# Program Invariants Generation over Polynomial Ring using Iterative Methods.

**Sergii Kryvyi, Oleksandr Maksymets**

***Abstract:*** *A solution for program polynomial invariant generation problem is presented. An iteration upper approximation method that was successfully applied on free algebras in this paper was adopted for polynomial ring. Set of invariants is interpreted as an ideal over polynomial ring. Relationship and intersection problems solution are proposed. Intersection of Gröbner basis is applied to solve intersection problem. Inverse obligatory is applied to solve relationship problem.*

***Keywords:*** *verification theory, invariant generation, polynomial ring.*

***ACM Classification Keywords****: D.2.4 [Software/Program Verification]: Assertion checkers; F.3.1 [Specifying and Verifying and Reasoning about Programs]: Invariants; I.2.2 [Automatic Programming]: Program verification; B.6.3 [Design Aids] : Verification; D.4.5 [Reliability]: Verification.*

## Introduction

Scope of this paper, as the name implies, refers to software verification problems. Software verification is one of the most difficult step of software development. Challenge of this step is that the developer, except knowledge of software development, require knowledge of methods of modern algebra, logic, combinatorics, number theory, and other related areas. In addition to these subjective factors there are also objective factors related to fact that currently available methods of verification are not at a sufficient level for software development. It turns out impossible to verify the systems of industrial size. The general picture, observed at this point is that the complexity of software is growing, and methods of analysis are significantly behind.

Software systems are divided into two classes: *class functional systems* and *class reactive systems* nowadays. The first class includes systems that have to work in finite time and have an input and output (ie, function computation). The second systems should work potentially infinite time, reacting in its compliance to internal and external events. Unfortunately, methods of properties verification of first systems do not apply to verify the properties of second systems. That led to the fact that for both classes were developed different methods.

Research in the field of formal verification methods develops in two directions: (1) *deductive verification* and *theorem proving* [1; 2; 3] and (2) *algorithmic verification* with check procedures, such as verification procedures in the satisfiability of logical formulas on model [4; 5], symbolic modeling [6] or symbolic execution paths [7].

If the verification of reactive systems is reduced to soluble properties of the finite automata theory, verification of functional programs is reduced to the problem of proving theorems in languages dynamic program logic or predicate logic of first order. One of the first language of program logic was proposed by Hoare [8] and named in his honor Hoare logic. The Hoare method is based on methods of finding and generating invariants of program cycles with further usage of deductive proof methods of assertions (theorems) about programs properties. This method stills currently principal at verification of software functional systems . *Verification problem* of functional systems refers to a problem on statements of $\varphi$ and $\psi$ about $P$ program to prove truth of a statement $\psi$ about output values of program $P$, asserting that the values of the input variables satisfy $\varphi$.

Solution of this problem is quite difficult and requires careful program analysis. It can be greatly facilitated if invariants for a given program state are known. An *invariant* of program at a state is an assertion that is true for any memory state reaching the program state. The use of invariants in the verification is reduced to the problem of invariant relations generation for a given structure of the program (for example, loop invariants). Further, based on the constructed set of invariants prove required predicate (postcondition) what existence would guarantee the

correctness of the program. Methods used to solve this problem called *data flow program analysis methods*. The appearance of these techniques have been associated with with the creation of high-quality and reliable software.

After verification of programs based on Floyd-Hoare-Dijkstra's inductive approval, using pre and post conditions and loop invariants [10] in the seventies (Wegbreit, 1974, 1975; German and Wegbreit, 1975; Katz and Manna, 1976; Cousot and Cousot, 1976; Suzuki and Ishihata, 1977; Dershowitz and Manna, 1978) there was silent period in this domain. Recently significant progress in development of automated provers, SAT solvers and models checkers had place. All mentioned tools use assertions as input data. Therefore, during last years problem of finding assertion for programs became actual again.

Investigations in this area were active also in USSR, especially in Kiev and Novosibirsk, during 70-th and 80-th. In result, effective invariant generation algorithm were built. Data algebras that algorithms affected were absolutely free algebra[13][21], groups, semi-groups, Abelian groups and Abelian semigroups, vector space [12], polynomial ring[18] [19].

We interpret program as $U$-$Y$ schema on algebra of polynomials. Iterative algorithms applied for free algebras and vector space was adopted in this paper for polynomial space [16] .

An *invariant* of $U$-$Y$ schema at state is an assertion that is true for any memory state reaching the schema state. Further we use term state as state(node) of $U$-$Y$ schema. Proposed approach generates basis of invariants for each program state taking in consideration passed invariants for initial state.

This work was inspired by related work done in generating invariants for polynomial space using Gröbner basis (Müller-Olm and Seidl, 2006 [22] , Sankaranarayanan et al., 2004, Rodriguez-Carbonell and Kapur, 2004 [23]). We argue some opportunity to discover more invariants using iteration method, that looks promising on smaller problems.

## Preliminaries

The concept of the $U$-$Y$ scheme considered in this section, is a the most common mathematical model of the program [14], which we receive after ignorance of specific program and information environment.

**Definition 1.** *(Memory) Let $R = \{r_1, \ldots, r_m\}$ be the set of variables (the memory) and $T(R)$ the $\Omega$-algebra of terms over $R$ interpreted on $D$. Where $D$ is a $\Omega$-universal algebra and we call it an data algebra. We consider $(D, \Omega)$ as polynomial ring $\Re[r_1, \ldots, r_m]$.*

Consider the set of expressions of the form $\pi(t_I, \ldots, t_n)$ , where $r$ is the symbol of an n-place predicate from $\pi$, $t_I, \ldots, t_n \in T(R)$.

Each of these expressions defines a certain assertion on the set variable values, whose value for a given memory state. Let $U(R, \Omega, \Pi)$ be the set of all such assertions and $U$ the set of propositional boolean functions of conditions from $U(R, \Omega, \Pi)$. The elements of the set $U(R, \Omega, \Pi)$ are termed basic conditions over the memory $R$, and the elements of the set $U$ are called elementary conditions over the memory $R$.

The assignment operator is an expression of the form $Y = \{r_i' = p(r_1, \ldots, r_n)\}$. Each assignment operator $y$ specifies a certain transformation on values of the memory.

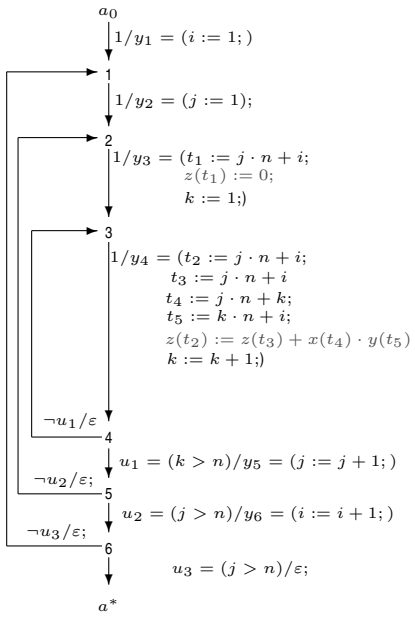
**Definition 2.** *($U$-$Y$ scheme) The couple $(U(R, \Omega, \Pi), Y(R, \Omega))$ is called the standard basis over the memory $R$ defined by the signature $(\Omega, \Pi)$. If $U \subseteq U(R, \Omega, \Pi)$, $Y \subseteq Y(R, \Omega)$, then the standard $U$-$Y$ program schema over memory is the set $A$ of schema states together with the set of transitions $S \subseteq A \times U \times Y \times A$.*

**Definition 3.** *(Language) $L$ is the language in which assertions about the properties of the memory values are written. Regarding the language $L$ we assume that any sentence in $L$ is representable by a formula $u(r_1, \ldots, r_m)$ in the language of the first-order predicate calculus containing the free variables $r_1, \ldots, r_m$ and interpreted on the data domain $D$. The signature of this calculus contains all the symbols of the signatures $\Omega$ and $\Pi$.*

As an example of $U$-$Y$ schema interpretation we consider program of two matrices $n \times n$ multiplication. Source code provided in pseudo-code :

```
MULT(A, B, n)
X, Y, Z : array[1 : n]
i, j, k, n : integer
  for i = 1 to n do
      for j = 1 to n do
          z(i, j) := 0
          for k = 1 to n do
              z(i, j) := z(i, j) + x(i, k) * y(k, j)
          end for
      end for
  end for
```

$U$-$Y$ scheme of the program $MULT(A, B, n)$ will look like:




$A = \{a_0, a_1, \ldots, a_*\}$ is a set of $U$-$Y$ schema nodes. $N_{a_i}$ is a basis of assertions that we have in node $a_i$ on current step of the method. $N_{a_0}, N_{a_1}, \ldots, N_{a_*}$ are assertion bases for nodes of $U$-$Y$ schema. $U$ is a set of conditions with elements that have structure $u = (p\,(r_1, \ldots, r_n) = 0)$, where $p\,(r_1, \ldots, r_n) \in \Re[r_1, \ldots, r_n]$. Set of assignments $Y$ has next element's structure $r_i := p(r_1, \ldots, r_n)$, where $p\,(r_1, \ldots, r_n) \in \Re[r_1, \ldots, r_n]$.

**Definition 4.** *(Algebraic Assertions) An algebraic assertion $\psi$ is an assertion of the form $\bigwedge_i p_i(r_1, \ldots, r_m) = 0$ where each $p_i \in \Re[r_1, \ldots, r_m]$. The degree of an assertion is the maximum among the degrees of the polynomials that make up the assertion.*

**Definition 5.** *(Ideals) A set $I \subseteq \Re[r_1, ..., r_n]$ is an ideal, if and only if*

*1.* $0 \in I$.

*2. If* $p_1, p_2 \in I$ *then* $p_1 + p_2 \in I$.

*3. If* $p_1 \in I$ *and* $p_2 \in \Re[r_1, ..., r_n]$ *then* $p_1 \cdot p_2 \in I$ *[15].*

An ideal generated by a set of polynomials $N$, denoted by $((P))$ is the smallest ideal containing $N$. Equivalently,

$$((P)) = \{g_1 p_1 + \ldots + g_m p_m | g_1, \ldots, g_m \in R[r_1, \ldots, r_n], p_1, \ldots, p_m \in P\}$$

An ideal $I$ is said to be finitely generated if there is a finite set $N$ such that $I = ((P))$. A famous theorem due to Hilbert states that all ideals in $\Re[r_1, \ldots, r_n]$ are finitely generated. As a result, algebraic assertions can be seen

as the generators of an ideal and vice-versa. Any ideal defines a variety, which is the set of the common zeros of all the polynomials it contains.

(Ideals intersection) A set $K$ is an intersection of ideals $I = \{f_1, \ldots, f_l\}$ and $J = \{g_1, \ldots, g_m\}$ if

$$K = \{s(r_1, \ldots, r_n) | s(r_1, \ldots, r_n) = \sum_{i=1}^{l} p_i \cdot f_i = \sum_{j=1}^{m} q_j \cdot g_j$$

where $p_1, \ldots, p_l, q_1, \ldots, q_m \in \Re[r_1, \ldots, r_n]\}$.

**Theorem 1** (Ideal intersection). *Let $I$ and $J$ be ideals in $R[r_1, \ldots, r_2]$.*

$$I \cap J = (t \cdot I + (1-t) \cdot J) \cap \Re[r_1, \ldots, r_2] \tag{1}$$

*where $t$ is a new variable [15].*

**Proof.** Note that $tI + (1-t)\, J$ is an ideal in $\Re[x_1, \ldots, x_n, t]$. To establish the desired equality, we use strategy of proving by inclusion in both directions.

Suppose $f \in I \cap J$. Since $f \in I$, we have $t \cdot f \in tI$. Similarly, $f \in J$ implies $(1-t) \cdot f \in (1-t)\, J$. Thus, $f = t \cdot f + (1-t) \cdot f \in tI + (1-t)\, J$. Since $I, J \subset \Re[x_1, \ldots, x_n]$.

To establish inclusion in the opposite direction, suppose $f \in (tI + (1-t)\, J) \cap \Re[r_1, \ldots, r_n]$. Then $f\,(r) = g\,(r,t) + h\,(r,t)$, where $g\,(r,t) \in tI$ and $h\,(r,t) \in (1-t)\, J$. First set $t = 0$. Since every element of $tI$ is a multiple of $t$, we have $g\,(r,0) = 0$. Thus, $f\,(r) = h\,(r,0)$ and hence, $f\,(r) \in J$. On the other hand, set $t = 1$ in the relation $f\,(r) = g\,(r,t) + h\,(r,t)$. Since every element of $(1-t)\, J$ is a multiple of $1-t$, we have $h\,(r,1) = 0$. Thus, $f\,(r) = g\,(r,1)$ and, hence, $f\,(r) \in I$. Since $f$ belongs to both $I$ and $J$, we have $f \in I \cap J$. Thus, $I \cap J \supset (t \cdot I + (1-t) \cdot J) \cap \Re[r_1, \ldots, r_2]$ and this completes the proof. ■

Generator finds invariants in language $L$. Generator consist of 3 components: function $ef : L \times U \times Y \to L$ named effect, semi-lattice structure of assertion's set on $L$ and iterative algorithm description.

Function $ef$ transform assertion $u$ that is true before execution of assign operator $y \in Y$ in condition $ef(u, y)$ that is true after execution.

Since $N$ and $ef(N, u, y)$ represent some of the predicates on the set $D$, then they can be considered as relations on $D$, defined by these predicates. Then the power set $B(L)$ can be presented in the form of a lattice with respect to set-theoretic operations of union and intersection, which contains zero $\emptyset$ and one $L$. Expressions $ef(N, u, y) \bigcap (\bigcup) ef(N', u', y')$ in this case refers to the intersection (union) of relations at $D$.

The number of different possible paths in the program (in the presence of at least one cycle) can be infinite. Then the process of building assertion basis of state $a$ can also become infinite. However, let consider states $a_i$ of $U$-$Y$ program denote $A$ that have transitions $(a_i, u_i, y_i, a)$ with $a$ and $N_i$, basis of assertions for state $a_i$. Then equality has place $N_a = \bigcap\limits_{i=1}^{k} ef(N_i, u_i, y_i)$ is assertion basis for state $a$ and candidate for invariant basis in this state. After assertion basis is not changing with iterations then assertion basis becomes invariant basis. This fact is the starting point for the construction of two iterative methods for generating invariants and has strict proof [20].

Upper Approximation Method (UAM) is the iterative process [11] that is defined by the recurrence relation

$$N_a^{(n)} = N_a^{(n-1)} \bigcap \Big( \bigcap_{(a', u, y, a) \in S} ef(N_{a'}^{(n-1)}, u, y) \Big), n > 0, a, a' \in A \tag{2}$$

and the initial approximation is defined by $N_{a_0}^{(0)} = \{u\}$ and some collection of simple paths that cover whole set of states. The evaluation of the initial approximation carried out along these paths, starting with $N_{a_0}^{(0)}$. If for some $a' \in A$ is already known $N_{a'}^{(0)}$ and a transition $(a, u, y, a')$ belongs to one of considered paths and $N_a^{(0)}$ is undefined then we set $N_a^{(0)} = ef(N_{a'}^{(0)}, u, y)$. From equality 2 for every $a \in A$ we have inclusions

$N_a^{(0)} \supseteq N_a^{(1)} \supseteq \ldots$. Therefore, required invariant basis can be obtained only after the stabilization of the iterative process. Since invariants search process can be infinite, that is UAM disadvantage. However, if the effective completion generates more complete basis of invariants than the lower approximation method [11].

Methods for generating invariants shows that finiteness of the invariants search process is closely connected with language terms of $L$. The most common languages are conditions languages such as equalities and inequalities based on that fact that any programming language virtually includes predicates of equality and inequality.

## Algorithm of UAM

Let provide listing in pseudo-code of UAM from [11].

**Input:** U-Y scheme $A$, $N_0$ is basis of invariant for state $a_0$.

**Output:** $N_{a_0}, N_{a_1}, \ldots, N_{a_*}$ invariant basis for every state.

$N_{a_0} := N_0$
ToVisit.push($a_0$)
Visited := {}
**while** ToVisit $\neq \emptyset$ **do**
  c := ToVisit.pop()
  Visited := Visited $\bigcup$ c
  **for all** $(c, y, a')$ **do**
    **if** Not $a'$ in Visited **then**
      $N_{a'}$:=$ef(N_c, y)$
      ToVisit.push($a'$))
    **end if**
  **end for**
**end while**
ToVisit := $A/\{a_0\}$
**while** ToVisit $\neq \emptyset$ **do**
  $c$ := ToVisit.pop()
  **if** $N_c \neq \emptyset$ **then**
    $N := N_c$
    **for all** $(a', y, c)$ **do**
      $N := N \bigcap ef(N_{a'}, y)$
    **end for**
    **if** $(N \neq N_c)$ **then**
      $N_c := N$
      ToVisit := ToVisit $\bigcup\{a|$ for all $(c, y, a)\}$
    **end if**
  **end if**
**end while**

Therefore, relationship($ef(\ldots)$), intersection($\bigcap$) and stabilization problems should be solved before algorithm application for polynomial algebra.

**Relationship Problem.** Given the assertions algebraic basis $N_a$ and the operator $y \in Y$. Construct the algebraic assertions basis $ef(N_a, y)$ that implies after assignment operator.

We consider particular case of invertible assignments to solve relationship problem. In this case equality that assignment presents $r_i' = p(r_1, \ldots, r_n)$ can be transform as $r_i = p(r_1, \ldots, r_i', \ldots, r_n)$, where $r_i'$ is new value of variable. Effect function that execute assignment of schema $A$ is simple replacement old variable with new polynomial.

**Intersection Problem.** Given the algebraic basis of assertions sets $I$ and $J$. Construct the algebraic basis assertions set $I \cap J$. Accordingly to Theorem 1 intersection construction can be held using formula (1).

**Stabilization Problem.** Show that the construction process of basis assertions sets associated to the program states will finish in finite time. Investigation of this problem is out of scope of this paper.

## Realization

Consider the $U$-$Y$ schema from Fig. 1. On this program, we will show work of algorithm considering data algebra as polynomial ring [17]. Applying the UAM to the $U$ - $Y$ schema we will receive next chain of computing and invariant relations.

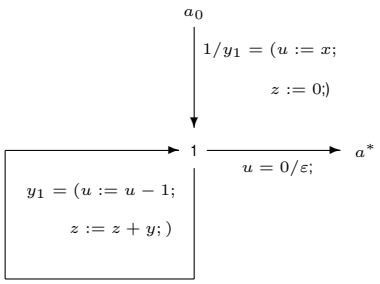



Fig. 1. $U$-$Y$ schema MULT($x, y$)

We generate the initial sets of assertions during first stage of the algorithm. At first, assertions bases for all the states of the program are assigned to 1 ($\forall N : N \bigcap 1 = N$), with $ToVisit = \{a_0\}$.

We move to state $a_0$ of $ToVisit$ and modify $Visited = \{a_0\}$. There is only one edge from $a_0$:

$$(a_0, 1, y = (u := x; z := 0), a_1).$$

We assign new value to assertion basis $N_{a_1}$ analyzing this edge:

$$N_{a_1} := ef(N_{a_0}, y) = ef(1, (u := x, z := 0)) = \{u - x = 0, z = 0\}.$$

In this case, $ToVisit = \{a_1\}$. We move to state $a_1$ of $ToVisit$ and modify $Visited = \{a_0, a_1\}$. State $a_1$ has two outgoing edges $(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1), (a_1, (u = 0), \varepsilon, a_*)$. We consider only the edge $a_1$ to $a_*$ because $a_1 \in Visited$.

After analysis we decide replace the edge $(a_1, (u = 0), \varepsilon, a_*)$ with $(a_1, 1, u := 0, a_*)$ That is required for every condition of equality type and should be replaced with assignment. We assign new value to assertion basis $N_{a_*}$ analyzing this edge:

$$N_{a_*} := ef(N_{a_1}, y) = ef(\{u - x = 0, z = 0\}, \{u := 0\}) = \{x = 0, z = 0\}.$$

In this case, $ToVisit = \{a_*\}$.

We move to state $a_*$ of $ToVisit$ and modify $Visited = \{a_0, a_1, a_*\}$. There are no edges from $a_*$. First stage of the UAM is over because $ToVisit = \emptyset$. We proceed to the second stage of the algorithm.

On second stage basis of assertions is generated on each step of iteration. After we intersect basis of obtained assertions with received on previous step. Start with initializing $ToVisit = A/\{a_0\} = \{a_1, a_*\}$.

We continue with state $a_1$ of $ToVisit$ . Initial value of assertion basis is

$$N = N_{a_1} = \{u - x = 0, z = 0\}.$$

State $a_1$ has two edges $(a_0, 1, (u := x; z := 0), a_1), (a_1, (u \neq 0), (u := u - 1; z := z + y), a_1)$.

We assign new value to assertion basis $N$ analyzing $(a_0, 1, (u := x; z := 0), a_1)$:

$$N := N \bigcap ef(N_{a_0}, y) := N \bigcap ef(1, (u := x; z := 0)) =$$
$$= \{u - x = 0, z = 0\} \bigcap \{u - x = 0, z = 0\} = \{u - x = 0, z = 0\}.$$

After analysis of $(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1)$ new value to assertion basis $N$ is assigned:

$$N := N \bigcap ef(N_{a_1}, y) := N \bigcap ef(\{u - x = 0, z = 0\}, (u := u - 1; z := z + y)) =$$
$$= \{u - x = 0, z = 0\} \bigcap \{u + 1 - x = 0, z - y = 0\} = \{z + (u - x)y = 0, (u - x)^2 + (u - x) = 0\}.$$

Based on $N \neq N_{a_1}$ we receive new assertion basis for $a_1$

$$N_{a_1} := \{z + (u - x)y = 0, (u - x)^2 + (u - x) = 0\}$$

and

$$ToVisit := ToVisit \bigcup \{a_1, a_*\} = \{a_1, a_*\}.$$

We move to state $a_*$ of $ToVisit$ and modify $Visited = \{a_1\}$. Let assign $N = N_{a_*} = \{x = 0, z = 0\}$. State $a_*$ has only one edge $(a_1, 1, u := 0, a_*)$.

We assign new value to assertion basis $N$ analyzing this edge:

$$N := N \bigcap ef(N_{a_1}, y) = = N \bigcap ef(\{z + (u - x)y = 0, (u - x)^2 + (u - x) = 0\}, u := 0) =$$
$$= N \bigcap \{z - xy = 0, x^2 - x = 0\} = \{x = 0, z = 0\} \bigcap \{z - xy = 0, x^2 - x = 0\} =$$
$$= \{z - xy = 0, x^2 - x = 0\}.$$

Based on $N \neq N_{a_*}$ we receive new assertion basis for $a_*$:

$$N_{a_*} := \{x = 0, z = 0\}.$$

We move to state $a_1$ of $ToVisit$ and modify $Visited = \emptyset$. Let assign

$$N := N_{a_1} = \{z + (u - x)y = 0, (u - x)^2 + (u - x) = 0\}.$$

State $a_1$ has two edges $(a_0, 1, (u := x; z := 0), a_1)$, $(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1)$.

We assign new value to assertion basis $N$ analyzing $(a_0, 1, (u := x; z := 0), a_1)$:

$$N := N \bigcap ef(N_{a_0}, y) := N \bigcap \{u - x = 0, z = 0\} =$$
$$= \{z + (u - x)y = 0, (u - x)^2 + (u - x) = 0\} \bigcap \{u - x = 0, z = 0\} =$$
$$= \{z + (u - x)y = 0, (u - x)^2 + (u - x) = 0\}.$$

After analysis of $(a_1, (u \neq 0), (u := u - 1; z := z + y), a_1)$ new value to assertion basis $N$ is assigned:

$$N := N \bigcap ef(N_{a_1}, y) := N \bigcap ef(N_{a_1}, (u := u - 1; z := z + y)) = \{xy - uy - z = 0, \ldots\}.$$

Based on $N \neq N_{a_1}$ we receive new assertion basis for $a_1$

$$N_{a_1} := \{xy - uy - z = 0, \ldots\}$$

and

$$ToVisit := ToVisit \bigcup \{a_1, a_*\} = \{a_1, a_*\}.$$

We continue with state $a_*$ of $ToVisit$ and modify $Visited = \{a_1\}$. Initial value of assertion basis received from previous step is

$$N = N_{a_*} = \{z - xy = 0, x^2 - x = 0\}.$$

State $a_*$ has only one edge $(a_1, 1, u := 0, a_*)$.

We assign new value to assertion basis $N$ analyzing this edge:

$$N := N \bigcap ef(N_{a_1}, u := 0) := N \bigcap \{xy - z = 0, \ldots\} = \{xy - z = 0, \ldots\}.$$

Based on $N \neq N_{a_*}$ we receive new assertion basis for $a_*$: $N_{a_*} := \{xy - z = 0, \ldots\}$.

Moving forward algorithm's steps we noticed that dimensions of bases $N_{a_1}$ and $N_{a_*}$ doesn't change, as well as one of the polynomials of these bases. This polynomials are $xy - uy - z = 0$ in $a_1$ and $xy - z = 0$ in $a_*$. As we notice from formulation of the problem invariant $z = xy$ validates programs. Proof of program correctness is obvious even without help of an automatic prover.

## Conclusion

In this paper we present theoretical basis and realization for application of UAM on program over polynomial ring as data algebra. We provide important definitions for program analysis and further methods. Part of polynomial ring theory were referred. Ideal interpretation for program invariants was chosen. We present reasoning for iterative approximation methods. Operations defined on Gröbner basis satisfy all requirements stated in [11] to apply UAM.

Method has been implemented using Maple software that contains powerful tools for symbolic operations. We showed method work on example and provided step by step listing of operations.

Development of iterative methods for invariant generation are facing with considerable challenge. However, development of programing languages, object oriented programming, lambda functions solving invariant generation problems for these languages requires the use of rather complex methods of modern general algebra.

## Bibliography

[1] Kaufmann M., Manolios P., Moore J.S. Computer Aidede Reasoning: An Approach. Kluwer Academic Publishers. - 2000. - 212 p.

[2] Nipkow T., Paulson L., Wenzel M. Isabelle/HOL: A Proof Assistant for Higher Order Logic. Springer Verlag. - LNCS. - 2002. - v. 2283. - PP. 3 – 51.

[3] Oppen D.C., Cook S.A. Proving Assertion About Programs that Manipulate Data Structures. In Proceed. of the 7-th Annual ACM Symposium on Theory of Computing (STOC 1975). - Aluquerque. - NM. - ACM Press. - 1975. - PP. 107 – 116.

[4] Clarke E.M., Emerson E.A. Design and Synthesis of Synchronization Skeleton Using Branching Time Temporal Logic. In D.C. Kozen, edit. Logic of Program Workshop. - Springer Verlag. - LNCS. - 1981. - v. 131. - PP. 52 – 71.

[5] Queille J.P., Sifakis J. Proving Specification and Verification of Concurent Systems in CESAR. In Proceed. of the 5-th Intern. Symposium on Programming. - Springer Verlag. - LNCS. - 1982. - v. 137. - PP. 373 – 351.

[6] Jones R. B. Simbolic Simulation Method for Industrial Formal Verification. Kluwer Academic Publishers. - 2002. - 286 p.

[7] Chou C. The Mathematical Foundation of Symbolic Trajectory Evaluation. In N. Halbwach and D. Peled, edit. Proc. of the 11-th International Conference on Computer Aided Verification (CAV 1999). - Springer Verlag. - LNCS. - v. 1633. - 1999.

[8] Hoare C.A.R. An axiomatic basis of computer programming. CACM. -1969. - v.12. - PP. 576–580.

[9] C.E.Shannon. The Mathematical theory of communication. In: The Mathematical Theory of Communication. Ed. C.E.Shannon and W.Weaver. University of Illinois Press, Urbana, 1949.

[10] Hoare T. The Verifying Compiler: A Grand Challenge for Computing Research. Journal of the ACM, No. 50(1), P. 63–69, 2003

[11] Godlevskii A. B., Kapitonova Y. V., Krivoi S. L., Letichevskii A. A. Iterative Methods of Program Analysis. Cybernetics and Systems Analysis Vol. 25, No. 2, 1989, 139–152.

[12] Krivoi S. L. About one invariant search algorithm in programs. - Cybernetics and Systems Analysis. - 1981. - No. 5, p. 12–18.

[13] Krivoi S. L. About invariant relations search in programs. Mathematical theory and design of computing machines -Kiev: UK AN USSR. -1978. - p. 35–51.

[14] Letichevsky A. A. On finding invariant relations of programs. In Algorithms in Modern Mathematics and Computer Science (Urgench, 1979), number 122 in LNCS, pages 304-314, 1981.

[15] Buchberger B., Winkler F. Gröbner Bases and Applications, Cambridge University Press, 1998

[16] Maksymets O. M. Check of Invariants generated by Iterative Algorithm for programs on Absolutely Free Algebra using Mathematical Induction, Problems of Programming 2012, Vol. 2-3, 228-333

[17] Maksymets O.M. Upper approximation method for polynomial invariants Theoretical and Applied Aspects of Cybernetics. Proceedings of the 2nd International Scientific Conference of Students and Young Scientists. Computer Science Section. - Kyiv. - 2012. - P. 45–47.

[18] Lvov M. S. About one algorithm of program polynomial invariants generation Proc. Workshop on Invariant Generation: (Techn. rep.) Univ. of Linz; Eds. M. Giese, T. Jebelean. N 0707 (RISC Report Series). Linz (Austria), 2007. P. 85–99 (electronic).

[19] Lvov M., Kreknin V. Nonlinear invariants for linear loops and eugen polynomials. - Programming technical complexes. - 2012.

[20] Godlevskii A. B., Kapitonova Y. V., Krivoi S. L., Letichevskii A. A. Iterative Methods of Program Analysis. Equalities and Inequalities. -Cybernetics and Systems Analysis. - 1990. - No. 3, p. 1–10.

[21] Sabelfeld V. K. Equivalente Transformationen fur Flussdiagramme. -Acta Informatica. - v. 10. -1978. - p. 127–155.

[22] Markus Müller-Olm, Michael Petter, and Helmut Seidl Interprocedurally Analyzing Polynomial Identities. - Symposium on Theoretical Aspects of Computer Science), LNCS 3884, Springer, 2006. - p. 50–67.

[23] Enric Rodriguez-Carbonell, Deepak Kapur An Abstract Interpretation Approach for Automatic Generation of Polynomial Invariants Proc. Static Analysis Symposium (SAS), Italy, August 2004.

## Authors' Information

***Sergii Kryvyi*** *Professor, Cybernetics department, Taras Shevchenko National University of Kiyv, Ukraine.*
*e-mail: s.l.krivoi@gmail.com*
*Major Fields of Scientific Research: Verification Theory, Petri Nets, Software Development.*
***Oleksandr Maksymets*** *Ph.D. student, Cybernetics department, Taras Shevchenko National University of Kiyv, Ukraine.*
*e-mail: maksymets@gmail.com*
*Major Fields of Scientific Research: Verification Theory, Data Flow Analysis.*