldots, S_m)).$$

Let us consider a definite iteration of the form

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

where S is a data structure that may be hierarchical, x is a variable called a loop parameter, v is a data vector of the loop body ($x \notin v$) and the iteration body $v := body(v, x)$ does not change the structure S. The result of this iteration is an initial value v_0 of the vector v if $empty(S)$. Let $\neg empty(S)$ and $vec(S) = [s_1, \ldots, s_n]$. Then the iteration body iterates sequentially for x defined as $s_1, \ldots, s_n$.

3. Iterations over tuples of data structures

To define a definite iteration over a tuple of data structures $S_1, \ldots, S_m$ (possibly hierarchical), we will use a function $sel(x_1, \ldots, x_m)$ for selection of one structure from the structures $S_1, \ldots, S_m$, where $x_i \in memb(S_i) \bigcup \{\omega\}$

$(i = 1, \ldots, m)$. The function $sel(x_1, \ldots, x_m)$ returns an integer j $(1 \leq j \leq m)$ such that $x_j \neq \omega$, and also $sel(\omega, \ldots, \omega)$ is undefined. In the case of $x_j \neq \omega$ and $x_i = \omega$ for all $i \neq j$ $(i = 1, \ldots, m)$, $sel(x_1, \ldots, x_m) = j$ and the definition of the function sel can be omitted.

Let us consider the iteration over the tuple of structures $S_1, \ldots, S_m$ with the selection function $sel(x_1, \ldots, x_m) = t$ of the form

$$\textbf{for } x_1 \textbf{ in } S_1, \ldots, x_m \textbf{ in } S_m \textbf{ do } v := body\,(v, x_t, t) \textbf{ end}, \tag{2}$$

where v is a data vector ($x_i \notin v$ for all $i = 1, \ldots, m$) and the iteration body does not change the structures $S_1, \ldots, S_m$. If $empty(S_i)$ for each $i = (1, \ldots, m)$, then the iteration result is an initial value v_0 of the vector v. Otherwise, we assume $t = sel(x_1, \ldots, x_m)$, where $x_i = choo(S_i)$ for each $i = 1, \ldots, m$, and also $choo(S_i) = \omega$ provided $empty(S_i)$. A new value v_1 of the vector v is defined so that $v_1 = body(v_0, x_t, t)$. The structure S_t is replaced by the structure $rest(S_t)$, and the other structures S_i $(i \neq t)$ are not changed. The process is applied to v_1 and the resulted structures until all structures become empty. The resulted value v_k $(k = \sum_{i=1}^{m} \mid memb(S_i) \mid)$ of the vector v is assumed to be the result of iteration (2).

Here the purpose is to reduce iteration (2) to iteration (1) with the help of hierarchical structures. We introduce the following notation in order to define a hierarchical structure

$$S = STR(S_1, \ldots, S_m)$$

from the structures $S_1, \ldots, S_m$ and the function $sel(x_1, \ldots, x_m)$. Let

$$\begin{aligned} &EMPTY = (empty(S_1) \wedge \ldots \wedge\ empty(S_m)),\\ &t1 = sel(choo(S_1), \ldots, choo(S_m)),\\ &REST = STR(S_1, \ldots, rest(S_{t1}), \ldots, S_m), \end{aligned}$$

provided $\neg EMPTY$. Then

$$(choo(S), rest(S)) = \textbf{if } EMPTY \textbf{ then } (\omega, \omega) \textbf{else } ((choo(S_{t1}), t1), REST).$$

Notice that this definition is consistent with the definition of hierarchical structures from Section 2, since the quantifiers bounded by the set $\{1, \ldots, m\}$ can be expressed by applying conjunction or disjunction m times, and $empty(S) \equiv (choo(S) = \omega)$.

Theorem 1. *Iteration* (2)*, where* $t = sel(x_1, \ldots, x_m)$*, is equivalent to the iteration*

$$\textbf{for } (x, \tau) \textbf{ in } S \textbf{ do } v := body\,(v, x, \tau) \textbf{ end}, \tag{3}$$

where the hierarchical structure $S = STR(S_1, \ldots, S_m)$ *is defined by means of the function* $sel(x_1, \ldots, x_m)$.

Proof. We will use induction on $\sum_{i=1}^{m} \mid memb(S_i) \mid = k$. If $k = 0$, then $empty(S_i)$ for each $i = 1, \ldots, m$, and, therefore, $empty(S)$. Let us suppose $k > 0$ and iterations (2) and (3) are equivalent if $\sum_{i=1}^{m} \mid memb(S_i) \mid = k-1$. It is evident that iteration (2) is equivalent to the program

$$v := body(v, choo(S_{t1}), t1);\ \textbf{for}\ x_1\ \textbf{in}\ S_1, \ldots, x_{t1}\ \textbf{in}\ rest(S_{t1}), \ldots, x_m\ \textbf{in}\ S_m\ \textbf{do}\ v := body\,(v, x_t, t)\ \textbf{end}. \quad (4)$$

By the inductive hypothesis, program (4) is equivalent to the program

$$v := body(v, choo(S_{t1}), t1);\ \textbf{for}\ (x, \tau)\ \textbf{in}\ REST\ \textbf{do}\ v := body\,(v, x, \tau)\ \textbf{end}. \quad (5)$$

It remains to notice that program (5) is equivalent to iteration (3) according to

$$rest(S) = REST,$$
$$choo(S) = (choo(S_{t1}), t1),$$
$$S = con(choo(S), rest(S))$$

provided $k > 0$. □

We introduce the following notions in order to formulate useful properties of the hierarchical structure $S = STR(S_1, \ldots, S_m)$. We will use $memb_1(S)$ to denote the multiset of the first components of elements of $memb(S)$. Let $vec_i(S)$ be the subsequence of $vec(S)$ which consists of all elements (r, i) for a suitable r, and $vec_{i1}(S)$ be the sequence of the first components of elements of $vec_i(S)$ $(i = 1, \ldots, m)$.

Theorem 2.

2.1. $memb_1(S) = \bigcup_{i=1}^{m} memb(S_i)$.

2.2. $vec_{i1}(S) = vec(S_i)$ *for each* $i = 1, \ldots, m$.

Proof. We will use induction on $\sum_{i=1}^{m} \mid memb(S_i) \mid = k$. If $k = 0$, then $empty(S)$, and $empty(S_i)$ for each $i = 1, \ldots, m$, and Theorem 2 holds. Let us suppose $k > 0$ and Theorem 2 holds for $\sum_{i=1}^{m} \mid memb(S_i) \mid = k - 1$.

2.1. By the induction hypothesis,

$$\begin{aligned} memb_1(S) &= memb_1(con(choo(S), rest(S))) \\ &= \{choo\,(S_{t1})\} \bigcup memb_1(rest(S)) \\ &= \{choo\,(S_{t1})\} \bigcup \bigcup_{i=1}^{t1-1} memb(S_i) \bigcup memb\,(rest(S_{t1})) \bigcup \\ &\qquad \bigcup_{i=t1+1}^{m} memb\,(S_i) \\ &= \bigcup_{i=1}^{m} memb(S_i) \end{aligned}$$

according to $memb(S_{t1}) = \{choo(S_{t1})\} \bigcup memb(rest(S_{t1}))$.

2.2. By the induction hypothesis,

$$\begin{aligned} vec_{i1}(S) &= vec_{i1}(con(choo(S), rest(S))) \\ &= con(vec_{i1}(str(choo(S))), vec_{i1}(rest(S))) \\ &= \textbf{if } i = t1 \textbf{ then } con(choo(S_i), vec(rest(S_i))) \\ &\qquad \textbf{else } vec(S_i) = vec(S_i) \end{aligned}$$

according to $vec_{i1}(str(choo(S)))$ is the empty sequence and $vec_{i1}(rest(S)) = vec(S_i)$ when $i \neq t_1$. □

In the case of $S = STR(S_1, S_2)$, $head(S)$ and $last(S)$ can be expressed simply in terms of $head(S_i)$ and $last(S_i)$ $(i = 1, 2)$.

Theorem 3. *If* $S = STR(S_1, S_2)$ *and* $\neg empty(S)$, *then*

$\neg empty(S_1)$, $head(S) = STR(head(S_1), S_2)$ *and* $last(S) = (last(S_1), 1)$ *or*

$\neg empty(S_2)$, $head(S) = STR(S_1, head(S_2))$ *and* $last(S) = (last(S_2), 2)$.

Proof. For definiteness, we suppose $t1 = 1$. We will use induction on $\mid memb(S) \mid = k$. The induction basis is the case $k = 1$ when $\neg empty(S_1)$, $empty(head(S_1)), empty(S_2), empty(head(S)), last(S) = (last(S_1), 1)$, and Theorem 3 holds. In the case of $k > 1$,

$$\begin{aligned} head(S) &= head(con(choo(S), rest(S))) \\ &= con(choo(S), head(rest(S))) \\ &= con(choo(S), head(STR(rest(S_1), S_2))) \end{aligned}$$

and $last(S) = last(rest(S))$. By the induction hypothesis for the structure $rest(S) = STR(rest(S_1), S_2)$, two cases are possible.

1. $\neg empty(rest(S_1))$ and

$$head(STR(rest(S_1), S_2)) = STR(head(rest(S_1)), S_2).$$

Let us denote the structure $STR(head(S_1), S_2)$ by HS. Then,

$$\begin{aligned} head(S) &= con(choo(S), STR(head(rest(S_1)), S_2)), \\ HS &= con(choo(HS), rest(HS)) \\ &= con(choo(HS), STR(rest(head(S_1)), S_2)). \end{aligned}$$

From $\neg empty(head(S)), choo(HS) = choo(S)$ and

$$rest(head(S_1)) = head(rest(S_1)),$$

it follows that $head(S) = HS$. Moreover,

$$last(rest(S)) = (last(rest(S_1)), 1) = (last(S_1), 1).$$

2. $\neg empty(S_2)$ and

$$head(STR(rest(S_1), S_2)) = STR(rest(S_1), head(S_2)).$$

Let us denote the structure STR (S_1, $head$ (S_2)) by SH. Then,

$$choo(S_2) = choo(head(S_2))$$

provided

$$\begin{array}{l} \neg empty(head(S_2)), \\ head(S) = con(choo(S), STR(rest(S_1), head(S_2))), \\ SH = con(choo(SH), rest(SH)), \\ rest(SH) = STR(rest(S_1), head(S_2)). \end{array}$$

From $choo(S) = choo(SH) = (choo(S_1), 1)$ it follows that $head(S) = SH$. Moreover, $last(rest(S)) = last(S_2)$. □

The following example shows that Theorem 3 does not hold for the structure $S = STR(S_1, S_2, S_3)$. Let $memb(S_i) = \{a_i\}$ $(i = 1, 2, 3)$, where $a_1 < a_2 < a_3$. We define the function $sel(x_1, x_2, x_3)$ as

if $x_1 \neq \omega \wedge x_2 \neq \omega \wedge x_3 \neq \omega$ **then** 1
else if $x_i = \omega \wedge x_j \neq \omega \wedge x_l \neq \omega$ **then** $max(j, l)$,

where i, j, l are different. Then

$$\begin{array}{lcl} vec(S) & = & [(a_1, 1)(a_3, 3)(a_2, 2)], \\ vec(head(S)) & = & [(a_1, 1)(a_3, 3)], \\ last(S) & = & (a_2, 2) = (last(S_2), 2). \end{array}$$

Therefore, $vec(STR(S_1, head(S_2), S_3)) = [(a_3, 3)(a_1, 1)] \neq vec(head(S))$.

Let us consider a special case when the iteration body does not depend on t, i. e., the iteration has the form

for x_1 **in** $S_1, \ldots, x_m$ **in** S_m **do** $v := body(v, x_t)$ **end**. $(2')$

In this case, the definition of $STR(S_1, \ldots, S_m)$ and Theorems 1–3 can be simplified. Let $(choo(S'), rest(S')) =$ **if** $EMPTY$ **then** (ω, ω) **else** $(choo(S_{t1}), REST)$.

Theorem 1′. *Iteration (2′), where $t = sel(x_1, \ldots, x_m)$, is equivalent to the iteration*

for x **in** S' **do** $v := body(v, x)$ **end**, $(3')$

where the hierarchical structure $S' = STR(S_1, \ldots, S_m)$ is defined by means of the function $sel(x_1, \ldots, x_m)$.

To formulate a theorem similar to Theorem 2.2, we introduce the following notion. A relation $a < b$ is satisfiable for a vector $[v_1, \ldots, v_n]$, if there exist integers i, j such that $1 \leq i < j \leq n$, $a = v_i$ and $b = v_j$.

Theorem 2′.

2′.1. $memb(S') = \bigcup_{i=1}^{m} memb(S_i)$.

2′.2. *If there exists* i $(1 \leq i \leq m)$ *such that a relation* $a < b$ *is satisfiable for the vector* $vec(S_i)$*, then the relation* $a < b$ *is satisfiable for the vector* $vec(S')$.

Theorem 3′. *If* $S' = STR(S_1, S_2)$ *and* $\neg empty(S')$*, then* $\neg empty(S_1)$, $head(S') = STR(head(S_1), S_2)$ *and* $last(S') = last(S_1)$*, or* $\neg empty(S_2)$, $head(S') = STR(S_1, head(S_2))$ *and* $last(S') = last(S_2)$.

It should be noted that the proofs of Theorems 1′, 2′.1, 3′ are similar to the proofs of Theorems 1, 2.1, 3, respectively. It remains to prove Theorem 2′.2. If the relation $a < b$ is satisfiable for the vector $vec(S_i)$, then there exists an integer l $(l \geq 0)$ such that $a = choo(rest^l(S_i))$, $b \in memb(rest^{l+1}(S_i))$. By Theorem 2′.1, $memb(S_i) \subseteq memb(S')$. From this it follows that there exist structures $\bar{S}_j$ $(j = 1, \ldots, m)$ and an integer r $(r \geq 0)$ such that

$$\begin{aligned} &rest^r(S') = STR(\bar{S}_1, \ldots, \bar{S}_m),\\ &a = choo(rest^r(S')),\\ &i = sel(choo(\bar{S}_1), \ldots, choo(\bar{S}_m)),\\ &\bar{S}_i = rest^l(S_i). \end{aligned}$$

By Theorem 2′.1, $b \in memb(rest^{l+1}(S_i)) \subseteq memb(rest^{r+1}(S'))$. From this it follows that the relation $a < b$ is satisfiable for the vector $vec(S')$. □

4. Iterations over tuples of data structures with exit statement

Let us consider the following definite iteration over a tuple of data structures with a body including the statement of termination of the iteration EXIT:

$$\begin{aligned} &\textbf{for } x_1 \textbf{ in } S_1, \ldots, x_m \textbf{ in } S_m \textbf{ do } v := body_1(v, x_t, t);\ \textbf{if } cond(x_1, \ldots, x_m)\\ &\textbf{then } EXIT; v := body_2(v, x_t, t) \textbf{ end}, \end{aligned} \tag{6}$$

where $t = sel(x_1, \ldots, x_m), x_i \notin v$ $(i = 1, \ldots, m)$, the condition $cond(x_1, \ldots, x_m)$ $(x_i \in memb(S_i) \bigcup \{\omega\})$ does not depend on variables from v, and the iteration body does not change the structures S_j $(j = 1, \ldots, m)$.

We will use $b1$ to denote $cond(choo(S_1), \ldots, choo(S_m))$, along with the notions $EMPTY$, $t1$, $REST$ from Section 3. Let us describe operational semantics of iteration (6). If $EMPTY$, then the result of iteration (6) is an initial value v_0 of the vector v. Otherwise, the result of iteration (6) is $v_1 = body_1(v_0, choo(S_{t1}), t1)$, when $b1$ is true. If $b1$ is false, then this process is continued with $v = v_2 = body_2(v_1, choo(S_{t1}), t1)$ and the structure $rest(S_{t1})$ instead of S_{t1}, when the other structures S_i $(i \neq t1)$ are not changed. A value v_l resulting from this process is the result of iteration (6).

Here the purpose is to reduce iteration (6) to iteration (1) with the help of hierarchical structures. Let us define a hierarchical structure $T = STR(S_1, \ldots, S_m)$ using the selection function $sel(x_1, \ldots, x_m)$ and the condition $cond(x_1, \ldots, x_m)$. The structure elements are triples (s, i, c), where $s \in memb(S_j)$ $(1 \leq j \leq m)$, $1 \leq i \leq m$, $c \in \{true, false\}$. Let

$$(choo(T),\ rest(T)) = \textbf{if } EMPTY \textbf{ then } (\omega, \omega) \textbf{else}$$
$$\textbf{if } b1 \textbf{ then } ((choo(S_{t1}), t1, true), \omega)$$
$$\textbf{else } ((choo(S_{t1}), t1,\ false), REST).$$

The element $last(T)$ can be represented in the form $(last_1(T), last_2(T), last_3(T))$.

Theorem 4. *Iteration* (6)*, where* $t = sel(x_1, \ldots, x_m)$*, is equivalent to the program*

$$\textbf{if } \neg empty(T) \textbf{ then begin for } (x, \tau, false) \textbf{ in } head(T) \textbf{ do} \tag{7}$$
$$v := body_1(v, x, \tau);\ v := body_2(v, x, \tau) \textbf{ end}; v := body_1(v, last_1(T), last_2(T));$$
$$\textbf{if } \neg last_3(T) \textbf{ then } v := body_2(v, last_1(T), last_2(T)) \textbf{ end},$$

where the hierarchical structure $T = STR(S_1, \ldots, S_m)$ *is defined by means of the function* $sel(x_1, \ldots, x_m)$ *and the condition* $cond(x_1, \ldots, x_m)$*.*

Proof. We will use induction on $k = \sum_{i=1}^{m} \mid memb(S_i) \mid$. If $k = 0$ then $empty(S_i)$ for each $i = 1, \ldots, m$, and, therefore, $empty(T)$ and Theorem 4 holds. Let us suppose $k > 0$ and Theorem 4 holds for $k-1$. Hence $\neg empty(T)$. Two cases are possible.

1. $b1 = true$. Then iteration (6) is equivalent to the statement $v := body_1(v, choo(S_{t1}), t1)$. The structure T consists of one element $last(T) = (choo(S_{t1}), t1, true)$, and $empty(head(T))$. Therefore, program (7) is equivalent to the statement $v := body_1(v, choo(S_{t1}), t1)$.

2. $b1 = false$. Then iteration (6) is equivalent to the program

$$v := body_1(v, choo(S_{t1}), t1);\ v := body_2(v, choo(S_{t1}), t1); \tag{8}$$
$$\textbf{for } x_1 \textbf{ in } S_1, \ldots, x_{t1} \textbf{ in } rest(S_{t1}), \ldots, x_m \textbf{ in } S_m \textbf{ do } v := body_1(v, x_t, t);$$
$$\textbf{if } cond(x_1, \ldots, x_m) \textbf{ then } EXIT;\ v := body_2(v, x_t, t) \textbf{ end}.$$

Two cases are possible.

2.1. $k = 1$. Then $empty(rest(S_{t1}))$ and $empty(S_i)$ for each $i \neq t1$ ($i = 1, \ldots, m$). Therefore, program (8) is equivalent to the program

$$v := body_1(v, choo(S_{t1}), t1);\ v := body_2(v, choo(S_{t1}), t1). \tag{9}$$

In this case the structure T consists of one element $last(T) = (choo(S_{t1}), t1, false)$, and $empty(head(T))$. Hence, program (7) is equivalent to program (9).

2.2. $k > 1$. Then $\neg empty(rest(T))$. By the induction hypothesis, program (8) is equivalent to the program

$$\begin{aligned} &v := body_1(v, choo(S_{t1}), t1);\ v := body_2(v, choo(S_{t1}), t1); \\ &\textbf{for } (x, \tau, false) \textbf{ in } head(rest(T)) \textbf{ do } v := body_1(v, x, \tau); \\ &v := body_2(v, x, \tau) \textbf{ end}; v := body_1(v, last_1(rest(T)), last_2(rest(T))); \\ &\textbf{if } \neg last_3(rest(T)) \textbf{ then } v := body_2(v, last_1(rest(T)), last_2(rest(T))). \end{aligned} \tag{10}$$

From

$$\begin{aligned} &last(T) = last(rest(T)), \\ &head(T) = con(choo(T), head(rest(T))), \\ &choo(T) = choo(head(T)) = (choo(S_{t1}), t1, false) \end{aligned}$$

it follows that program (10) is equivalent to program (7). □

Theorem 4 can be simplified when the body of iteration (6) begins or ends with the exit statement, i. e., $body_i(v, x_t, t) = v$ for $i = 1$ or $i = 2$. At first, we consider the case $i = 2$. In the case, elements of a new structure R come from corresponding elements of the structure T by elimination of the third component. Let

$$\begin{aligned} (choo(R), rest(R)) = \textbf{if } EMPTY \textbf{ then } (\omega, \omega) \textbf{ else } ((choo(S_{t1}), t1), \\ \textbf{if } b1 \textbf{ then } \omega \textbf{ else } REST). \end{aligned}$$

Corollary 1. *The iteration*

$$\begin{aligned} &\textbf{for } x_1 \textbf{ in } S_1, \ldots, x_m \textbf{ in } S_m \textbf{ do } v := body(v, x_t, t);\ \textbf{if } cond(x_1, \ldots, x_m) \\ &\textbf{then } EXIT \textbf{ end}, \end{aligned} \tag{11}$$

where $t = sel(x_1, \ldots, x_m)$, is equivalent to the iteration

$$\textbf{for } (x, \tau) \textbf{ in } R \textbf{ do } v := body(v, x, \tau) \textbf{ end}, \tag{12}$$

where the hierarchical structure $R = STR(S_1, \ldots, S_m)$ is defined by means of the function $sel(x_1, \ldots, x_m)$ and the condition $cond(x_1, \ldots, x_m)$.

Proof. By Theorem 4, iteration (11) is equivalent to the program

if $\neg empty(R)$ **then begin for** (x, τ) **in** $head(R)$ **do** $v := body(v, x, \tau)$ **end**; $v := body(v, last_1(R), last_2(R))$ **end**,

which is equivalent to iteration (12). □

Let us consider the case when the iteration body begins with the exit statement. We will use $pr(T)$ to denote a structure consisting of elements which are the projections of corresponding elements of the structure T on the first and second components. Then $R = pr(T)$. Let us define the following hierarchical structure $K = STR(S_1, \ldots, S_m)$ as

$$(choo(K), rest(K)) = \textbf{if } EMPTY \vee b1 \textbf{ then } (\omega, \omega) \textbf{ else } ((choo(S_{t1}), t1), REST).$$

Claim 1. $K = \textbf{if } empty(T) \vee \neg last_3(T) \textbf{ then } pr(T) \textbf{ else } pr(head(T))$.

Proof. We will use induction on $\mid memb(T) \mid = l$. If $l = 0$, then $empty(T)$, $empty(pr(T))$ and Claim 1 holds. Let us suppose $l > 0$ and Claim 1 holds for $l - 1$. Two cases are possible.

1. $b1 = true$. Then $empty(K)$ and the structure T consists of the one element $(choo(S_{t1}), t1, true)$. Therefore, $empty(head(T))$, $last_3(T) = true$ and Claim 1 holds.

2. $b1 = false$. We will denote the structure $REST$ from the definitions of T and K as $REST_T$ and $REST_K$, respectively. Then

$$\begin{aligned} &choo(T) = (choo(S_{t1}), t1, false),\\ &rest(T) = REST_T,\\ &choo(K) = pr(choo(T)),\\ &rest(K) = REST_K. \end{aligned}$$

Two cases are possible.

2.1. $last_3(T) = false$. Then $empty(rest(T))$ or $last_3(rest(T)) = false$. By the induction hypothesis, $rest(K) = REST_K = pr(REST_T) = pr(rest(T))$. From this it follows that

$$\begin{aligned} &K = con(choo(K),\\ &rest(K)) = con(pr(choo(T)),\\ &pr(rest(T))) = pr(T). \end{aligned}$$

2.2. $last_3(T) = true$. Then $\neg empty(rest(T))$ and $last_3(rest(T)) = true$. By the induction hypothesis, $rest(K) = REST_K = pr(head(REST_T)) = pr(head(rest(T)))$. From this it follows that

$$K = con(pr(choo(T)),$$
$$pr(head(rest(T)))) = pr(con(choo(T),$$
$$head(rest(T)))) = pr(head(T)). \quad \square$$

Corollary 2. *The iteration*

$$\textbf{for } x_1 \textbf{ in } S_1, \ldots, x_m \textbf{ in } S_m \textbf{ do if } cond(x_1, \ldots, x_m) \textbf{ then } EXIT;$$
$$v := body(v, x_t, t) \textbf{ end}, \tag{13}$$

where $t = sel(x_1, \ldots, x_m)$*, is equivalent to the iteration*

$$\textbf{for } (x, \tau) \textbf{ in } K \textbf{ do } v := body(v, x, \tau) \textbf{ end} \tag{14}$$

where the hierarchical structure $K = STR(S_1, \ldots, S_m)$ *is defined by means of the function* $sel(x_1, \ldots, x_m)$ *and the condition* $cond(x_1, \ldots, x_m)$*.*

Proof. If $empty(T)$, then $empty(K)$ by Claim 1. It remains to apply Theorem 4. When $\neg empty(T)$, we consider two cases.

1. $last_3(T) = true$. Then by Claim 1, $K = pr(head(T))$. From Theorem 4 it follows that iteration (13) is equivalent to iteration (14).

2. $last_3(T) = false$. Then by Claim 1, $K = pr(T)$. By Theorem 4, iteration (13) is equivalent to the program

$$\textbf{for } (x, \tau) \textbf{ in } head(K) \textbf{ do } v := body(v, x, \tau) \textbf{ end};$$
$$v := body(v, last_1(K), last_2(K))$$

which is equivalent to iteration (14). $\square$

In the case when the body of iteration (6) does not depend on t, Theorem 4 and Corollaries 1, 2 can by simplified. Let us define hierarchical structures T', R', K' similar to the structures T, R, K, respectively.

$$(choo(T'), rest(T')) = \textbf{if } EMPTY \textbf{ then } (\omega, \omega) \textbf{ else if } b1 \textbf{ then}$$
$$((choo(S_{t1}), true), \omega) \textbf{ else } ((choo(S_{t1}), false), REST),$$
$$(choo(R'), rest(R')) = \textbf{if } EMPTY \textbf{ then } (\omega, \omega) \textbf{ else } (choo(S_{t1}),$$
$$\textbf{if } b1 \textbf{ then } \omega \textbf{ else } REST),$$
$$(choo(K'), rest(K')) = \textbf{if } EMPTY \vee b1 \textbf{ then } (\omega, \omega)$$
$$\textbf{else } (choo(S_{t1}), REST).$$

Theorem 4′. *The iteration*

$$\textbf{for } x_1 \textbf{ in } S_1, \ldots, x_m \textbf{ in } S_m \textbf{ do } v := body_1(v, x_t); \textbf{ if } cond(x_1, \ldots, x_m)$$
$$\textbf{then } EXIT; \ v := body_2(v, x_t) \textbf{ end},$$

where $t = sel(x_1, \ldots, x_m)$*, is equivalent to the program*

$$\textbf{if } \neg empty(T') \textbf{ then begin for } (x, false) \textbf{ in } head(T') \textbf{ do}$$
$$v := body_1(v, x); \ v := body_2(v, x) \textbf{ end}; \ v := body_1(v, last_1(T'));$$

if $\neg last_2(T')$ **then** $v := body_2(v, last_1(T'))$ **end**,

where the hierarchical structure $T' = STR(S_1, \ldots, S_m)$ *is defined by means of the function* $sel(x_1, \ldots, x_m)$ *and the condition* $cond(x_1, \ldots, x_m)$.

Corollary 1′. *The iteration*

for x_1 **in** $S_1, \ldots, x_m$ **in** S_m **do** $v := body(v, x_t)$;
if $cond(x_1, \ldots, x_m)$ **then** $EXIT$ **end**,

where $t = sel(x_1, \ldots, x_m)$, *is equivalent to the iteration*

for x **in** R' **do** $v := body(v, x)$ **end**,

where the hierarchical structure $R' = STR(S_1, \ldots, S_m)$ *is defined by means of the function* $sel(x_1, \ldots, x_m)$ *and the condition* $cond(x_1, \ldots, x_m)$.

Corollary 2′. *The iteration*

for x_1 **in** $S_1, \ldots, x_m$ **in** S_m **do if** $cond(x_1, \ldots, x_m)$ **then** $EXIT$;
$v := body(v, x_t)$ **end**,

where $t = sel(x_1, \ldots, x_m)$, *is equivalent to the iteration*

for x **in** K' **do** $v := body(v, x)$ **end**,

where the hierarchical structure $K' = STR(S_1, \ldots, S_m)$ *is defined by means of the function* $sel(x_1, \ldots, x_m)$ *and the condition* $cond(x_1, \ldots, x_m)$.

It should be noted that proofs of Theorem 4′ and Corollaries 1′, 2′ are similar to the proofs of Theorem 4 and Corollaries 1, 2, respectively.

5. Replacement operation and its application

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 $v_0 = v$ provided $empty(S)$, $v_i = body(v_{i-1}, s_i)$ for all $i = 1, \ldots, n$ provided $\neg empty(S)$ and $vec(S) = [s_1, \ldots, s_n]$. Therefore, $rep(v, str(s), body) = body(v, s)$. The following theorem similar to Theorems 4 and 6 [11] presents useful properties of the replacement operation.

Theorem 5.

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

5.2. *Iteration (1) is equivalent to the multiple assignment*

$v := rep(v, S, body)$.

Corollary 3.

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

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

The replacement operation allows us to formulate the following proof rule without invariants for iteration (1). Let $R(y \leftarrow exp)$ be a result of substitution of an expression exp for all occurrences of a variable y into a formula R. Let $R(vec \leftarrow vexp)$ denote the result of a synchronous substitution of the components of an expression vector $vexp$ for all occurrences of the corresponding components of a vector vec into a formula R.

rl1. $\{P\}prog\{Q(v \leftarrow rep(v, S, body))\} \vdash$
$\{P\}prog;$ **for** x **in** S **do** $v := body(v, x)$ **end** $\{Q\}$,

where the post-condition Q does not depend on the loop parameter x.

The following corollary is evident from Theorem 5.2.

Corollary 4. *The proof rule rl*1 *is derived in the standard system of proof rules for usual statements including the multiple assignment.*

To prove the verification conditions including the replacement operation $rep(v, S, body)$, the following induction principle is used.

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 $rep(v, S, body)$, functional symbols, variables and constants by means of Boolean operations, first-order quantifiers and substitution of constants for variables from v.

Induction principle 2. The property $prop(rep(v, S, body))$ holds for each structure S, if there exists an integer $c \geq 0$ such that two conditions hold:
(1) for each structure S such that $lng(S) \leq c$, the property $prop(rep(v, S, body))$ holds;
(2) for each structure S such that $lng(S) > c$, there exists a term $T(S)$ for which $lng(T(S)) < lng(S)$ and $prop(rep(v, T(S), body)) \rightarrow prop(rep(v, S, body))$.

6. Case of study: iterations over files

In the case of definite iterations over files, we will apply a problem-oriented technique for proving verification conditions containing the replacement operation. The technique requires to impose some restrictions on the body of iteration (1). We assume that this body has the form $(f, v) := body(f, v, x)$, where x is an iteration parameter, f is a file and v is a vector of other variables. We also suppose that the projections of $body$ on f and v can be represented in the form $body_f(f, v, x) = con(f, e(v, x))$, $body_v(f, v, x) =$

$bdv(v,x)$, where the expression $e(v,x)$ presents a file and, moreover, $e(v,x)$ and $bdv(v,x)$ do not depend on f. Notice that the representation of the iteration body is natural when the iteration realizes writing in the file f. We consider $rep(S)$ to be a short form for $rep((f,v),S,body)$. Let $rep_f(S)$ and $rep_v(S)$ be the projections of $rep(S)$ on f and v, respectively.

Theorem 6. *In the case of* $\neg empty(S)$

6.1. $rep_f(S) = con(rep_f(head(S)), e(rep_v(head(S)), last(S)))$,

6.2. $rep_f(S) = con(f, e(v, choo(S)), rep_f((\emptyset, bdv(v, choo(S))), rest(S), body))$, *where* $\emptyset$ *is the empty file.*

Proof. Theorem 6.1 immediately follows from Corollary 3.1. To prove Theorem 6.2, we will use induction on $n =\mid memb(S) \mid$. If $n = 1$, then Theorem 6.2 follows from

$$rep_f((f,v),S,body) = body_f(f,v,choo(S)) = con(f, e(v, choo(S)))$$

and

$$rep_f((\emptyset, bdv(v, choo(S))), rest(S), body) = \emptyset.$$

Let us suppose $n > 1$ and Theorem 6.2 holds for $n-1$. We will use bf and bv to denote $body_f(f,v,choo(S))$ and $body_v(f,v,choo(S))$, respectively. From Corollary 3.2 and the induction hypothesis, it follows that

$$\begin{aligned} rep_f(S) &= rep_f((bf,bv), rest(S), body) \\ &= con(bf, e(bv, choo(rest(S))), rep_f((\emptyset, bdv(bv, choo(rest(S)))), \\ &\qquad rest^2(S), body)). \end{aligned}$$

It remains to notice that $bf = con(f,\ e(v, choo(S)))$, $bv = bdv(v, choo(S))$, and, by the induction hypothesis, $rep_f\,((\emptyset, bv), rest\,(S), body) =$ $con(\emptyset, e\,(bv, choo(rest(S))), rep_f((\emptyset, bdv(bv, choo(rest(S)))), rest^2(S), body)$. □

7. Illustrative examples

Example 1. Merging of ordered files with cleaning.

To specify a merging program, we introduce the following notation. Let $set(f)$ be a set of all elements of the file f. Let $ord(f)$ (respectively, $sord(f)$) denote a predicate whose value is true, if the file f has been sorted in ascending order $\leq$ (respectively, $<$) of elements, and false otherwise. We assume that $ord(\emptyset) = sord(\emptyset) = true$. Let $file(z)$ be a function which returns a file consisting of one element z.

We consider the following program for merging (with cleaning) of ordered files f_1 and f_2 to an ordered file g:

$$\{P\}\ g := \emptyset;\ y := \omega;\ \textbf{for } x_1 \textbf{ in } f_1, x_2 \textbf{ in } f_2 \textbf{ do} \tag{15}$$
$$\textbf{if } y \neq x_t \textbf{ then begin } g := con(g, x_t);\ y := x_t \textbf{ end end } \{Q\},$$

where $t = sel(x_1, x_2) = \textbf{if } x_1 \leq x_2 \textbf{ then } 1 \textbf{ else } 2$ for $x_1 \neq \omega$ and $x_2 \neq \omega$, $P = ord(f_1) \wedge ord(f_2)$, $Q = (sord(g) \wedge set(g) = set(f_1) \bigcup set(f_2))$.

By Theorem $1'$, program (15) is equivalent to the following program

$$\{P\}\ g := \emptyset;\ y := \omega;\ \textbf{for } x \textbf{ in } F \textbf{ do } (g, y) := (body_g(g, y, x), body_y(y, x))$$
$$\textbf{end } \{Q\}, \tag{16}$$

where

$$body_g(g, y, x) = con(g, e(y, x)), e(y, x) = \ \textbf{if } y \neq x \textbf{ then } x \textbf{ else } \emptyset,$$
$$body_y(y, x) = \ \textbf{if } y \neq x \textbf{ then } x \textbf{ else } y,$$

and the hierarchical structure $F = STR(f_1, f_2)$ is defined as

$$(choo(F), rest(F)) = \textbf{if } EMPTY \textbf{ then } (\omega, \omega) \textbf{ else } (choo(f_{t1}), REST).$$

By the proof rule $rl1$ from Section 5, one verification condition

$$P \rightarrow Q(g \leftarrow rep_g((\emptyset, \omega), F, body))$$

is generated from program (16).

From $body_y(y, x) = x$, Corollary 3.1, Theorem 6.1 and induction principle 2 for $c = 1$, Claim 2 follows.

Claim 2. In the case of $\neg empty(F)$, the following properties hold:

2.1. $rep_y(F) = last(F)$,

2.2. $rep_g(F) = \textbf{if } empty(head(F)) \textbf{ then } file(last(F)) \textbf{ else } con(rep_g(head(F)), e(last(head(F)), last(F)))$,

2.3. $last(rep_g(F)) = last(F)$.

From Theorem $2'.2$, $3'$ and induction principle 1 for $c = 1$, Claim 3 follows.

Claim 3. $ord(f_1) \wedge ord(f_2) \rightarrow ord(vec(F))$.

To prove the verification condition, we will use induction principle 2 for $c = 1$. In the case of $empty(F)$, $empty(f_i)$ $(i = 1, 2)$, $rep_g(F) = \emptyset$ and Claim 3 is evident. In the case of $\neg empty(F)$ and $empty(head(F))$, Claim 3 follows from Claim 2.2, Theorem $2'.1$ and $rep_g(F) = file(last(F))$. Let us suppose $\neg empty(head(F))$ and $\neg empty(f_1)$. By Claim 2.2,

$$rep_g(F) = con(rep_g(head(F)), e(last(head(F)), last(F))).$$

By Theorem 3′, $head(F) = STR(head(f_1), f_2)$ and $last(F) = last(f_1)$. From the induction hypothesis for $head(F)$, $ord(head(f_1))$and $ord(f_2)$, it follows that $sord(rep_g(head(F)))$ and $set(rep_g(head(F))) = set(head(f_1)) \bigcup set(f_2)$. Two cases are possible.

1. $last(head(F)) = last(F)$. Then $rep_g(F) = rep_g(head(F))$. It remains to notice that, by Claim 2.3, $last(rep_g(F)) = last(f_1) \in set(rep_g(head(F)))$ and $set(f_1) = set(head(f_1)) \bigcup \{last(f_1)\}$.

2. $last(head(F)) \neq last(F)$. Then $rep_g(F) = con(rep_g(head(F)), last(F))$.

It remains to apply Claims 2.3 and 3.

Example 2. Intersection of ordered files.

To specify an intersection program, we use the notation from example 1. We consider the following program which forms the intersection of ordered files f_1 and f_2 as a strictly ordered file g:

$$\begin{aligned} &\{P\}\ g := \emptyset;\ y_1 := \omega;\ y_2 := \omega;\ \textbf{for}\ x_1\ \textbf{in}\ f_1, x_2\ \textbf{in}\ f_2\ \textbf{do} \\ &\textbf{if}\ y_1 \neq x_t\ \textbf{then begin}\ y_1 := x_t;\ y_2 := t\ \textbf{end else} \\ &\textbf{if}\ y_2 \neq t\ \textbf{then begin}\ g := con(g, y_1);\ y_2 := t\ \textbf{end}; \\ &\textbf{if}\ x_1 = \omega \vee x_2 = \omega\ \textbf{then}\ EXIT\ \textbf{end}\ \{Q\}, \end{aligned} \tag{17}$$

where $t = sel(x_1, x_2) =$ **if** $x_1 \leq x_2$ **then** 1 **else** 2 for $x_1 \neq \omega$ and $x_2 \neq \omega$, $P = ord(f_1) \wedge ord(f_2)$, $Q = (sord(g) \wedge set(g) = set(f_1) \bigcap set(f_2))$.

By Corollary 1, program (17) is equivalent to the following program

$$\begin{aligned} &\{P\}\ g := \emptyset;\ y_1 := \omega;\ y_2 := \omega;\ \textbf{for}\ (x, \tau)\ \textbf{in}\ R\ \textbf{do} \\ &(g, y_1, y_2) := body(g, y_1, y_2, x, \tau)\ \textbf{end}, \end{aligned} \tag{18}$$

where

$$\begin{aligned} &body_g(g, y_1, y_2, x, \tau) = con(g, e(y_1, y_2, x, \tau)), \\ &e(y_1, y_2, x, \tau) = \textbf{if}\ y_1 \neq x \vee y_2 = \tau\ \textbf{then}\ \emptyset\ \textbf{else}\ x, \\ &body_v(y_1, y_2, x, \tau) = \textbf{if}\ y_1 \neq x\ \textbf{then}\ (x, \tau)\ \textbf{else} \\ &\qquad\qquad \textbf{if}\ y_2 \neq \tau\ \textbf{then}\ (x, \tau)\ \textbf{else}\ (y_1, y_2) = (x, \tau), \end{aligned}$$

and the hierarchical structure $R = STR(f_1, f_2)$ is defined as

$$\begin{aligned} (choo(R), rest(R)) = \ &\textbf{if}\ EMPTY\ \textbf{then}\ (\omega, \omega)\ \textbf{else}\ ((choo\,(f_{t1}), t1), \\ &\textbf{if}\ b1\ \textbf{then}\ \omega\ \textbf{else}\ REST). \end{aligned}$$

By the proof rule $rl1$, one verification condition

$$VC : P \to Q(g \leftarrow rep_g((\emptyset, \omega, \omega), R, body))$$

is generated from program (18).

We will use $h(R)$ to denote the file $rep_g((\emptyset, choo(f_{t1}), t1), rest(R), body)$ in the case of $\neg empty(R)$. The following claim follows from Theorem 6.2.

Claim 4. *If* $\neg empty(R)$, *then*

$$rep_g((\emptyset, y_1, y_2), R, body) = \textbf{if } y_1 \neq choo(f_{t1}) \vee y_2 = t1 \textbf{ then } h(R) \textbf{ else } con(choo(f_{t1}), h(R)),$$

where $R = STR(f_1, f_2)$ *and* $t1 = sel(choo(f_1), choo(f_2))$.

The verification condition VC is proved by the case analysis. If $empty(R)$, then $empty(f_i)(i = 1, 2)$ and VC holds. Let us suppose $\neg empty(R)$ and $t1 = 1$. By Claim 4, $rep_g((\emptyset, \omega, \omega), R, body) = h(R)$. We will use induction on $\mid memb(R) \mid = k$. If $k = 1$, then $empty(rest(R)), empty(h(R)), empty(f_2)$, and, therefore, VC holds. We suppose $k > 1$ and VC holds for the structure $rest(R)$. Then $\neg empty(f_i)(i = 1, 2)$, $\neg empty(rest(R))$ and $rest(R) = STR(rest(f_1), f_2)$. Two cases are possible.

1. $choo(rest(R)) = (choo(rest(f_1)), 1)$. Then

$$choo(f_1) \leq choo(rest(f_1)) \leq choo(f_2).$$

From Claim 4 it follows that $h(R) = h(rest(R))$. By the induction hypothesis, $sord(h(R))$ and $set(h(R)) = set(rest(f_1)) \bigcap set(f_2) = set(f_1) \bigcap set(f_2)$, since $choo(f_1) \notin set(f_2)$ in the case of $choo(f_1) < choo(rest(f_1))$.

2. $choo(rest(R)) = (choo(f_2), 2)$. Then $choo(f_1) \leq choo(f_2)$. From Claim 4 it follows that

$$h(R) = \textbf{if } choo(f_1) \neq choo(f_2) \textbf{ then } h(rest(R)) \textbf{ else } con(choo(f_2),\ h(rest(R)))$$

because $y_1 = choo(f_1), y_2 = 1, t1 = sel(rest(f_1), f_2)$ and R is changed by $rest(R)$. By the induction hypothesis, $sord(h(rest(R)))$ and

$$set(h(rest(R))) = set(rest(f_1)) \bigcap set(f_2).$$

In the case of $choo(f_1) < choo(f_2), VC\ holds$ since $h(R) = h(rest(R))$ and $choo(f_1) \notin set(f_2)$. When $choo(f_1) = choo(f_2)$,

$$\begin{aligned} set(h(R)) &= \{choo(f_2)\} \bigcup (set(rest(f_1)) \bigcap set(f_2)) \\ &= (\{choo(f_1)\} \bigcup set(rest(f_1))) \bigcap set(f_2) \\ &= set(f_1) \bigcap set(f_2). \end{aligned}$$

It remains to prove $sord(h(R))$ in the case of $choo(f_1) = choo(f_2)$.
If $empty(rest(f_1))$, then $empty(h(rest(R))), h(R) = file(choo(f_2))$ and, therefore, $sord(h(R))$. Otherwise, $choo(f_2) < choo(rest(f_1))$ and $choo(f_2) \notin set(h(rest(R)))$. From this and $sord(h(rest(R)))$, $sord(h(R))$ follows.

8. Conclusion

A symbolic method for verification of definite iterations over tuples of data structures is described in this paper. The new kind of definite iterations allows loops with several input data structures to be represented compactly and naturally, and to be used as flat loops [1]. Moreover, exit from iteration body under a condition is accepted that allows the use of an important kind of while-loops.

The application of the symbolic verification method to definite iterations over tuples of data structures consists of 3 stages. In the first stage of verification, these iterations are transformed to standard definite iterations over hierarchical data structures. Such transformations are justified by Theorems 1, $1'$, 4, $4'$. Useful properties of the transformations are presented by Theorems 2, $2'$, 3, $3'$. In the second stage, verification conditions containing the replacement operation are generated with the help of the rule $rl1$. In the third stage, the verification conditions are proved with the help of both a universal technique based on induction principles 1 and 2, and a problem-oriented technique presented by Theorem 6 for iterations over files.

Instead of loop invariants, the symbolic method uses properties of both hierarchical structures and the replacement operation. These properties, as a rule, are simpler than loop invariants, and new notions are not necessary for representation of the properties. Induction principles 1 and 2 are rather flexible and allow us to use, for proving the properties, different induction strategies including forward and backward strategies based on Corollary 3. The use of properties of hierarchical data structures simplifies presentation of properties of the replacement operation, and, hence, proving them.

Examples 1 and 2 illustrate advantages of the symbolic verification method. The use of the exit statement in Example 2 allows us to present an optimal version of a suitable program. It is suggested to apply the symbolic method to verification of definite iterations over tuples of linear lists.

References

[1] Abd-El-Hafiz S. K., Basili V. R. A knowledge-based approach to the analysis of loops // IEEE Trans. Software Engineering. — 1996. — Vol. 22, № 5. — P. 339–360.

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

[3] Gries D., Gehani N. Some ideas on data types in high-level languages // Communs. ACM. — 1977. — Vol. 20, № 6. — P. 414–420.

[4] Hehner E. C. R., Gravell A. M. Refinement semantics and loop rules // Lect. Notes Comput. Sci. — 1999. — Vol. 1709. — P. 1497–1510.

[5] Hoare C. A. R. An axiomatic basis of computer programming // Communs. ACM. — 1969. — Vol. 12, № 10. — P. 576–580.

[6] Hoare C. A. R. A note on the for statement // BIT. — Vol. 12, № 3. — P. 334–341.

[7] Linger R. C., Mills H. D., Witt B. I. Structured Programming: Theory and Practice. — Addison-Wesley, Reading, MA, 1979.

[8] Mills H. D. Structured programming: retrospect and prospect // IEEE Software. — 1986. — Vol. 3, № 6. — P. 58–67.

[9] Nepomniashy V. A. Loop invariant elimination in program verification // Programming and Computer Software. — 1985. — № 3. — P. 129–137.

[10] Nepomniaschy V. A. On problem-oriented program verification // Programming and Computer Software. — 1986. — № 1. — P. 1–9.

[11] Nepomniaschy V. A. Symbolic verification method for definite iteration over data structures // Information Processing Letters. — 1999. — Vol. 69. — P. 207–213.

[12] Nepomniaschy V. A. Verification of definite iteration over hierarchical data structures // Lect. Notes Comput. Sci. — 1999. — Vol. 1577. — P. 176–187.

[13] Stark J., Ireland A. Invariant discovery via failed proof attempts // Lect. Notes Comput. Sci. — 1999. — Vol. 1559. — P. 271–288.

[14] Stavely A. M. Verifying definite iteration over data structures // IEEE Trans. Software Engineering. — 1995. — Vol. 21, № 6. — P. 506–514.