Моделирование и анализ информационных систем. Т. 24, № 6 (2017), с. 743–754
Modeling and Analysis of Information Systems. Vol. 24, No 6 (2017), pp. 743–754

DOI: 10.18255/1818-1015-2017-6-743-754

UDC 004.052.42

Invariant Elimination of Definite Iterations over Arrays in C Programs Verification

Maryasov I. V.[1], **Nepomniaschy V. A.**, **Kondratyev D. A.**[1]

Received September 8, 2017

Abstract. This work represents the further development of the method for definite iteration verification [7]. It extends the mixed axiomatic semantics method [1] suggested for C-light program verification. This extension includes a verification method for definite iteration over unchangeable arrays with a loop exit in C-light programs. The method includes an inference rule for the iteration without invariants, which uses a special function that expresses loop body. This rule was implemented in verification conditions generator, which is the part of our C-light verification system. To prove generated verification conditions an induction is applied which is a challenge for SMT-solvers. At proof stage the SMT-solver CVC4 is used in our verification system. To overcome mentioned difficulty a rewriting strategy for verification conditions is suggested. A method based on theory extension by new theorems to prove verification conditions is suggested. An example, which illustrates the application of these methods, is considered. The article is published in the authors' wording.

Keywords: C-light, loop invariants, mixed axiomatic semantics, definite iteration, arrays, CVC4, specification, verification, Hoare logic

For citation: Maryasov I. V., Nepomniaschy V. A., Kondratyev D. A., "Invariant Elimination of Definite Iterations over Arrays in C Programs Verification", *Modeling and Analysis of Information Systems*, **24**:6 (2017), 743–754.

On the author:
Ilya V. Maryasov, orcid.org/0000-0002-2497-6484, PhD,
A. P. Ershov Institute of Informatics Systems SB RAS,
6 Akademik Lavrentyev av., Novosibirsk 630090, Russia, e-mail: ivm@iis.nsk.su

Valery A. Nepomniaschy, orcid.org/0000-0003-1364-5281, PhD,
A. P. Ershov Institute of Informatics Systems SB RAS,
6 Akademik Lavrentyev av., Novosibirsk 630090, Russia, e-mail: vnep@iis.nsk.su

Dmitry A. Kondratyev, orcid.org/0000-0002-9387-6735, postgraduate student,
A. P. Ershov Institute of Informatics Systems SB RAS,
6 Akademik Lavrentyev av., Novosibirsk 630090, Russia, e-mail: apple-66@mail.ru

Acknowledgments:
[1]This research is partially supported by RFBR grant 15-01-05974.

Introduction

C program verification is an important task nowadays. A lot of projects (e. g. [3, 4]) propose different solutions. But none of the mentioned above suggests any methods for loop verification without invariants whose construction is a challenge. Therefore, the user has to provide these invariants. In many cases it is a difficult task. Tuerk [13] suggested

to use pre- and post-conditions for while-loops but the user still has to construct them himself.

Our method describes a class of loops, which can be verified without invariants or pre- and post-conditions for loops. It deals with a definite iteration of a special form. We extend our mixed axiomatic semantics of the C-light language [1] with a new rule for verification of such iterations. This extension includes a verification method for definite iteration over unchangeable arrays with a loop exit in C-light programs. The method includes an inference rule for the iteration without invariants, which uses a special function that expresses loop body. This rule was implemented in verification conditions generator, which is the part of our C-light verification system.

At the proof stage, the SMT-solver CVC4 [2] is used. A rewriting strategy for the induction based verification conditions is suggested to prove them in CVC4.

Also an algorithm based on theory extension by special implications to prove verification conditions is suggested. It allows a source formula to be proved successfully. The induction processing approach [12] implemented in CVC4 is too constrained by orientation to inductive data types. The suggested algorithm allows to overcome this difficulty.

1. Definite Iteration and Replacement Operation

The method of loop invariants elimination for definite iteration was suggested in [11]. It includes four cases:

1. Definite iteration over unchangeable data structures without loop exits.
2. Definite iteration over unchangeable data structures with a loop exit.
3. Definite iteration over changeable data structures with a loop exit.
4. Definite iteration over hierarchical data structures with a loop exit.

The first case was considered in [7]. Our paper deals with the second case.
Consider the statement

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

where S is a structure, x is the variable of the type "an element S", v is a vector of loop variables which does not contain x and *body* represents the loop body computation, which does not modify x and S, and which terminates for each $x \in memb(S)$, where $memb(S)$ is the multiset of elements of the structure S. The loop body can contain only the assignment statements, the **if** statements and the **break** statements. Such **for** statement is named a definite iteration.

To express the effect of the iteration let us define a replacement operation $rep(v, S, body, n)$, where $rep(v, S, body, 0) = v$, $rep(v, S, body, i) = body(rep(v, S, body, i-1), s_i)$ for all $i = 1, 2, \ldots, n$ if $\neg empty(S)$.

A number of theorems, which express important properties of the replacement operation, were proved in [11].

The inference rule [10] for definite iterations has the form [8]:

$$\frac{E, SP \vdash \{\exists v'\ P(v \leftarrow v')\ \wedge v = rep(v', S, body)\}\ \mathbf{A};\{Q\}}{E, SP \vdash \{P\}\ \mathbf{for\ x\ in\ S\ do\ v} := \mathbf{body(v, x)\ end\ A};\{Q\}}$$

Here A are program statements after the loop. We use forward tracing: we move from the program beginning to its end and eliminate the leftmost operator (at the top level) applying the corresponding rule of the mixed axiomatic semantics [1] of C-light. E is the environment which contains an information about current function (its identifier, type and body) which is verified, an information about current block and label identifier if **goto** statement occurred earlier. SP is program specification which includes all preconditions, postconditions, and invariants of loops and labeled statements.

2. Definite Iteration over Arrays with a Loop Exit

Let S be a one-dimensional array of n elements. Consider the special case of definite iteration

$$\mathbf{for}\ (\mathbf{i} = \mathbf{0}; \mathbf{i} < \mathbf{n}; \mathbf{i}++)\ \mathbf{v} := \mathbf{body(v, i)}\ \mathbf{end}$$

where $\mathbf{v} := \mathbf{body(v, i)}$ consists of assignment statements, **if** statements (possibly nested) and **break** statements.

In order to generate verification conditions we have to determine v, $body(v, i)$, and the function rep [8].

Let the loop body has the form

$$\begin{gathered}\{\mathbf{x_1} = \mathbf{expr_1}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k});\\ \mathbf{x_2} = \mathbf{expr_2}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k});\\ \ldots\\ \mathbf{x_k} = \mathbf{expr_k}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k});\}\end{gathered}$$

where $expr_j (j = 1, 2, \ldots k)$ are some C-light expressions.

The vector v of loop variable consists of all variables from left parts of assignment statements: $v = (x_1, x_2, \ldots, x_k)$. From the statements before the loop, we can get the initial value of v and obtain the first axiom for rep:

$$rep(v, S, body, 0) = (x_{1_0}, x_{2_0}, \ldots, x_{k_0})$$

where $x_{j_0}, j = 1, 2, \ldots, k$ are the initial values of x_j before the loop execution.

At the next step, we make consecutive substitutions

$$\begin{gathered}x_1 = expr_1(x_1, x_2, \ldots, x_k);\\ x_2 = expr_2(expr_1(x_1, x_2, \ldots, x_k), x_2, \ldots, x_k);\\ \ldots\\ x_k = expr_k(expr_1(x_1, x_2, \ldots, x_k), expr_2(expr_1(x_1, x_2, \ldots, x_k), x_2, \ldots, x_k), \ldots, x_k);\end{gathered}$$

And then in the right parts $rep((x_1, x_2, \ldots, x_k), S, body, i - 1)$ is substituted for x_j.

For each **if** statement of the form $\mathbf{if}\ (\mathbf{e(i, x_1, x_2, \ldots, x_k)})\{\mathbf{A};\}\ \mathbf{else}\ \{\mathbf{B};\}$, where **A** and **B** are compound statements consisting of assignment statements, two axioms are added to the output of verification conditions generator:

$$\forall x_1 \forall x_2 \ldots \forall x_k \; e(i, x_1, x_2, \ldots, x_k) \Rightarrow A^*$$
$$\forall x_1 \forall x_2 \ldots \forall x_k \; \neg e(i, x_1, x_2, \ldots, x_k) \Rightarrow B^*$$

where A^* and B^* are obtained by consecutive substitutions as described above.

The **break** statement could appear at the top level of the loop or in the **if** statement. The first case is obvious, it means that the loop iterates no more than once and all the statements after **break** in the loop body are ignored. Therefore, the function *rep* is defined for $i = 0, 1$.

The second case means that for some j such that $0 < j \leq n$ a loop exit occurs and such j is defined by the condition of the **if** statement. Therefore, for all i such that $j \leq i \leq n$

$$rep((x_1, x_2, \ldots, x_k), S, body, i) = rep((x_1, x_2, \ldots, x_k), S, body, j)$$

In this case the following axiom is added:

$$\forall x_1 \forall x_2 \ldots \forall x_k \; e(i, x_1, x_2, \ldots, x_k) \Rightarrow (A^* \wedge (\forall l \, i < l \Rightarrow A^*))$$

For the case when the **break** statement is located in the **else** statement, the negation of e is used.

For each nested **if** statements of the form $\mathbf{if}\,(\mathbf{e_1}(\mathbf{i}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}))\{\mathbf{A_1}; \mathbf{if}\,(\mathbf{e_2}(\mathbf{i}, \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}))\mathbf{A_2}\,\mathbf{else}\,\mathbf{A_3}; \mathbf{A_4}\}\,\mathbf{else}\,\{\mathbf{B};\}$, where $\mathbf{A_1}, \mathbf{A_2}, \ldots \mathbf{A_4}$ are compound statements consisting of assignment statements, the following axioms are added to the output of verification conditions generator:

$$\forall x_1 \forall x_2 \ldots \forall x_k \; ((e_1(i, x_1, x_2, \ldots, x_k) \Rightarrow A_1^*) \wedge e_2(i, x_1, x_2, \ldots, x_k)) \Rightarrow (A_2; A_4)^*$$
$$\forall x_1 \forall x_2 \ldots \forall x_k \; ((e_1(i, x_1, x_2, \ldots, x_k) \Rightarrow A_1^*) \wedge \neg e_2(i, x_1, x_2, \ldots, x_k)) \Rightarrow (A_3; A_4)^*$$
$$\forall x_1 \forall x_2 \ldots \forall x_k \; \neg e_1(i, x_1, x_2, \ldots, x_k) \Rightarrow B^*$$

where A_1^*, $(A_2; A_4)^*$, $(A_3; A_4)^*$ and B^* are obtained by consecutive substitutions as described above. For multiple nested **if** statements we make axioms in the similar way.

$\forall i (1 \leqslant i \leqslant k)$ rep_i is automatically generated to simplify the proof of rep-based verification conditions in CVC4. Note that $\forall i (1 \leqslant i \leqslant k)$ rep_i corresponds to variable x_i. This generation is based on substitution of the i-th *rep* by rep_i.

3. Extending Theory to Prove Verification Conditions

To prove by induction some proposition $\forall n \; P(n)$ we use Leino approach [6]. It is to add an extra axiom (induction step) of the form $\forall j \; P(j) \Rightarrow P(j + 1)$ and to modify the verification condition by adding a base case of induction $P(1)$. In our case of definite iteration over unchangeable one-dimensional arrays the inductive variable is the length of array. Therefore, the verification conditions generator is able to rewrite the verification condition which contains a *rep* function automatically.

Let us consider the following method of proving formula ϕ, which has the form:

$$\forall x_1 x_2 ... x_{m-1} x_m \quad a(x_1 x_2 ... x_{m-1} x_m) \implies b(x_1 x_2 ... x_{m-1} x_m), \tag{1}$$

where

$$a = H_1(x_{K_{1_1}} x_{K_{1_2}} ... x_{K_{1_{l_1-1}}} x_{K_{1_{l_1}}}) \wedge H_2(x_{K_{2_1}} x_{K_{2_2}} ... x_{K_{2_{l_2-1}}} x_{K_{2_{l_2}}}) \wedge ...$$
$$\wedge H_{n-1}(x_{K_{n-1_1}} x_{K_{n-1_2}} ... x_{K_{n-1_{l_{n-1}-1}}} x_{K_{n-1_{l_{n-1}}}}) \wedge H_n(x_{K_{n_1}} x_{K_{n_2}} ... x_{K_{n_{l_n-1}}} x_{K_{n_{l_n}}}), \quad (2)$$

$K_1 \cup K_2 \cup ... \cup K_{n-1} \cup K_n = \{1, ..., m\}$ and $|K_1| = l_1, |K_2| = l_2, ..., |K_{n-1}| = l_{n-1}, |K_n| = l_n$.

The formula ϕ and set of axioms and theorems are the input arguments of the algorithm.

Let us consider the set of axioms and theorems:

$$\forall y_{11} y_{12} ... y_{1p_1-1} y_{1p_1} \ f_1, \quad \forall y_{21} y_{22} ... y_{2p_2-1} y_{2p_2} \ f_2 ...$$
$$\forall y_{q-11} y_{q-12} ... y_{q-1p_{q-1}-1} y_{q-1p_{q-1}} \ f_{q-1}, \quad \forall y_{q1} y_{q2} ... y_{qp_q-1} y_{qp_q} \ f_q \quad (3)$$

The message "The formula ϕ is true" or "unknown" is the output value of the algorithm.

1. Let i:=1.

2. Let us consider $\forall y_{i1} y_{i2} ... y_{ip_i-1} y_{1p_i} \ f_i$.

   If the structure of f_i is of the form of $c(y_{i1} y_{i2} ... y_{ip_i-1} y_{1p_i}) \implies d(y_{i1} y_{i2} ... y_{ip_i-1} y_{1p_i})$ then go to the step 3 else go to the step 9.

3. If the structure of c is in the form of

$$G_1(y_{R_{1_1}} y_{R_{1_2}} ... y_{R_{1_{s_1-1}}} y_{R_{1_{s_1}}}) \wedge G_2(y_{R_{2_1}} y_{R_{2_2}} ... y_{R_{2_{s_2-1}}} y_{R_{2_{s_2}}}) \wedge ...$$
$$\wedge G_{t-1}(y_{R_{t-1_1}} y_{R_{t-1_2}} ... y_{R_{t-1_{s_{t-1}-1}}} y_{R_{t-1_{l_{t-1}}}}) \wedge G_t(y_{R_{t_1}} y_{R_{t_2}} ... y_{R_{t_{s_t-1}}} y_{R_{t_{l_t}}}), \quad (4)$$

   where $t \leqslant n$, $R_1 \cup R_2 \cup ... \cup R_{t-1} \cup R_t = \{i1, i2, ..., ip_i\}$ and $|R_1| = s_1, |R_2| = s_2, ..., |R_{t-1}| = s_{t-1}, |R_t| = s_t$ then go to the step 4 else go to the step 9.

4. Let us consider a subformula of ϕ $a' = H'_1(x') \wedge H'_2(x') \wedge ... \wedge H'_{n-1}(x') \wedge H'_n(x')$ where each conjunct $H'_i(x')$ results from conjunct $H_i(x_{K_{i_1}} x_{K_{i_2}} ... x_{K_{i_{l_i-1}}} x_{K_{i_{l_i}}})$ by substitution of all occurrences of $x_{K_{i_1}} x_{K_{i_2}} ... x_{K_{i_{l_i-1}}} x_{K_{i_{l_i}}}$ by unique identifier x'.

   Let us consider subformula $c' = G'_1(x') \wedge G'_2(x') \wedge ... \wedge G'_{t-1}(x') \wedge G'_t(x')$ where each conjunct $G'_i(x')$ results from conjunct $G_i(y_{R_{i_1}} y_{R_{i_2}} ... y_{R_{i_{s_i-1}}} y_{R_{i_{s_i}}})$ by substitution of all occurrences of $y_{R_{i_1}} y_{R_{i_2}} ... y_{R_{i_{s_i-1}}} y_{R_{i_{s_i}}}$ by unique identifier x'.

   Let us consider bijection set $\{e_1, e_2, ..., e_{v-1}, e_v\}$ where $\forall i$ e_i is bijection from the set $\{1, .., t\}$ of conjunct indexes of subformula c to the subset U of conjunct indexes of a. Note that $e_i(w) = u \iff G'_w(x')$ is syntactically equal to $H'_u(x')$.

5. Let $j := 1$.

6. Generate a table of correspondence w between variables y of formula c and variables x of the following formula

$$H_{e_j(1)} \wedge H_{e_j(2)} \wedge ... \wedge H_{e_j(t-1)} \wedge H_{e_j(t)}. \quad (5)$$

using matching between G_v and $H_{e_j(v)}$ subformulas. If there is not such correct table w then go to the step 8 else go to the step 7.

7. Let us consider the following formula

$$d'(w(y_{i1})w(y_{i2})...w(y_{ip_i-1})w(y_{1p_i})) = d[y_{i1} \leftarrow w(y_{i1}), y_{i2} \leftarrow w(y_{i2}), ..., y_{ip_i-1} \leftarrow w(y_{ip_i-1}), y_{1p_i} \leftarrow w(y_{1p_i})] \quad (6)$$

It has been generated using simultaneous substitution of $y_{i1}, y_{i2}, ..., y_{ip_i-1}, y_{1p_i}$ by corresponding variables $w(y_{i1}), w(y_{i2}), ...w(y_{ip_i-1}), w(y_{1p_i})$.

Let us consider the following formula

$$\forall x_1 x_2 ... x_{m-1} x_m \quad a \wedge d' \implies b \quad (7)$$

It may be proved using Leino approach based on induction. The SMT solver CVC4 may be used in such case. If such proving results in "unsat" then go to the step 11 else go to the step 8.

8. Let $j := j + 1$. If $j \leqslant v$ then go to the step 6 else go to the step 9.

9. Let $i := i + 1$. If $i \leqslant q$ then go to the step 2 else go to the step 10.

10. The algorithm results in "unknown".

11. The algorithm results in "The formula ϕ is true".

The suggested algorithm allows the theory for proving verification conditions to be extended by new theorems. They may be used to simplify proof.

4. Example: Array Searching Program

Let us demonstrate the application of our method. Consider the following function **search_count**. For a given integers *key* and *entr* it returns 1 if not less than *entr* elements of the given array of integers *arr* are equal to *key*, where *length* is *arr* length. Otherwise the function returns 0.

The annotated (in SMT-LIB v2 syntax of CVC4) C-light [5] program has the form:

```
/* (assert (and (> length 0) (> entr 0))) */
int search_count(int* arr, int length, int key, int entr){
    auto int result = 0, cnt = 0, i;
    for (i = 0; i < length; i++){
        if ((key == arr[i]) && (entr > (cnt + 1))){
            cnt++;
        }
        else if ((key == arr[i]) && (entr == (cnt + 1))){
            cnt++;
            result = 1;
```

```
            break;
        }
    }
    return result;
}
/* (assert (and
             (=> (<= entr (COUNT length arr key entr)) (= result 1))
             (=> (> entr (COUNT length arr key entr)) (= result 0))) */
```

The logical function **COUNT** returns the number of occurrences of *key* in *arr* from 0 to $length - 1$. Note that in CVC4 all functions must be total, so we also need to consider the case for $i \leqslant 0$. The other axioms describe the common case and define **COUNT** recursively. Let us consider some of axioms of the function **COUNT**:

```
(declare-fun COUNT (Int (Array Int Int) Int Int) Int)

(assert (forall ((i Int) (arr (Array Int Int)) (key Int) (entr Int))
   (=>
      (and (< 0 i) (= (select arr (- i 1)) key))
      (= (COUNT i arr key entr) (+ (COUNT (- i 1) arr key entr) 1)))))

(assert (forall ((i Int) (arr (Array Int Int)) (key Int) (entr Int))
   (=>
       (and (< 0 i) (not (= (select arr (- i 1)) key)))
       (= (COUNT i arr key entr) (COUNT (- i 1) arr key entr)))))
```

In *rep* function $v = (cnt, result)$ and its initial value before the iteration is $(0, 0)$. The approach from section 2 allows axioms of *rep* to be generated. Let us consider some of these axioms:

```
(declare-fun rep1 (Int (Array Int Int) Int Int) Int)

(declare-fun rep2 (Int (Array Int Int) Int Int) Int)

(assert (forall ((i Int) (arr (Array Int Int)) (key Int) (entr Int))
    (=>
        (and
             (< 0 i)
             (= key (select arr (- i 1)))
             (> entr (+ (rep1 (- i 1) arr key entr) 1)))
        (= (rep1 i arr key entr) (+ (rep1 (- i 1) arr key entr) 1)))))

(assert (forall ((i Int) (arr (Array Int Int)) (key Int) (entr Int))
    (=>
         (and
              (< 0 i)
              (not (= key (select arr (- i 1))))
```

```
            (> entr (+ (rep1 (- i 1) arr key entr) 1)))
        (= (rep1 i arr key entr) (rep1 (- i 1) arr key entr)))))

(assert (forall ((i Int) (arr (Array Int Int)) (key Int) (entr Int))
    (=>
        (and
            (< 0 i)
            (> entr (+ (rep1 (- i 1) arr key entr) 1)))
        (= (rep2 i arr key entr) (rep2 (- i 1) arr key entr)))))

(assert (forall ((i Int) (arr (Array Int Int)) (key Int) (entr Int))
    (=>
        (and
            (< 0 i)
            (= key (select arr (- i 1)))
            (= entr (+ (rep1 (- i 1) arr key entr) 1)))
        (= (rep2 i arr key entr) 1))))
```

CVC4 is the SMT-solver but our task is to check a validity of verification conditions, not satisfiability. Therefore, the verification conditions generator produces the negation of the verification condition:

```
(assert (not (forall ((length Int) (arr (Array Int Int))
                      (key Int) (entr Int) (result Int))
    (=>
        (and (> length 0) (> entr 0) (= result (rep2 j arr key entr)))
        (and
            (=>
                (<= entr (COUNT length arr key entr))
                (= result 1))
            (=>
                (> entr (COUNT length arr key entr))
                (= result 0)))))))
```

And then we expect the answer “unsat” which means that the negation is unsatisfiable therefore the verification condition is true.

However, SMT solvers do not support proofs by induction, which appears inevitably in our verification method. In this example we get the answer “unknown” which means that CVC4 is not able to determine whether the formula is satisfiable or not. Rustan Leino suggested a rewriting strategy and a heuristic for when to apply it to verify simple inductive theorems [6].

The simplification of verification condition allows it to be considered as a conjunction of the first conjunct

```
(assert (forall ((length Int) (arr (Array Int Int))
                 (key Int) (entr Int) (result Int))
                 (=>
```

```
                    (and
                       (> length 0)
                       (> entr 0)
                       (= result (rep2 length arr key entr))
                       (> entr (COUNT length arr key entr)))
   (= result 0))))
```

and the second conjunct

```
(assert (forall ((length Int) (arr (Array Int Int))
                 (key Int) (entr Int) (result Int))
                (=>
                   (and
                       (> length 0)
                       (> entr 0)
                       (= result (rep2 length arr key entr))
                       (<= entr (COUNT length arr key entr)))
                   (= result 1))))
```

They are referred to as the first and the second part of verification condition.

Algorithm from section 3 allows the first conjunct to be proved. The theory has been extended by the following theorem:

```
(assert (forall ((j Int) (arr (Array Int Int))
                 (key Int) (entr Int))
                 (=>
                    (and
                       (> j 0)
                       (> entr 0)
                       (> entr (COUNT j arr key entr)))
                    (> entr (rep1 j arr key entr)))))
```

It has been proved using CVC4. This proof is based on Leino approach.

Note that renaming variables can result in equality of premises of this formula and some of premises of the first part of the verification condition. Thus, it can result in extension of premises of the first part of the verification condition by conclusion of the considered formula.

Leino approach allows extended verification condition to be proved using CVC4. The base case of induction is trivial. Let us consider the extension of theory by negation of the induction step:

```
(assert (not (forall ((length Int))
     (=>
           (forall ((arr (Array Int Int)) (key Int)
                    (entr Int) (result Int))
                 (=>
                     (and
```

```
                    (> length 0)
                    (> entr 0)
                    (= result (rep2 length arr key entr))
                    (> entr (COUNT length arr key entr))
                    (> entr (rep1 length arr key entr)))
                (= result 0)))
        (forall ((arr (Array Int Int)) (key Int)
                 (entr Int) (result Int))
                (=>
                    (and
                        (> (+ length 1) 0)
                        (> entr 0)
                        (= result (rep2 (+ length 1) arr key entr))
                        (> entr (COUNT (+ length 1) arr key entr))
                        (> entr (rep1 (+ length 1) arr key entr))
                    (= result 0)))))))
```

This theorem has been proved using CVC4.

The proof of the second part of verification condition is similar to the proof of the first one. The theory has been extended by the following theorem:

```
(assert (forall ((j Int) (arr (Array Int Int))
                 (key Int) (entr Int))
                 (=>
                    (and
                        (> j 0)
                        (> entr 0)
                        (<= entr (COUNT j arr key entr)))
                  (<= entr (rep1 j arr key entr)))))
```

Leino approach allows it to be proved using CVC4. Also this approach has been applied at the step 7 of the algorithm from section 3. Thus, this algorithm allows the second part of verification condition to be proved.

Therefore the verification condition has been proved.

5. Conclusion

This paper represents an extension of our system [9] for C-light program verification. In the case of definite iteration over unchangeable arrays with a loop exit, this extension allows us to generate verification conditions without loop invariants. This generation is based on the described inference rule for the C-light **for** statement which introduces the replacement operation. It expresses definite iteration in the special form described in the paper.

The suggested rewriting strategy for induction-based formulas allowed us to prove obtained verification conditions using CVC4 [2]. The proposed algorithm of extending the theory allows verification condition to be proved. Proving formula $a \implies b$ is the

goal of the algorithm execution. Note that this algorithm is based on using implication $c \implies d$ where c is a subformula of a after variables renaming. The proof of formula $a \implies b$ is reduced to proof of formula $(a \wedge d) \implies b$. Using unique identifier allows a and c to be matched. Also the table of correspondence between variables of a and c is created by the algorithm. Note that this table is used for renaming variables of formula d.

The rewriting strategy allowed CVC4 to prove the partial correctness of the example from [7]. It iterates over an array of integers and for a given integer computes the number of its occurrences in this array. Another successfully proved example is the program which for a given integer finds its first occurrence in the given array of integers [8].

We plan to consider the case of elimination of loop invariant for changeable data structures and to verify classical array sorting programs without invariants.

References

[1] I. S. Anureev, I. V. Maryasov, V. A. Nepomniaschy, "C-programs Verification Based on Mixed Axiomatic Semantics", *Automatic Control and Computer Sciences*, **45**:7 (2011), 485–500.

[2] C. Barrett, C. L., Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli, "CVC4", 23rd Int. Conf. CAV, *Lecture Notes in Computer Science*, **6806** (2011), 171–177.

[3] E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, S. Tobies, "VCC: A Practical System for Verifying Concurrent C", 22nd Int. Conf. TPHOLs, *Lecture Notes in Computer Science*, **5674** (2009), 23–42.

[4] J.-C. Filliâtre, C. Marché, "Multi-prover Verification of C Programs", 6th ICFEM, *Lecture Notes in Computer Science*, **3308** (2004), 15–29.

[5] D. A. Kondratyev, "The Extension of the MetaVCG Approach by Semantic Mark-up Concept", *Int. Workshop-conf. "Tools & Methods of Program Analysis"*, St. Petersburg, 2015, 107–118.

[6] K. R. M. Leino, "Automating Induction with an SMT Solver", 13th Int. Conf. VMCAI, *Lecture Notes in Computer Science*, **7148** (2012), 315–331.

[7] I. V. Maryasov, V. A. Nepomniaschy, "Loop Invariants Elimination for Definite Iterations over Unchangeable Data Structures in C Programs", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 773–782.

[8] I. V. Maryasov, V. A. Nepomniaschy, D. A. Kondratyev, "Verification of Definite Iteration over Arrays with a Loop Exit in C Programs", *System Informatics*, 2017, № 10, 57–66.

[9] I. V. Maryasov, V. A. Nepomniaschy, A. V. Promsky, D. A. Kondratyev, "Automatic C Program Verification Based on Mixed Axiomatic Semantics", *Automatic Control and Computer Sciences*, **48**:7 (2014), 407–414.

[10] M. Moriconi, R. L. Schwarts, "Automatic Construction of Verification Condition Generators From Hoare Logics", Automata, Languages and Programming, *Lecture Notes in Computer Science*, **115** (1981), 363–377.

[11] V. A. Nepomniaschy, "Verification of Definite Iteration over Hierarchical Data Structures", FASE/ ETAPS, *Lecture Notes in Computer Science*, **1577** (1999), 176–187.

[12] A. Reynolds, V. Kuncak, "Induction for SMT solvers", 16th Int. Conf. VMCAI, *Lecture Notes in Computer Science*, **8931** (2015), 80–98.

[13] T. Tuerk, "Local Reasoning about While-Loops", *VSTTE*, 2010, 29–39.

Марьясов И. В., Непомнящий В. А., Кондратьев Д. А., "Элиминация инвариантов финитных итераций над массивами при верификации Си программ", *Моделирование и анализ информационных систем*, **24**:6 (2017), 743–754.

DOI: 10.18255/1818-1015-2017-6-743-754

Аннотация. Данная работа представляет дальнейшее развитие метода верификации финитной итерации [7]. Он расширяет метод смешанной аксиоматической семантики [1], предложенный для верификации C-light программ. Это расширение включает метод верификации для финитной итерации над неизменяемыми массивами с выходом из цикла в C-light программах. Метод содержит правило вывода для итерации без инвариантов, которое использует специальную функцию, выражающую действие тела цикла. Данное правило было реализовано в генераторе условий корректности, являющемся частью нашей системы верификации C-light программ. Для доказательства порождённых условий корректности применяется метод математической индукции, вызывающий сложности у SMT-решателей. В нашей системе верификации на стадии доказательства используется SMT-решатель CVC4. Для преодоления упомянутой трудности применяется стратегия переписывания условий корректности. Для доказательства условий корректности предложен метод, основанный на расширении теории новыми теоремами. Рассмотрен пример, иллюстрирующий применение данных методов. Статья публикуется в авторской редакции.

Ключевые слова: C-light, Си, инварианты циклов, смешанная аксиоматическая семантика, финитная итерация, массивы, CVC4, спецификация, верификация, логика Хоара

Об авторе:
Марьясов Илья Владимирович, orcid.org/0000-0002-2497-6484, канд. физ.-мат. наук,
Институт систем информатики им. А. П. Ершова СО РАН,
пр. Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: ivm@iis.nsk.su

Непомнящий Валерий Александрович, orcid.org/0000-0003-1364-5281, канд. физ.-мат. наук,
Институт систем информатики им. А. П. Ершова СО РАН,
пр. Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: vnep@iis.nsk.su

Кондратьев Дмитрий Александрович, orcid.org/0000-0002-9387-6735, аспирант,
Институт систем информатики им. А. П. Ершова СО РАН,
пр. Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: apple-66@mail.ru

Благодарности:
[1]Работа частично поддержана грантом РФФИ 15-01-05974.