A Method of Verification of Functional Programs Based on Graph Models

Andrew M. Mironov

Moscow State University, Russia

Abstract. In the paper we introduce a concept of a graph model of a functional program. We show how to use this model for verification of functional programs.

Keywords: functional programs, graph model, verification.

1 Introduction

1.1 The problem of verification of programs

The problem of verification of programs has the following form:

- given a program P and its formal specification S,
- prove that $P \models S$ (i.e. P meets S).

In this paper we consider the following special case of this problem:

- P is a functional program (FP)
- S is another FP, and
- $P \models S$ means that functions computed by P and S are equal.

For example, S is a description of a required function, and P is an implementation of this function.

1.2 Comparison with related works

The problem of verification of FPs is considered in many works (see, for example, [4]–[9]). The main methods of verification of FPs are computational induction and structural induction (a bried description of these methods can be found in [4], ch. 5). Verification of FPs with use of these methods is based on construction of logical assertions related to verified FPs, which are called *admissible predicates*. Because there is no a general algorithm of construction of such predicates, then the most realistic way of verification of FPs based on these methods is an elaboration of interactive procedures of synthesis of corresponding admissible predicates.

The main advantage of verification method of FPs proposed in the present paper in comparison with other works is the following. If verified FPs are represented by graph models, then the problem of construction of a corresponding

admissible predicate can be decomposed to the problem of construction of logical assertions related to some pairs of nodes of these graph models, and a complexity of these assertions can be essentially less than a complexity of the admissible predicate.

2 A notion of a functional program

In this paper we consider a simplified notion of a FP. Without loss of generality, we assume that FPs under consideration does not contain nested recursions.

For precise definition of a notion of a FP we need in auxiliary notions described below.

2.1 Variables, constants, functional symbols

We assume that there are given

- a set $\mathcal{D}$ of **values**, which contains all natural numbers
- a set Con of **constants**, and every constant $c \in Con$ is associated with a value which is denoted by the same symbol c
- a set Var of **variables**, and every variable $x \in Var$ is associated with a set $\mathcal{D}_x \subseteq \mathcal{D}$ of values, which can be substituted instead of this variable
- a set Fun of **function symbols (FSs)**, and every $f \in Fun$ is associated with an integer number $ar(f) \geq 1$, which is called an **arity** of the FS f.

Some of FSs are called **primary** FSs. Every primary FS is associated with a partial function, which is denoted by the same symbol f, and has the form

$$f : \mathcal{D}^{ar(f)} \to \mathcal{D}$$

We assume that the set `Fun` contains the following FSs.

- FS `if_then_else` of the arity 3.
  A corresponding function maps every triple
  $$(d_1, d_2, d_3) \in \mathcal{D}^3$$
  - to d_2, if $d_1 = 1$, and
  - to d_3, otherwise.

  For every triple $(d_1, d_2, d_3) \in \mathcal{D}^3$ the expression
  $$\texttt{if\_then\_else}\ (d_1, d_2, d_3)$$
  will be denoted briefly as
  $$d_1\ ?\ d_2 : d_3$$
- FS `eq` of arity 2. A corresponding function maps every pair (d_1, d_2)
  - to 1, if $d_1 = d_2$, and
  - to 0, otherwise.

  For every pair $(d_1, d_2) \in \mathcal{D}^2$ the expression `eq` (d_1, d_2) will be denoted briefly as $d_1 = d_2$.

Below the symbol Fun^+ denotes a set of all FSs which differ from `if_then_else`.

2.2 Expressions, substitutions, evaluations

Expressions are constructed from variables, constants and FSs. The set of all expressions is defined as follows.

1. Every variable and every constant is an expression.
2. For
   - every list of expressions $e_1, \ldots, e_n$, and
   - every FS f of arity n

   the string
   $$f(e_1, \ldots, e_n)$$
   is an expression.

A **substitution** is a string θ of the form

$$\theta \;=\; [x_1 := e_1, \ldots, x_k := e_k] \tag{1}$$

where

- $x_1, \ldots, x_k$ is a list of distinct variables, and
- $e_1, \ldots, e_k$ are expressions.

For every substitution θ of the form (1), and every expression e the string θe denotes the result of replacement in e every variable x_i with corresponding expression e_i.

An **evaluation** is a partial function of the form

$$\xi : Var \to \mathcal{D}$$

For every pair ξ_1, ξ_2 of evaluations, and every variable $x \in Var$, the string

$$\xi_1(x) = \xi_2(x)$$

denotes the fact that values $\xi_1(x)$ and $\xi_2(x)$ either are both undefined, or are both defined and equal.

For every expression e a **value** $\xi(e)$ of e on evaluation ξ is defined iff one of the following conditions holds.

1. $e = x \in Var$, and $\xi(x)$ is defined. In this case $\xi(e) \stackrel{\text{def}}{=} \xi(x)$.
2. $e = c \in Con$. In this case $\xi(e) \stackrel{\text{def}}{=} c$.
3. e has the form $e_1 \;?\; e_2 \;:\; e_3$, and $\xi(e_1)$ is defined. In this case
   - if $\xi(e_1) = 1$, then $\xi(e) \stackrel{\text{def}}{=} \xi(e_2)$, i.e.
     - either $\xi(e)$ and $\xi(e_2)$ are both undefined ,
     - or $\xi(e)$ and $\xi(e_2)$ are both defined and equal,
   - if $\xi(e_1) \neq 1$, then $\xi(e) \stackrel{\text{def}}{=} \xi(e_3)$
4. e has the form $f(e_1, \ldots, e_k)$, and
   - $f \in Fun^+$
   - FS f is associated with a partial function
   - values $\xi(e_1), \ldots, \xi(e_k)$ are defined,
   - a value of function f on the tuple $(\xi(e_1), \ldots, \xi(e_k))$ is defined.

   In this case
   $$\xi(e) \stackrel{\text{def}}{=} f(\xi(e_1), \ldots, \xi(e_k))$$

2.3 Functional programs

A **functional program (FP)** is a system of equations of the form

$$\begin{cases} f_1(x_{11}, \ldots, x_{1k_1}) = e_1 \\ \ldots \\ f_n(x_{n1}, \ldots, x_{nk_n}) = e_n \end{cases} \tag{2}$$

where

- $f_1, \ldots, f_n$ are distinct FSs, which are names of **functions, defined by this FP**,
  (all the FSs $f_1, \ldots, f_n$ are not primary FSs)
- $x_{11}, \ldots, x_{nk_n}$ are distinct variables, which are formal parameters of the defined functions, and
- $e_1, \ldots, e_n$ are expressions with the following properties: for every $i = 1, \ldots, n$
  - the set of all variables occurring in e_i, is equal to the set

$$\{x_{i1}, \ldots, x_{ik_i}\}$$

  - every FS occurred in e_i,
    - either is a primary FS,
    - or belongs to the set $\{f_1, \ldots, f_n\}$.
  - every subexpression of e_i of the form $f_j(\ldots)$ contains only one non-primary FS (i.e. a FP does not contain nested recursions).

2.4 Functions defined by FPs

Functions defined by FPs are computed by a standard recursion: if a function f_i is defined by system (2), then its value on the tuple $(d_1, \ldots, d_{k_i})$ is equal to a value of expression

$$[x_{i1} := d_1, \ldots, x_{ik_i} := d_{k_i}]e_i \tag{3}$$

A value of expression (3) is computed as follows.

- If (3) is a constant, then its value is equal to this constant.
- If (3) has the form

$$u_1 \; ? \; u_2 \; : \; u_3 \tag{4}$$

  then at first a value of u_1 is computed, and
  - if a value of u_1 is undefined, then a value of (4) is undefined,
  - if a value of u_1 is equal to 1, then a value of u_2 is computed, and
    - if a value of u_2 is undefined, then a value of (4) also is undefined,
    - otherwise a value of u_2 is equal by definition to a value of (4)
  - if a value of u_1 is not equal to 1, then a value of u_3 is computed, and
    - if a value of u_3 is undefined, then a value of (4) is undefined,
    - otherwise a value of u_3 is equal by definition to a value of (4)

– If (3) has the form

$$f(u_1, \ldots, u_m) \tag{5}$$

where f is a primary FS, which is not equal to `if_then_else`, then values of $u_1, \ldots, u_m$ are computed, and after this a value of function f on the tuple of values of $u_1, \ldots, u_m$ is computed. By definition, this value is equal to a value of (5).
If
- either one of values of $u_1, \ldots, u_m$ is undefined,
- or a value of f on a tuple of values of u_1, …, u_m is undefined

then a value of (5) is undefined.

– If (3) has the form

$$f_j(u_1, \ldots, u_{k_j})$$

where f_j is a FS, which is a name of a function defined by system (2), then its value is equal to a value of expression

$$[x_{j1} := u_1, \ldots, x_{jk_j} := u_{k_j}]e_j \tag{6}$$

which is computed by the same way as a value of (3) is computed.

If the above process is not terminated, then a value of f_i on the tuple $(d_1, \ldots, d_{k_i})$ is undefined.

3 Graph models of functional programs

3.1 Graph models of FPs

For every FP (2) we can construct a graph which is called **a graph model (GM)** of FP (2), in which

– every node has a label which is equal to some subexpression of one of the expressions $e_1, \ldots, e_n$, and
– every edge has a label of one of the following types:
  1. **a condition**: a label of this type has the form

$$\varphi\,? \quad \text{or} \quad \neg\varphi\,?$$

where φ
- either is some subexpression of one of expressions from (2),
- or is constant 1.

  2. **a substitution**: a label of this type has the form

$$[x_1 := u_1, \ldots, x_m := u_m]$$

where
- $x_1, \ldots, x_m$ are distinct variables which have occurrences in (2), and
- $u_1, \ldots, u_m$ are some subexpressions of one of expressions from (2).

3. **a link to a subexpression**: a label of this type has the form

$$(i)$$

and has the following meaning: for every edge with a label (i)
- label of a start node of this edge has the form $f(u_1, \ldots, u_m)$, and
- label of an end node of this edge is u_i.

GM of FP (2) is defined as follows:

- A set of its nodes is equal to the set of subexpressions of expressions from (2). A label of every node is equal to the corresponding subexpression.
- For every equation $f_i(x_{i1}, \ldots, x_{ik_i}) = e_i$ from (2), GM has an edge with label 1? from a node with the label $f_i(x_{i1}, \ldots, x_{ik_i})$ to a node with the label e_i.
- For every node N with a label of the form

$$e_1 \; ? \; e_2 \; : \; e_3$$

GM contains
- an edge from N to a node with the label e_2, and a label of this edge is $e_1?$, and
- an edge from N to a node with the label e_3, and a label of this edge is $\neg e_1?$.

- For every node N with a label of the form

$$f_i(u_1, \ldots, u_{k_i})$$

where f_i is a name of some function which is defined by FP (2), GM contains an edge from N to a node with the label $f_i(x_{i1}, \ldots, x_{ik_i})$, and label of this edge has the form

$$[x_{i1} := u_1, \quad \ldots, \quad x_{ik_i} := u_{k_i}]$$

- For
  - every node N with a label of the form

$$f(u_1, \ldots, u_m) \tag{7}$$

where $f \in Fun^+$, and
  - every $i = 1, \ldots, m$

GM contains an edge from N to a node with the label u_i, and a label of this edge is (i).

3.2 Simplification of GMs

GMs of FPs can be transformed to equivalent (in certain sense) GMs. The transformations consists of a removing of edges and nodes, as following. Let a GM has a node N, such that a label of every edge incoming in N and every edge outgoing from N has a type "a condition". In this case

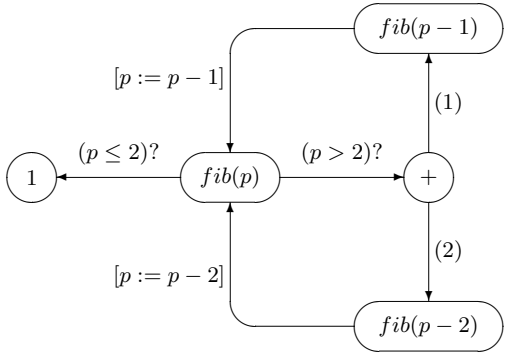

Fig. 1: A simplified GM of FP fib.

– node N can be removed, and
– every pair of edges of the form

$$N_1 \xrightarrow{\varphi?} N, \quad N \xrightarrow{\varphi_2?} N_2$$

can be changed on one edge of the form

$$N_1 \xrightarrow{(\varphi_1 \wedge \varphi_2)?} N_2$$

This operation can be made several times.

Also it is possible

– to change labels of nodes of the form (7), where f is a primary FS, on labels of the form f
(i.e. not write a list of arguments after symbol f)
– to remove every edge with a label of the form (i), and to remove a node which is an end of this edge.

A GM, which is a result of such transformations, is called **a simplification** of an original GM.

3.3 Examples of GMs of FPs

Consider a FP, which defines a Fibonacci function

$$fib(p) = (p \leq 2)?1 : fib(p-1) + fib(p-2)$$

where the set of values, associated with variable p, is the set of natural numbers $(1, 2, \ldots)$.

A simplified GM, which corresponds to this FP, has the form depicted in the figure 1.

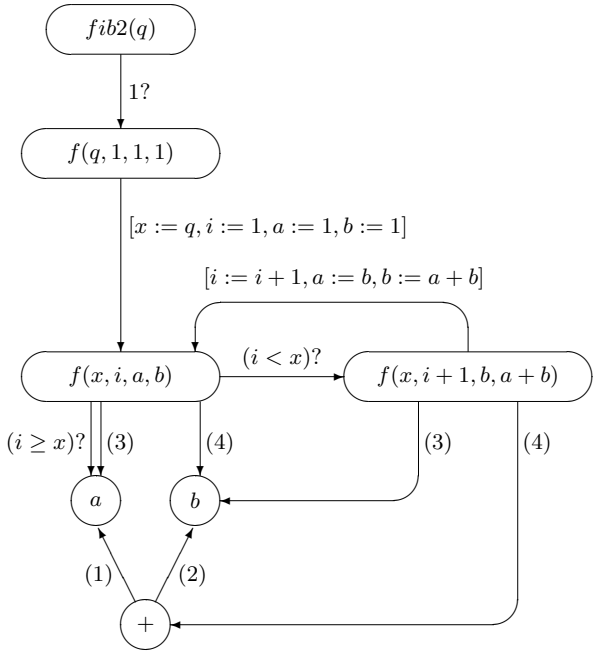

Fig. 2: A simplified GM of a FP, which defines function fib2.

Consider other FP, which also defines a Fibonacci function, but computes it by the method "from bottom to top" (the previous FP computes Fibonacci function by the method "from top to bottom"):

$$fib2(q) = f(q, 1, 1, 1)$$
$$f(x, i, a, b) = (i < x)?f(x, i + 1, b, a + b) : a$$

where all variables range over the set of integer numbers.

A simplified GM, which corresponds to this FP, has the form, which is depicted in figure 2.

4 A method of proving of equality of functions defined by FPs

Assume that two FP are given, and

- first FP defines functions $f_1, \ldots, f_n$, and
- second FP defines functions $g_1, \ldots, g_m$.

If we would like to prove that functions f_1 and g_1, which are defined by these FPs, are equal (under a condition that arguments of these functions satisfy certain relation which is called a **precondition**), then it is possible to prove as follows.

At first we rename variables in these FPs, such that every variable has an occurrence in only one equation in these FPs. Second, we construct GMs of these FPs.

Theorem.

Let there is a set of arcs, which connect nodes of first GM with nodes of second GM, and satisfy the following conditions.

1. *Every arc has a label, which is a boolean-valued expression.*
2. *There is an arc,*
   - *which connects a node in first GM with a label of the form* $f_1(x_{11}, \ldots, x_{1n_1})$, *with a node in second GM with a label of the form* $g_1(y_{11}, \ldots, y_{1m_1})$, *and*
   - *a label of which is equal to a precondition.*
3. *If*
   - *one GM has an edge with label* θ *from a node* v_1 *to a node* v_1',
   - v_2 *is a node of other GM*
   - *node* v_1 *is connected with a node* v_2 *by an arc with a label* α,

   then node v_1' *is connected with a node* v_2 *by an arc with a label* α', *such that the following implication holds:*

$$\alpha \to \theta\alpha'$$

4. *If*
   - *one GM has an edge with a label* φ? *from a node* v_1 *to a node* v_1',
   - v_2 *is a node of other GM*
   - *node* v_1 *is connected with node* v_2 *by an arc with label* α,

   then node v_1' *is connected with node* v_2 *by an arc with label* α', *such that the following implication holds:*

$$(\alpha \wedge \varphi) \to \alpha'$$

5. *If*
   - *node* v_1 *of one GM is connected with a node* v_2 *of other GM by an arc with a label* α,
   - v_1 *and* v_2 *have no outgoing edges*
   - *labels of* v_1 *and* v_2 *are expressions* e_1 *and* e_2 *respectively*

   then the following implication holds:

$$\alpha \to (e_1 = e_2) \tag{8}$$

6. *If*
   - *node* v_1 *of one GM is connected with a node* v_2 *of other GM by an arc with a label* α,
   - *label* v_1 *is a primary FS*

then in this case
- *label* v_2 *is the same FS,*
- *let*
  - *a set of ends of edges outgoing from* v_1 *has the form*

$$\{v_{11}, \ldots, v_{1n}\}$$

and for every $i = 1, \ldots, n$ *node* v_1 *is connected with node* v_{1i} *by an edge with a label* (i)
  - *a set of ends of edges outgoing from* v_2 *has the form*

$$\{v_{21}, \ldots, v_{2n}\}$$

and for every $i = 1, \ldots, n$ *node* v_2 *is connected with node* v_{2i} *by an edge with a label* (i),

then for every $i = 1, \ldots, n$ *node* v_{1i} *is connected with a node* v_{2i} *by an arc with a label* α_i*, such that the following implication holds:*

$$\alpha \to (\alpha_1 \wedge \ldots \wedge \alpha_n)$$

7. *If*
   - *one GM has*
     - *an edge with a label* θ *from a node* v_1 *to a node* v'_1,
     - *there are edges outgoing from* v_1 *such that*
       - *labels of these edges are* $(1), \ldots, (n)$*, and*
       - *ends of these edges are nodes* $v_{11}, \ldots, v_{1n}$ *respectively,*
     - *there are edges outgoing from a node* v'_1 *such that*
       - *labels of these nodes are* $(1), \ldots, (n)$*, and*
       - *ends of these edges are* $v'_{11}, \ldots, v'_{1n}$ *respectively*
   - v_2 *is another node of second GM, such that for every* $i = 1, \ldots, n$ *there is an arc which connects* v_2 *with* v_{1i}*, and a label of this arc is* α_i

   then for every $i = 1, \ldots, n$ *node* v_2 *is connected with a node* v'_{1i} *by an arc with a label* α'_i*, such that the following implication is hold:*

$$\alpha_i \to \theta\alpha'_i$$

Then, if both FPs terminate their computation on every tuple of values of arguments which satisfy a precondition, then values of functions f_1 *and* g_1 *on these tuples of values of arguments are equal.*

Proof.

If an arc with label α connects nodes with labels e_1 and e_2 respectively, then we shall interprete this arc as a proposition that the following implication holds:

$$\alpha \to (e_1 = e_2) \tag{9}$$

According to this interpretation, an arc which connects

– a node of first GM with a label of the form

$$f_1(x_{11}, \ldots, x_{1n_1})$$

and
– a node of second GM with a label of the form

$$g_1(y_{11}, \ldots, y_{1m_1})$$

(such arc exists by assumption 2 of a condition of the theorem)

represents a goal proposition.

For proving that goal proposition is true for every evaluation ξ_0, we

– assume that a condition of this implication holds on this evaluation, and
– prove that a conclusion of this implication also holds on this evaluation.

In this case truth of a condition of implication (9) means that evaluation ξ_0 is a list of values of arguments of the FP, which satisfies a precondition. By assumption, functioning of every FP on this list terminates. This functioning can be represented as a finite tree,

– nodes of which correspond to nodes of GM (several nodes of the tree can correspond to one node of the GM), and are labeled by expressions of the form ξe, where
  - e is an expression, which is a label of a corresponding node GM, and
  - ξ is a substitution, which associates every variable from e with a value.
– edges of which correspond to edges of the GM, and have the same labels as corresponded edges of the GM.

Roots of these trees correspond to nodes of the GMs with labels $f_1(x_{11}, \ldots, x_{1n_1})$ and $g_1(y_{11}, \ldots, y_{1m_1})$.

If

– node w_1 of first tree corresponds to node v_1 of first GM,
– node w_2 of second tree corresponds to node v_2 of second GM, and
– among arcs mentioned in the theorem, there is an arc which connects v_1 and v_2 with a label α,

then we draw an arc with the same label α, which connects w_1 and w_2.

The condition of the theorem and a definition of functioning of a FP imply the following propositions.

– If
  - there is an arc with a label α, which connects a node of first tree with a node of other tree, and
  - labels of these nodes have the form $\xi_1 e_1$ and $\xi_2 e_2$ respectively

then expression α holds on evaluation

$$\xi_1 \cup \xi_2$$

which is equal to ξ_1 on variables occurring in e_1, and to ξ_2 on variables occurring in e_2 (this can be proved by induction, where a base of induction is an assumption that the precondition holds on evaluation ξ_0, and inductive steps can be founded by the corresponding assumptions in the theorem).

– Every terminal node of one of the trees is connected by an arc with a terminal node of other tree.

Because every arc, which connects terminal nodes, satisfies condition (8), in which

– α is a label of this arc, and
– e_1 and e_2 are labels of nodes of a GM, which corresponds to these terminal nodes of the trees

then (8) implies the implication

$$(\xi_1 \cup \xi_2)\alpha \rightarrow (\xi_1 e_1 = \xi_2 e_2)$$

where $\xi_1 e_1$ and $\xi_2 e_2$ are labels of the above terminal nodes. As it was stated above, expression α holds on the evaluation $\xi_1 \cup \xi_2$. Consecuently, values of the expressions $\xi_1 e_1$ and $\xi_2 e_2$ (which are labels of terminal nodes) are equal.

Moving from terminal nodes to root nodes, we can use the above propostions for proving that for every pair of nodes which are connected by an arc, values of expressions, which are labels of these nodes, are equal.

Because root nodes of these trees also are connected by an arc, then values of expressions, which are labels of these nodes, are equal. ■

The proposed method can be modified in such a manner that for proving a equality of functions defined by different FPs it is necessary to define not all arcs mentioned in the theorem, but only several of them (omitted arcs can be generated automatically). We shall not describe in detail the modified method. We only note that for proving of equality of functions which are defined by FPs in section 3.3, it is enough to define only the following arcs:

1. an edge with a label $p = q$, which connects
   – a node with a label $fib(p)$ of first GM, and
   – a node with a label $fib2(q)$ of second GM
2. an edge with a label $(p = x) \wedge (i \leq x)$, which connects
   – a node with a label $fib(p)$ of first GM, and
   – a node with a label $f(x, i, a, b)$ of second GM
3. an edge with a label $p = i$, which connects
   – a node with a label $fib(p)$ of first GM, and
   – a node with a label a of second GM
4. an edge with a label $p = i + 1$, which connects
   – a node with a label $fib(p)$ of first GM, and
   – a node with a label b of second GM.

References

1. K.G. van den Berg, P.M. van den Broek: Static analysis of functional programs. Information and Software Technology, 1995, Volume 37, Number 4, Pages 213-224.
2. A.Ireland, A.Bundy: Automatic verification of functions with accumulating parameters. Journal of Functional Programming, 9(2):225–245, March 1999.
3. J. Loeckx and K. Sieber: The Foundations of Program Verification. Teubner, second edition, 1987.
4. Z. Manna: Mathematical Theory of Computation. McGraw-Hill Inc., 1974.
5. Manna, Z., Ness, S., and Vuillemin, J.: Inductive methods for proving properties of programs. Commun. ACM 16, 8 (Aug. 1973), 491–502.
6. Z. Manna and J. McCarthy: Properties of Programs and Partial Function Logic. Machine Intelligence, 5:27–37.
7. Z. Manna and A. Pnueli: Formalization of Properties of Functional Programs. J. ACM, 17(3):555–569, 1970.
8. N. Popov and T. Jebelean: Using Computer Algebra Techniques for the Specification, Verification and Synthesis of Recursive Programs. Mathematics and Computers in Simulation, pages 1–13, 2007.
9. N. Popov and T. Jebelean: A Prototype Environment for Verification of Recursive Programs. In Z. Istenes, editor, Proceedings of FORMED08, pages 121–130, March 2008.