Bull. Nov. Comp. Center, Comp. Science, 3 (1995), 59–68
© 1995 NCC Publisher

On completeness of mechanism of annotation-directives*

V.N. Kasyanov

An annotated program is a program written down in a programming language extended by the annotations which are formalized comments in the basic programs and relevant for the semantics of the program can be described. The paper focuses on the completeness property of the directive mechanism for different kinds of basic program manipulations. It is shown that any basic program transformation represented by a normal Markov algorithm may be modelled within annotated program framework so that annotations specify only elementary transformations of annotated programs and for any basic program the transformation process not only has the same result as the normal algorithm modelled, but also performs the similar sequence of processing steps.

1. Introduction

An annotated program is a program written down in a programming language extended by the annotations which are formalized comments in the basic programs and relevant for the semantics of the program to be annotated [1]. The extensions of high-level languages by special annotations (pragmas) are in common use in compilers and are currently considered as a part of language description [2–4]. One of three main approaches to transformational program development is a so-called extended compilation [5] characterized by permitting advice and partially relaxing the limits on the basic language. That is, the transformation system accepts not only the basic program, but also some annotations being guidance on how to do the transformation. But annotations are not just pragmas or hints for compiler or another transformation system (e.g., for automatic parallelization of a sequential program). They can be used to modify the semantics of the basic program, but only in a very moderate manner [1–7]. In [1, 6–8] an attempt was made to suggest methods and tools for annotated programming, whereby a given general-purpose program can be annotated by known information about a specific context of its applications and correctly transformed into a specialized program which is equivalent to the original one on the context-defined ranges of inputs and outputs and is better than

*Partially supported by the Russian Foundation for Basic Research under Grant 95–01–01334.

that by quality criteria given by the context. According to the approach presented, the following properties hold. Annotations added to a basic program specify a covering context. It means that any actual program application from the context described must be admissible by annotations, but some admissible applications may be beyond the context. Annotated programs are subjected to concretizing transformations as a whole. It means that the transformations can change not only the basic program but their annotations as well. Annotations may specify the context of basic program application both explicitly – as assertions which are predicate constraints on admissible properties of program fragments or admissible states of computations, and implicitly – as directives which specify admissible transformations of annotated programs or states of computations in indicated points of a program.

It was shown [6–8] that the class of correct transformations of annotated programs covers various kinds of basic program manipulations. It contains all equivalent transformations as well as a number of such nonequivalent transformations which specialize or generate a basic program to be transformed, in particular, partial evaluation (mixed computation) of basic programs on partially given inputs [9]. So, the approach permits specializing and generalizing transformations of basic programs to be reduced to equivalent transformations of annotated programs and equivalent transformation techniques developed earlier to be employed for their investigation. Another advantage of the approach outlined above is the possibility to perform global transformations of basic programs by iterative application of elementary (context-free) transformations of annotated programs.

It was shown in [7] that within annotated language framework, where annotations are names of elementary transformations of annotated programs, the transformational semantics of Algol-like programming language can be described. This paper investigates a problem of completeness of the directive mechanism for different kinds of basic program manipulations. It is proved that any basic program transformation represented by a normal Markov algorithm [10, 11] may be modelled within annotated program framework in such a way that annotations specify only elementary transformations of annotated programs and for any basic program the transformation process not only has the same result as the normal algorithm modelled, but also performs the similar sequence of processing steps.

The paper consists of three parts. The first one presents the notion of normal Markov's algorithm; the second one describes the subclass of annotated programs and gives some notions connected with the problems of realizing normal Markov's algorithms by annotated programs; the third one presents an algorithm of realizing normal algorithms by annotated programs of investigated type.

2. Normal Markov's algorithms

Fix an alphabet Σ, and let symbols $\rightarrow$ and $\bullet$ be not in Σ. After Markov, formulas of the type

$$\alpha \rightarrow \beta \text{ and } \alpha \rightarrow \bullet\beta,$$

are called *simple* and *final substitution formulas* respectively, here α, $\beta \in \Sigma^*$, i.e., α, β are some words in Σ (may be empty); α is called the *left* word of the formula, β is called the *right* word of the formula.

An arbitrary finite sequence of substitution formulas is called a *scheme.*

The following rule for the processing of words in Σ is called *normal algorithm* with a given scheme π in the alphabet Σ.

Given an arbitrary word $\gamma \in \Sigma^*$. It is assumed that a word $\gamma_0 = \gamma$ is derived from a word γ after zero steps of the processing. For some n, $n \geq 0$, let a word γ_n, be already known as derived from γ after n steps of the processing. Then $(n+1)$-th step of the processing is the following. Find the first substitution formula in the scheme π such that its left part α is a subword of the word γ_n; then substitute the right word β of the formula for the first appearance of α in γ_n (in particular, if α is an empty word Λ, then, by definition, it is assumed that the result of this substitution is a word $\beta\gamma_n$). The resulting word is denoted by γ_{n+1}. If γ_{n+1} is derived using the simple substitution formula, then γ_{n+1} is said to be a word derived from γ after $n+1$ steps of the processing. If the used substitution formula is final, then the word γ_{n+1} is called the result of the processing of the word γ by an algorithm π and is denoted by $\pi(\gamma)$.

If the scheme π does not contain a substitution formula such that its left word is a subword of γ_n, then $\gamma_{n+1} = \gamma_n$ and a word γ_{n+1} is called the result of the processing of the word γ by the algorithm π, i.e., $\pi(\gamma) = \gamma_{n+1}$.

In both latter cases the processing of the word γ is said to be terminated after $(n+1)$-th step. If there are no steps when the processing terminates, then the result of the processing of the word γ by the algorithm π is undetermined. In this case an undetermined value is assigned to $\pi(\gamma)$.

3. Annotated programs

Let there be an alphabet D, not intersecting with Σ, which elements are called *directives.* Let symbols $\{, \}, \Rightarrow$ be not in $\Sigma \cup D$.

Denote by $A(D)$ the set of all words of the form $\{d_1, \ldots, d_k\}$, where $k \geq 0$ and $d_1, \ldots, d_k$ are directives, distinct in pairs, from D. The elements of $A(D)$ are called *annotations.*

A pair (d, p) is called *context-free transformation* of annotated programs, where $d \in D$ is the name of the transformation, p is an arbitrary recursive set of pairs (φ, ψ) of words in the alphabet $\Sigma \cup A(D)$. In the pair (φ, ψ)

the word φ, is called a *replaceable* fragment, and ψ is called a *replacing* fragment. The set p may be specified by some "simple" rule, which allows us to know whether an arbitrary pair of words has the property p. In this paper we consider as p only finite sets of pairs of words; to specify p we use formulas of the type $\{\varphi_1 \Rightarrow \psi_1, \ldots, \varphi_m \Rightarrow \psi_m\}$, where $m = |p|$ and the pair (φ_i, ψ_i) belong to p for any i.

An arbitrary word α in the alphabet $\Sigma \cup A(D)$ is called an *annotated program.*

For any $\alpha \in (\Sigma \cup A(D))^*$ we denote by $BASE(\alpha)$ the so-called *base* of the annotated program, which is a word in the alphabet Σ, derived from α by eliminating all elements from $A(D)$, i.e.,

$$BASE(\alpha) = \begin{cases} \alpha, & \text{if } \alpha \in \Sigma^*, \\ BASE(\beta), & \text{if } \alpha = c\beta \quad \text{and } c \in A(D), \\ \alpha BASE(\beta), & \text{if } \alpha = c\beta \quad \text{and } c \in \Sigma. \end{cases}$$

The processing of the annotated program α is formulated in the following way. A fragment β of the program α is called *active*, if it is of the form $\gamma\omega$, where $\gamma \in A$, and γ contains a directive d, for which β is a replaceable fragment (note that with respect to [5] we narrow down the notion of an active fragment and thereby narrow down a class of processings of annotated programs). The transformation of an active fragment β consists of the replacement of it in α with the corresponding replacing fragment from the directive d. The processing of α is performed in steps, each being a transformation of a nonempty set of active fragments. The process terminates, if the program does not contain active fragments on some step. It is easy to see that for the same program the definition stated above admits a number of ways of processing varying in a set of active fragments, choosing for the transformation at some step.

Let $\alpha_0 = \alpha_1, \ldots, \alpha_n \ldots$ be the processing of the annotated program α. Then a *frame* of the processing is a sequence of words derived from the sequence $\beta_0, \beta_1, \ldots, \beta_n, \ldots$, where $\beta_i = BASE(\alpha_i)$ by eliminating all β_j, $j \geq 1$, such that $\beta_j = \beta_{j-1}$.

A program α is called *deterministic* if it determines a processing such that there are no two different active fragments on each step of the processing.

It is said that the deterministic annotated program α *realizes* the processing of the word γ by normal algorithm π, if there exists an integer $k > 0$, such that there is no more than k sequential steps retaining the base program in the processing of α, and the frame of the processing of α coincides with the sequence of words obtained in the processing of γ by the algorithm π.

A normal algorithm with the scheme π in the alphabet Σ is *realizable* by the annotated program, if there exist an alphabet D of directives and the function $f : \Sigma^* \to (\Sigma \cup A(D))^*$, such that for every $\gamma \in \Sigma^*$, $f(\gamma)$ is the deterministic annotated program realizing the processing of γ by the normal algorithm with the scheme π.

4. Realizability of the class of normal algorithms

Let a scheme $\pi = (\pi_1, \ldots, \pi_n)$ be specified in the alphabet Σ. For every i, $1 \leq i \leq n$, we denote by φ_i and ψ_i respectively the left and the right words of the substitution formula π_i of the scheme π.

We consider as D a set of $2n + 6$ symbols which are not in Σ. Let $D = \{\bigtriangledown, \bigtriangleup, \Box, \Diamond, \circ, q_0, q_1, \ldots, q_n, p_1, \ldots, p_n\}$. For every $\alpha \in \Sigma^*$ denote as $\overline{\alpha}$ a word in the alphabet $\Sigma \cup A(D)$, derived from α by the following rules:

$$\overline{\alpha} = \begin{cases} \Lambda, & \text{if } \alpha = \Lambda, \\ c\{\circ\}\overline{\beta}, & \text{if } \alpha = c\beta \quad \text{and} \quad c \in \Sigma. \end{cases}$$

The set of transformations associated with the directives from D, is determined by the rules:

1. A *new step* transformation is associated with the directive $\bigtriangledown$ and consists of the replacements:
   (a) $\{\bigtriangledown, q_n\} \Rightarrow \{\bigtriangledown, \Box\}$;
   (b) $\{\bigtriangledown, q_i\} \Rightarrow \{\bigtriangledown, p_{i+1}\}$ for every $i = 0, 1, \ldots, n-1$;
   (c) $\{\bigtriangledown\}\{\bigtriangleup, q_i\} \Rightarrow \{\bigtriangledown, q_i\}\{\bigtriangleup\}$ for every $i = 0, \ldots, n$;
   (d) $\{\bigtriangledown\}\alpha\{\bigtriangleup, q_i\} \Rightarrow \{\bigtriangledown, q_i\}\alpha\{\bigtriangleup\}$ for every $i = 0, \ldots, n$ and for each $\alpha \in \Sigma$;
   (e) $\{\bigtriangledown\}\alpha\{\circ, q_i\} \Rightarrow \{\bigtriangledown, q_i\}\alpha\{\circ\}$ for every $i = 0, \ldots, n$ and for each $\alpha \in \Sigma$;
   (f) $\{\bigtriangledown\}\alpha\{\circ, \Diamond\} \Rightarrow \{\bigtriangledown, \Box\}\alpha\{\circ\}$ for every $\alpha \in \Sigma$.

2. A *transfer* transformation is associated with the directive $\circ$ and consists of the replacements for every $\alpha \in \Sigma$ and for every $i = 0, 1, \ldots, n$:
   (a) $\{\circ\}\alpha\{\circ, q_i\} \Rightarrow \{\circ, q_i\}\alpha\{\circ\}$;
   (b) $\{\circ\}\alpha\{\bigtriangleup, q_i\} \Rightarrow \{\circ, q_i\}\alpha\{\bigtriangleup\}$;
   (c) $\{\circ\}\alpha\{\circ, \Diamond\} \Rightarrow \{\circ, \Diamond\}\alpha\{\circ\}$.

3. A *termination* transformation is associated with the directive $\Box$ and consists of the replacements for every $\alpha \in \Sigma$:
   (a) $\{\bigtriangledown, \Box\}\{\bigtriangleup\} \Rightarrow$;
   (b) $\{\bigtriangledown, \Box\}\alpha\{\circ\} \Rightarrow \alpha\{\circ, \Box\}$;
   (c) $\{\circ, \Box\}\alpha\{\circ\} \Rightarrow \alpha\{\circ, \Box\}$;

(d) $\{\circ, \Box\}\alpha\{\triangle\} \Rightarrow \alpha.$

4. No replacements are associated with the directives $\triangle, \diamond$ and q_i, for $1 \le i \le n$;
5. A ***substitution*** transformation is associated with the directive p_i; for every $i, 1 \le i \le n$, it is determined by the following rules:
   (a) if $\varphi_i \neq \Lambda$, then for every α from Σ, differing from the first symbol of φ_i, the replacements are the following:
   $\{\circ, p_i\}\alpha\{\circ\} \Rightarrow \{\circ\}\alpha\{\circ, p_i\}$;
   $\{\circ, p_i\}\alpha\{\triangle\} \Rightarrow \{\circ\}\alpha\{\triangle, q_i\}$
   $\{\nabla, p_i\}\alpha\{\circ\} \Rightarrow \{\nabla\}\alpha\{\circ, p_i\}$;
   $\{\nabla, p_i\}\alpha\{\triangle\} \Rightarrow \{\nabla\}\alpha\{\triangle, q_i\}$;
   $\{\nabla, p_i\}\{\triangle\} \Rightarrow \{\nabla\}\{\triangle, q_i\}$;
   (b) if π_i is a simple substitution formula, $\varphi_i \neq \Lambda$ and $\psi_i \neq \Lambda$, then the replacements are the following:
   $\{\nabla, p_i\}\overline{\varphi}_i \Rightarrow \{\nabla, q_o\}\overline{\psi}_i$;
   $\{\circ, p_i\}\overline{\varphi}_i \Rightarrow \{\circ, q_0\}\overline{\psi}_i$;
   (c) if π_i is a simple substitution formula, $\varphi_i \neq \Lambda$ and $\psi_i = \Lambda$, then the replacements are the following:
   $\{\nabla, p_i\}\overline{\varphi}_i\{\circ\} \Rightarrow \{\nabla, q_0\}$;
   $\{\nabla, p_i\}\overline{\varphi}_i\{\triangle\} \Rightarrow \{\nabla, q_0\}\{\triangle\}$;
   (d) if π_i is a simple substitution formula, $\varphi_i = \Lambda$ and $\psi_i \neq \Lambda$, then the replacements are the following for every $\alpha \in \Sigma$:
   $\{\nabla, p_i\}\alpha \Rightarrow \{\nabla, q_0\}\overline{\psi}_i\{\circ\}\alpha$;
   $\{\nabla, p_i\}\{\triangle\} \Rightarrow \{\nabla, q_0\}\overline{\psi}_i\{\triangle\}$;
   (e) if π_i is a simple substitution formula, $\varphi_i = \Lambda$ and $\psi_i = \Lambda$, then the replacements are the following:
   $\{\nabla, p_i\} \Rightarrow \{\nabla, q_0\}$;
   (f) if π_i is a final substitution formula, $\varphi_i \neq \Lambda$ and $\psi_i \neq \Lambda$, then the replacements are the following:
   $\{\nabla, p_i\}\overline{\varphi}_i \Rightarrow \{\nabla, \Box\}\overline{\psi}_i$;
   $\{\circ, p_i\}\overline{\varphi}_i \Rightarrow \{\circ, \diamond\}\overline{\psi}_i$;
   (g) if π_i is a final substitution formula, $\varphi_i \neq \Lambda$ and $\psi_i = \Lambda$, then the replacements are the following:
   $\{\nabla, p_i\}\overline{\varphi}_i\{\circ\} \Rightarrow \{\nabla, \Box\}$;
   $\{\nabla, p_i\}\overline{\varphi}_i\{\triangle\} \Rightarrow$;
   (h) if π_i is a final substitution formula, $\varphi_i = \Lambda$ and $\psi_i \neq \Lambda$, then the replacements are the following for every $\alpha \in \Sigma$:
   $\{\nabla, p_i\}\alpha \Rightarrow \{\nabla, \Box\}\overline{\psi}_i\{\circ\}\alpha$;
   $\{\nabla, p_i\}\{\triangle\} \Rightarrow \overline{\psi}_i$;

(i) if π_i is a final substitution formula, $\varphi_i = \Lambda$ and $\psi_i = \Lambda$, then the replacements are the following:
$\{\nabla, p_i\} \Rightarrow \{\nabla, \square\}$.

We denote as $\varphi_{i,j} \Rightarrow \psi_{i,j}$ the j-th replacement of the i-th transformation for some fixed numerations of transformations, associated with the directives from D, and the replacements within a transformation.

Function $f : \Sigma^* \to (\Sigma \cup A(D))^*$ is defined as follows. For every $\gamma \in \Sigma^*$ we consider as $f(\gamma)$ the word $\{\nabla, p_1\}\overline{\gamma}\{\Delta\}$. It is clear that $f(\gamma)$ is an annotated program. Now let us show that it is determininistic and realizes the processing of γ by an algorithm with the scheme π.

An annotation is called *simple*, if it contains less than two directives, and *complicated* otherwise.

The following properties hold by definition:

Proposition 1. *If $\varphi_{i,j}$ and $\varphi_{k,l}$ begin with the same complicated annotation, and either $i \neq k$, or $j \neq l$, then there is no word ω in the alphabet $\Sigma \cup A(D)$, such that $\varphi_{i,j} = \varphi_{k,l}\omega$.*

Proposition 2. *If $\varphi_{i,j}$ and $\varphi_{k,l}$ end with the same complicated annotation and either $i \neq k$, or $j \neq l$, then there is no word ω in the alphabet $\Sigma \cup A(D)$, such that $\varphi_{i,j} = \omega\varphi_{k,l}$.*

Proposition 3. *If $i \neq k$ or $j \neq l$, then there is no complicated annotation A and two words ω_1 and ω_2 in the alphabet $\Sigma \cup A(D)$, such that $\varphi_{i,j} = \omega_1 A$ and $\varphi_{k,l} = A\omega_2$.*

Theorem 1. *For every $\gamma \in \Sigma^*$ the annotated program $f(\gamma)$ is deterministic.*

Proof. Let α be a denotation for the annotated program $f(\gamma)$ for some $\gamma \in \Sigma^*$, let $\alpha_0 = \alpha, \alpha_1, \ldots, \alpha_k, \ldots$, be some processing of the annotated program α. It is clear that every α_i contains no more than one complicated annotation, this annotation is one of the following types for some i and j, $0 \leq i \leq n$, $0 < j \leq n$:

$$\{\nabla, p_j\}, \{\nabla, \square\}, \{\nabla, q_i\}, \{\circ, \diamond\}, \{\circ, \square\}, \{\circ, q_i\}, \{\circ, p_j\}, \{\Delta, q_i\}.$$

This property obviously holds for α_0 by definition of f and it is preserved in going from α_l to α_{l+1}, because the replaceable fragment of each replacement contains a complicated annotation, and its replacing fragment contains no more than one complicated annotation of the indicated type.

It follows herefrom and from the propositions 1–3 that the theorem is true. $\square$

Lemma 1. *Let $\alpha_0 = \alpha, \alpha_1, \ldots, \alpha_l, \ldots$ be the processing of the annotated program α of the type $\{\nabla, p_i\}\overline{\beta}\{\Delta\}$ for some i, $1 \leq i \leq n$, and some $\beta \in \Sigma^*$, which does not contain a subword φ_i. Then there exists l such that $BASE(\alpha_i) = BASE(\alpha)$ for all $0 \leq i \leq l$, and either $\alpha_l = \beta$, if $i = n$, or $\alpha_l = \{\nabla, p_{i+1}\}\overline{\beta}\{\Delta\}$, if $i < n$.*

Proof. Directly from the definition we have the following steps in the processing of α.

First we transfer from α_0 to $\alpha_j = \{\nabla\}\overline{\beta}\{\Delta, \underline{q_i}\}$ using the transformations 5(a), then we transfer from α_j to $\alpha_t = \{\nabla, q_i\}\overline{\beta}\{\Delta\}$ using the transformations 1(c), 1(d), 1(e), 2(a) and 2(b); and then we transfer from α_t either to $\{\nabla, p_{i+1}\}\overline{\beta}\{\Delta\}$ using the transformation 1(b), if $i < n$, or to $\{\nabla, \Box\}\overline{\beta}\{\Delta\}$ using the transformation 1(a), and then to β using the transformations 3(a), 3(b), 3(c), 3(d), if $i = n$.

Since none of the transformations listed above changes the base of the annotated program, the lemma is proved. □

Lemma 2. *Let $\alpha_0 = \alpha, \alpha_1, \ldots, \alpha_l, \ldots,$ be the processing of an annotated program of the form $\{\nabla, p_i\}\overline{\beta}\{\Delta\}$ for some $i, 1 \leq i \leq n$, and some $\beta \in \Sigma^*$ with a subword φ_i. Let ω be a word obtained from β after the replacement of the first appearance of a subword φ_i to ψ_i. Then there exist t and $l, 0 < t < l$ such that $BASE(\alpha_i) = BASE(\alpha)$ for all $0 \leq i < t, BASE(\alpha_i) = BASE(\omega)$ for all $t \leq i \leq l$, and either $\alpha_l = \omega$ if π_i is the final substitution formula, or $\alpha_l = \{\nabla, p_l\}\overline{\omega}\{\Delta\}$, if π_i is the simple substitution formula.*

Proof. Let $\varphi_i = \Lambda$. Directly from the definition we have the following steps in the processing of α. If π_i is simple, then first we transfer from α to $\alpha_t = \{\nabla, q_0\}\overline{\omega}\{\Delta\}$ using transformations 5(d) or 5(e), and next we transfer from α_t to $\alpha_s = \{\nabla, p_l\}\overline{\omega}\{\Delta\}$ using transformation 1(b). If π_i is the final substitution formula, then either we transfer from α directly to $\alpha_t = \omega$ using transformation 5(h), or first we transfer to $\alpha_t = \{\nabla, \Box\}\overline{\omega}\{\Delta\}$ using transformations 5(h), 5(g), and next we transfer from α_t to some $\alpha_l = \omega$ using transformation 3.

Let $\varphi_i \neq \Lambda, \beta = \varphi_i\tau$ and $\omega = \psi_i\tau$ for some $\tau \in \Sigma^*$. In this case we have the following steps of the processing of α. If π_i is the simple substitution formula, then first we transfer from α to $\alpha_t = \{\nabla, q_0\}\overline{\omega}\{\Delta\}$ using transformations 5(b) or 5(c), and next from α_t to $\alpha_s = \{\nabla, p_i\}\overline{\omega}\{\Delta\}$ using 1(b). If π_i is the final substitution formula, then either first we transfer from α to $\alpha_t = \{\nabla, \Box\}\overline{\omega}\{\Delta\}$ using transformations 5(f), 5(g), and next from α_t to some $\alpha_l = \omega$ using transformation 3, or we directly transfer to $\alpha_t = \omega$ using transformation 5(g).

Let $\varphi_i \neq \Lambda$, $\beta = \eta\psi_i\tau$ and $\omega = \eta\psi_i\tau$ for some $\eta, \tau \in \Sigma^*$, where $\eta \neq \Lambda$. In this case we have the following steps of the processing of α. If π_i

is the simple substitution formula, then first we transfer from α to some $\alpha_{t-1} = \{\nabla\}\overline{\eta}\{\circ, p_i\}\overline{\varphi\tau}\{\Delta\}$ using 5(a), and next to $\alpha_t = \{\nabla\}\overline{\eta}\{\circ, q_0\}\overline{\psi}_i\overline{\tau}\{\Delta\}$ using 5(b), and next to $\alpha_l = \{\nabla, p_1\}\overline{\omega}\{\Delta\}$ using 1 and 2. If π_i is the final substitution formula, then first we transfer from α to some $\alpha_{t-1} = \{\nabla\}\overline{\eta}\{\circ, p_i\}\overline{\varphi}_i\overline{\tau}\{\Delta\}$ using 5(a), then to $\alpha_t = \{\nabla\}\overline{\eta}\{\circ, \diamond\}\overline{\psi}_i\overline{\tau}\{\Delta\}$ using 5(f), next transfer to some $\alpha_s = \{\nabla, \square\}\overline{\omega}\{\Delta\}$ using 1(f) and 2(c), and at last to some $\alpha_l = \omega$ using transformation 3. □

Theorem 2. *Normal algorithm with the scheme π is realizable by the annotated program with the set of directives D and the function f.*

Proof. Consider some $\gamma \in \Sigma^*$. Let $\alpha_0, \alpha_1, \dots, \alpha_k, \dots$ be some processing of the annotated program $f(\gamma)$. Due to Theorem 1 it is unique. Let $\gamma_0 = \gamma_1, \dots, \gamma_l, \dots$ be a sequence of words obtained from γ in the processing by normal algorithm with the scheme π.

To prove the theorem it is enough to show that there exists an increasing sequence $s_0 = 0, s_1, \dots, s_l, \dots$ such that for any k the following properties are valid:

1) $\alpha_{s_k} = f(\gamma_k)$, if γ_k is obtained from γ_{k-1} using the substitution formula which is not final;

2) $\alpha_{s_k} = \gamma_k$, if either γ_k is obtained from γ_{k-1} using final substitution, or there is no substitution formula in π such that its left word is a subword of γ_k;

3) if γ_k exists, and $k > 0$, then there exists $t, s_{k-1} < t \le s_k$, such that $BASE(\alpha_{s_{k-1}}) = BASE(\alpha_i)$ for all $s_{k-1} \le i < t$ and $BASE(\alpha_{s_k}) = BASE(\alpha_j)$ for all $t \le j \le s_k$.

We prove this by induction with respect to k. For $k = 0$ the statement is clear. Assume it to be clear for every k, $0 \le k < l$, and consider $k = l > 0$.

The following cases are possible:

1) there is no word φ_i being a subword of γ_{k-1}, i.e., $\gamma_k = \gamma_{k-1} = \pi(\gamma)$;

2) $\gamma_k = \eta\psi_i\omega$ and $\gamma_{k-1} = \eta\varphi_i\omega$ for some simple substitution formula $\pi_i = \varphi_i \to \psi_i$;

3) $\gamma_k = \eta\psi_i\omega$ and $\gamma_{k-1} = \eta\varphi_i\omega$ for some final substitution formula $\pi_i = \varphi_i \to \psi_i$.

Due to Lemmas 1 and 2 and the inductive assumption, we obtain the validity of the properties in question for $k = l$ in every of the three cases. □

References

[1] V.N. Kasyanov, *Practical approach to program optimization*, Preprint of Computer Center of Siberian Division of the USSR Academy of Science, Novosibirsk, 1978, No. 135 (in Russian).

[2] *Reference Manual for the Ada Programming Language*, United States Department of Defence, 1983.

[3] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L.Jr. Steele, M.E. Zosel, *The High Performance Fortran Handbook*, The MIT Press, Canbridge, 1993.

[4] K.-P. Lohr, *Concurrency annotations for reusable software*, Comm. of the ACM, Vol. 36, No. 9, 1993, 81–89.

[5] M.S. Feather, *A survey and classification of some program transformation approaches and techniques*, In: Program Specification and Transformation, North-Holland, Amsterdam, 1987, 165–195.

[6] V.N. Kasyanov, *Optimizing Transformations of Programs*, Moscow, Nauka, 1988 (in Russian).

[7] V.N. Kasyanov, *Annotated program transformations*, Lecture Notes in Computer Science, Vol. 405, 1989, 171–180.

[8] V.N. Kasyanov, *Transformational approach to program concretization*, Theoretical Computer Science, Vol. 90, No. 1, 1991, 37–46.

[9] *New Generation Computing*, Special Issue: Selected Papers from Workshop on Patial Evaluation and Mixed Computation, Vol. 6, No. 2,3, 1988.

[10] A.A. Markov, *Theory of Algorithms*, Proc. of Math. Institute of the USSR Academy of Science, Moscow, Vol. 42, 1954 (in Russian).

[11] A.A. Markov, N.M. Nagornij, *Theory of Algorithms*, Moscow, Nauka, 1984 (in Russian).