Theoretical Computer Science 46 (1986) 175–195
North-Holland

GENERALIZED REGULAR EXPRESSIONS—A LANGUAGE FOR SYNTHESIS OF PROGRAMS WITH BRANCHING IN LOOPS

A. BRAZMA and E. KINBER

Computing Centre, Latvian State University, Riga, U.S.S.R.

Communicated by A. Salomaa
Received February 1985
Revised February 1986

Abstract. Regular expressions are generalized to the effect that, besides letters from a finite alphabet, they may also contain natural numbers. Within the framework of these generalized expressions the task of the inductive synthesis of programs from its sample run is formalized. Special automata recognizing the sets defined by generalized expressions are introduced, and their equivalence problem is shown to be recursively solvable. The set-theoretic properties of the sets defined by generalized expressions are also studied.

0. Introduction

Program synthesis from example computations has been extensively studied in recent years [1–6]. One of the possible approaches introduced by Barzdin [1, 3] is based upon a search of 'regular' substrings containing arithmetical progressions. In [1], a formal language and an algorithm that synthesizes programs containing **for**-loops are presented. In [2], a different version of the language is used to handle programs with **while**-loops. However, none of these programming languages supports branching (except on a truncation condition).

Let us consider sort-merge. Let a and b be two areas which we should merge into area c. Evidently, one of the most natural informal descriptions of the algorithm is the following one.

Example 0.1

input a, b;
if $a(1) \geqslant b(1)$? let it be; **then** $c(1) := a(1)$;
if $a(2) \geqslant b(1)$? let it be; **then** $c(2) := a(2)$;
if $a(3) \geqslant b(1)$? let it be not; **then** $c(3) := b(1)$;
if $a(3) \geqslant b(2)$? let it be not; **then** $c(4) := b(2)$;
if $a(3) \geqslant b(3)$? let it be not; **then** $c(5) := b(3)$;
if $a(3) \geqslant b(4)$? let it be; **then** $c(6) := a(3)$;
if $a(4)$ is empty, **then**
  $c(7) := b(5)$; $c(8) := b(6)$; $c(9) := b(7)$;
if $b(8)$ is empty, **then** halt;

if $b(5)$ is empty, **then**
  $c(7) := a(4)$; $c(8) := a(5)$;
if $a(6)$ is empty, **then** halt;
return c;

The next example is well known from the theory of computability—the algorithm enumerating domains of partial recursive functions. One of the possible descriptions used to explain the algorithm is the following.

Example 0.2. Let φ be a partial recursive function (p.r.f.). Let

$$f'(n, m) \underset{\text{Df}}{=} \begin{cases} \varphi(n) & \text{if the computation of } \varphi(n) \text{ halts after } m \text{ or less} \\ & \quad \text{than } m \text{ steps,} \\ \Lambda & \text{otherwise,} \end{cases}$$

where $n, m \in \mathbb{N}$ and Λ is a special symbol.

Let $f(k) = f'(n, m)$, where k is the Cantor number of the pair (n, m). Now, the general recursive function F enumerating the domain of the function φ can be computed as follows:

compute $f(1)$; let $f(1) = \Lambda$; compute $f(2)$; let $f(2) = \Lambda$;
compute $f(3)$; let $f(3) \neq \Lambda$;
  then $F(1) = f(3)$;
compute $f(4)$; let $f(4) = \Lambda$; compute $f(5)$; let $f(5) \neq \Lambda$;
  then $F(2) = f(5)$;
compute $f(6)$; let $f(6) \neq \Lambda$;
  then $F(3) = f(6)$;
compute $f(7)$; let $f(7) \neq \Lambda$; compute $f(8)$; let $f(8) = \Lambda$;
  ⋮
and so on.

Both of these inductive descriptions include branching ('if $a(i) \geqslant b(i)$?' in the first, and 'let $f(i) = \Lambda$' or '$f(i) \neq \Lambda$' in the second description; neither the language from [1] nor the language from [2] are fit for formalization. To allow for the formalization of such descriptions which include branching we will introduce a new language—the language of generalized regular expressions.

Here we understand 'formalization' in the following sense. Any generalized regular expression describes a definite set of words which in our context actually represents a set of sample computations of some algorithm. The generalized expression defining the set that is equal to the set of all possible sample computations of the given algorithm can be regarded as a program of this algorithm. Therefore, the language of generalized regular expressions can be regarded as a programming language. The task of synthesis is to find the correct program using only a finite set of sample computations.

In this paper we will first strictly define the language of generalized regular expressions and show how programs for the algorithms from Examples 0.1 and 0.2 can be written in this language. Then we will study the properties of the language and finally, discuss the problem of synthesis.

1. The definition of generalized regular expressions

Let $A = A' \cup \mathbb{N}$ be an alphabet where A' is a finite set of characters and $\mathbb{N}$ is the set of natural numbers. Numbers are represented in decimal notation. Substrings representing numbers are underlined (except for one-digit numbers) and the underlined string is regarded as one letter.

Let $X = \{x, y, \ldots, z\}$ be a finite set of so-called variables such that $A \cap X = \emptyset$ holds.

In the definition of generalized regular expressions we will use the following operation symbols: *: iteration, $\cup$: union, $^+$: taking of the next natural number, $+$: plus, $:=$: awarding of a value, and the following signs: „, (,), [, and]. We assume that none of these symbols is in the alphabet A (for distinction we use the bold-face type for them).

To define a generalized regular expression we first define its frame:

(i) if $a \in A$, then a is a frame;

(ii) if $x \in X$, then x, x^+, $[x+c]$ and $[x^+ + c]$ are frames;

(iii) if P and R are frames then $(P)^*$, $P \cup R$ and PR are frames;

(iv) if P is a frame, $x_1, x_2, \ldots, x_k \in X$, $\xi_1, \xi_2, \ldots, \xi_k \in X \cup \mathbb{N}$, $\xi_i^\times \in \{\xi_i, \xi_i^+\}$, then $((x_1 := \xi_1^\times, \ldots, x_k := \xi_k^\times)(P))$ is a frame;

(v) there are no other frames.

Now, if P is a frame, $x_1, \ldots, x_k \in X$ are all the variables of P and $c_1, c_2, \ldots, c_k \in \mathbb{N}$, then

$$(x_1 := c_1, x_2 := c_2, \ldots, x_k := c_k)(P)$$

is a generalized regular expression.

For example,

$$P_1 \underset{\text{Df}}{=} (x := 0)(ax^+)^*,$$

$$P_2 \underset{\text{Df}}{=} (x := 0, y := 0)((ax^+) \cup (by^+))^*,$$

$$P_3 \underset{\text{Df}}{=} (x := 0, y := 0, z := 0, u := 0)((ax^+yz^+) \cup (bxy^+z^+))^*((u := z)(cu^+)^*),$$

$$P_4 \underset{\text{Df}}{=} (x := 0, y := 0)((y := x^+)(ax[y^+ + 1])^*)^*,$$

$$P_5 \underset{\text{Df}}{=} (x := 0, y := 0, z := 0)((y := 0, z := x^+)b(azy^+)^*)^*$$

are generalized regular expressions if $x, y, z, u \in X$ and $a, b, c \in A$.

Usually, we will call generalized regular expressions programs. We will call any substring of a program in parentheses a subprogram. In the sequel we will restrict our class of programs to programs that contain no subprograms:

$$(Q_1 \cup Q_2 \cup \cdots \cup Q_r)^*,$$

where at least one Q_i is of the type

$$(x_1 := \mu_1, \ldots, x_k := \mu_k).$$

Let P be a program. We apply to P the following routine: First, we replace each subprogram $(R)^*$ by $RR \ldots R$ (k times), where k is an arbitrary natural number possibly different for different subprograms. Then we replace each subprogram $(R) \cup (S)$ either by R or by S. Thus we obtain some program $\bar{P}$ called the realization of the program P. For example,

$$\bar{P}_1 = (x := 0)(ax^+ax^+ax^+ax^+ax^+),$$

$$\bar{P}_2 = (x := 0, y := 0)(ax^+ax^+by^+by^+ax^+by^+by^+ax^+),$$

$$\bar{P}_3 = (x := 0, y := 0, z := 0, u := 0)(ax^+yz^+ax^+yz^+ax^+yz^+$$
$$bxy^+z^+bxy^+z^+bxy^+z^+bxy^+z^+)((u := z)(cu^+cu^+cu^+)),$$

$$\bar{P}_4 = (x := 0, y := 0)((y := x^+)(ax[y^+ + 1]ax[y^+ + 1]ax[y^+ + 1])(y := x^+)$$
$$(ax[y^+ + 1]ax[y^+ + 1])),$$

$$\bar{P}_5 = (x := 0, y := 0, z := 0)((y := 0, z := x^+)b(azy^+azy^+azy^+)(y := 0, z := x^+)$$
$$b(azy^+azy^+azy^+)(y := 0, \bar{z} = x^+)b(azy^+azy^+azy^+azy^+))$$

are realizations of the programs P_1–P_5.

Note, that the set of all realizations of the program P equals the regular set defined by the regular expression obtained from P, ignoring the semantics of operations $:=$, $^+$, and $+$.

In order to define the value of a realization $\bar{P}$, let us first enumerate all occurrences of each variable in $\bar{P}$. For example, for $\bar{P}_2$ we obtain

$$(x_{(1)} := 0, y_{(1)} := 0)(ax^+_{(2)}ax^+_{(3)}by^+_{(2)}by^+_{(3)}by^+_{(4)}ax^+_{(4)}by^+_{(5)}by^+_{(6)}ax^+_{(5)}).$$

Let us define the value $v(t)$ of the given entrance of variable t, as follows:

- if $x_{(i)}$ is in operation $x_{(i)} := \xi$, then $v(x_{(i)}) = v(\xi)$;
- if $x_{(i)}$ is in operation $x^+_{(i)}$, then $v(x_{(i)}) = v(x_{(i-1)}) + 1$;
- otherwise, $v(x_{(i)}) = v(x_{(i-1)})$.

Now we obtain the value of the realization $\bar{P}$ by replacing in $\bar{P}$ every occurrence $t_{(i)}$ of each variable t by its value $v(t_{(i)})$, computing all sums $[v(t_{(i)}) + c]$ and removing all square brackets and all subprograms of the sort $(x_1 := \mu_1, \ldots, x_k := \mu_k)$. For

example,

$$v(\bar{P}_1) = a1a2a3a4a5,$$

$$v(\bar{P}_2) = a1a2b1b2b3a3b4b5a4,$$

$$v(\bar{P}_3) = a101a202b213b224b235c6c7c8,$$

$$v(\bar{P}_4) = a13a14a15a24a25,$$

$$v(\bar{P}_5) = ba11a12a13ba21a22a23ba31a32a33a34.$$

The set of values of a program P is the set of values of all realizations of P. We denote the set of values of a program P by $V(P)$. We will call any element from $V(P)$ an example of the program P.

Now, let us present the programs (the generalized regular expressions) for sort-merge and an algorithm enumerating the domains at partial recursive functions.

Example 1.1 (*sort-merge*).

($x := 1, y := 1, z := 0, u := 0$**)**
(*input* a, b;
if $a(1) \geqslant b(1)$
  ((let it be; **then** $c(z^+) := a(x)$; **if** $a(x^+) \geqslant b(y)$**)**
   ∪(let it be not; **then** $c(z^+) := b(y)$; **if** $a(x) \geqslant b(y^+)$**))***
  ((let it be; **then** $c(z^+) := a(x)$**)**
   ∪(let it be not; **then** $c(z^+) := b(y)$**))**
if $a(x^+)$ is empty; **then** $(u := y)(c(z^+) := b(u^+))^*$
if $b(u^+)$ is empty; **then** halt;
if $b(y^+)$ is empty; **then** $(u := x)(c(z^+) := a(u^+))^*$
if $a(u^+)$ is empty; **then** halt;
return c;**)**

(We assume all the symbols used in the program to be in alphabet A, except for those which are in bold-face).

Evidently, Example 0.1 is the example of the program from Example 1.1.

Note that from the point of view of our formalism such strings as "$c(z^+) := a(x)$", "halt", "if" etc. do not have any semantics. This is important for synthesis because the supposed synthesizer as in [1, 2] can work entirely on syntactic level. On the other hand, to get an algorithm from the given program, we should interpret these strings as instructions.

Example 1.2 (*enumeration of the domains of partial recursive functions*).

$(x := 0, y := 0)$
  (compute $f(x^+)$;
    ((let $f(x) = \Lambda$;)
     $\cup$(let $f(x) \neq \Lambda$; **then** $F(y^+) = f(x^+)$))*
      ⋮
     and so on)

Example 0.2 is the example of this program.

The languages of [1] and [2] are based on formalizations of the notions 'dots', which helps to represent a 'regular string' containing arithmetical progressions. In generalized regular expressions this is achieved by the operations + and *. In comparison to [1, 2], new in our language is the operation union: "$\cup$", which supports the express branching. In [1, 2, 3], the algorithms are given that synthesize programs by one sufficiently long example. The algorithms are based on search for 'regular strings' and their substitution for appropriate 'dots terms'. Due to the operation of union, to synthesize a generalized regular expression by one example is evidently impossible. Therefore, the algorithm of synthesis should be more complicated here. To develop the theoretical framework for synthesis of generalized regular expressions we present special automata recognizing the sets defined by these expressions and study their properties.

Denote the set of programs by $\mathcal{P}$.

2. The definition of the class of automata

Let A and X be finite sets, $A \cap X = \emptyset$. Elements $x \in X$ will be called variables.

Definition 2.1. *The automaton's diagram* is a finite oriented graph in which arcs can be labelled. Depending on the type of labels, the arcs are divided into three classes: A-arcs, X-arcs and Λ-arcs. Namely,

(a) every A-arc is labelled by a single label $a \in A$;

(b) every X-arc is labelled by a single label $x \in X$ or $x + c$ with $c \in \mathbb{N}$; additionally, the operator $x \leftarrow x + 1$ can be associated with an X-arc labelled by the same $x \in X$ (or $x + c$);

(c) a finite sequence of the following operators

$x \leftarrow c$, where $c \in \mathbb{N}$, $x \in X$,

$x \leftarrow y$, where $x, y \in X$,

$x \leftarrow y + 1$, where $x, y \in X$

is associated with every Λ-arc so that every $x \in X$ can be presented in the left part of no more than one operator associated with the given arc;

(d) there is an initial state, say, q_0;

(e) a single Λ-arc leaves q_0; a sequence of operators

$$x_1 \leftarrow c_1, x_2 \leftarrow c_2, \ldots, x_k \leftarrow c_k,$$

where $\{c_1, c_2, \ldots, c_k\} \subset \mathbb{N}$, is associated with it.

The arc q_0q_1 in Fig. 1 below is a Λ-arc, the arc q_2q_1 is labelled by $x \in X$, and the operator $x \leftarrow x+1$ is associated with it.

The set $Q(D)$ of vertices of the automaton's diagram D is said to be the set of states. A subset $F(D) \subseteq Q(D)$ is called the set of accepting states.

We assume that every automaton's diagram D defines some automaton $\mathfrak{A} = \mathfrak{A}(D)$; in the sequel, we do not distinguish these two notions. Further, we also regard only automata that contain no Λ-loops (loops containing only Λ-arcs). Let $\mathscr{A}$ denote the class of the automata defined above.

Now let us fix an automaton $\mathfrak{A} \in \mathscr{A}$ and define the language $L(\mathfrak{A})$ accepted by $\mathfrak{A}$. For each arc l in $\mathfrak{A}$ let $\omega(l)$ denote the label of l (in case of a Λ-arc, $\omega(l) = \Lambda$).

Definition 2.2. By a *path* in $\mathfrak{A}$ we mean any path in the graph $D(\mathfrak{A})$ from the initial state q_0 to a state q (called the *end* of the path) in which some arcs (and loops, respectively) can be passed more than once. If the end of a path q is in $F(\mathfrak{A})$, then the path is called *accepting*.

Let $\alpha = l_1, l_2, \ldots, l_k$ be an arbitrary path in $\mathfrak{A}$ with the end q.

Definition 2.3 (*word* $u(\alpha)$). We define the value $u(x, i, \alpha) \in \mathbb{N}$ of a variable $x \in X$ after passing l_i as follows. Define $u(x, 0, \alpha) = 0$. Now, let $u(x, i, \alpha)$ be already defined. Then,

(1) if no operator containing the left x is associated with l_{i+1}, then $u(x, i+1, \alpha) =_{\mathrm{Df}} u(x, i, \alpha)$;

(2) if an operator $x \leftarrow c$ is associated with l_{i+1}, then $u(x, i+1, \alpha) =_{\mathrm{Df}} c$;

(3) if an operator $x \leftarrow y+c$, $c \in \{0, 1\}$ is associated with l_{i+1}, then: if there is an operator containing the left part y before $x \leftarrow y+c$ in the sequence of operators associated with l_{i+1}, then $u(x, i+1, \alpha) =_{\mathrm{Df}} u(y, i+1, \alpha)+c$, otherwise, $u(x, i+1, \alpha) =_{\mathrm{Df}} u(y, i, \alpha)+c$.

Now, let for every $i \in \{1, 2, \ldots, k\}$

$$u(i, \alpha) \underset{\mathrm{Df}}{=} \begin{cases} \omega(l_i) & \text{if } l_i \text{ is an } A\text{-arc}, \\ \Lambda & \text{if } l_i \text{ is a } \Lambda\text{-arc}, \\ u(x, i, \alpha)+c & \text{if } l_i \text{ is an } X\text{-arc and } \omega(l_i) = x+c. \end{cases}$$

Let $u(\alpha) = u(1, \alpha)u(2, \alpha) \ldots u(k, \alpha)$.

Definition 2.4 (*the language* $L(\mathfrak{A})$). The language accepted by the automaton $\mathfrak{A}$ is the following set: $\{u(\alpha) \mid \alpha \text{ is an accepting path}\}$.

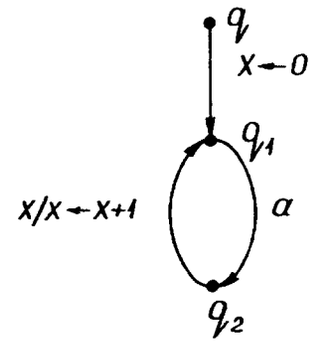

Fig. 1.

Example 2.5. (1) The automaton $\mathfrak{A}_1$ with $F(\mathfrak{A}) = \{q_1\}$ in Fig. 1 accepts the language $V(P_1)$.

(2) The automaton $\mathfrak{A}_2$ in Fig. 2 with $F(\mathfrak{A}_2) = \{q_2\}$ accepts $V(P_2)$.

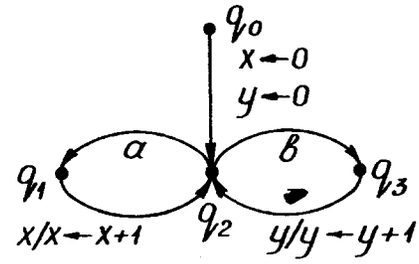

Fig. 2.

3. Programs and automata

There is an evident connection between the classes $\mathscr{P}$ and $\mathscr{A}$. Namely, a version of Kleene's theorem for regular languages holds for $\mathscr{P}$ and $\mathscr{A}$.

Let $\mathscr{L}(\mathscr{P}) = \{V(P) \mid P \in \mathscr{P}\}$ and $\mathscr{L}(\mathscr{A}) = \{L(\mathfrak{A}) \mid \mathfrak{A} \in \mathscr{A}\}$.

Theorem 3.1. $\mathscr{L}(\mathscr{P}) = \mathscr{L}(\mathscr{A})$.

Proof. In fact, the proof is a copy of Kleene's proof for regular languages (see, for instance, [6]). It is only to be noted that, given an automaton $\mathfrak{A}$, in order to construct a corresponding program P, the empty word must be associated with each Λ-arc. In addition, if the operator $x \leftarrow x+1$ is associated with an X-arc l labelled by $x+c$, then $x^+ + c$ must be associated with l; otherwise, $x+c$ must be associated with l. □

4. The inclusion problem for languages in $\mathscr{L}(\mathscr{A})$

Theorem 4.1. *The inclusion problem is decidable for arbitrary automata* $\mathfrak{A}, \mathfrak{N} \in \mathscr{A}$.

Proof. A path α in the diagram of an arbitrary automaton $\mathfrak{M}$ is said to be simple iff α passes an arbitrary loop in $\mathfrak{M}$ no more than one time.

Let us fix a simple path α_0 in $\mathfrak{A}$ which leads to $q \in F(\mathfrak{A})$. Let us consider the set S of paths α in $\mathfrak{A}$ such that

(1) the set of loops belonging to path α is equal to that same set for α_0, and

(2) α differs from α_0 only by the number of by-passes of the loops. Let $\mathfrak{U}(S) = \{u(\alpha) \mid \alpha \in S\}$. Our goal is to construct an algorithm that checks the inclusion $U(S) \subseteq L(\mathfrak{N})$. Let LF be the set of all linear expressions $c_0 + c_1 z_1 + c_2 z_2 + \cdots + c_k z_k$, where $c_0, c_1, \ldots, c_k \in \mathbb{N}$, $k \in \mathbb{N}$, and $z_1, z_2, \ldots, z_K$ are variables defined on $\mathbb{N}$.

We have to define a procedure that, given S and $\mathfrak{N}$, constructs two finite trees; the vertices of the trees will correspond to states of $\mathfrak{A}$ (belonging to the S-part) and $\mathfrak{N}$, respectively; the arcs of the trees correspond to arcs in α_0 and $\mathfrak{N}$; some expressions $h \in \mathrm{LF}$ will correspond to every X-arc in the trees.

To illustrate some necessary notions we will use the following example (Fig. 3).

So, let α_0 be the simple path that defines S (say,

$$\alpha_0 = l_1 l_2 l_3 l_4 l_5 l_7 l_9 l_6 l_8 l_{10} l_{18} l_2 l_5 l_{11} l_{12} l_{13} l_{14} l_{12} l_{13} l_{15} l_{16} l_{13} l_{15} l_{17}$$

in our example). The path α_0 defines some fragment $\mathfrak{A}(\alpha_0)$ of the diagram $\mathfrak{A}$ (see Fig. 3). Select in $\mathfrak{A}(\alpha_0)$ the simple loops—sequences of arcs and vertices containing only one vertex exactly twice. (In our example,

$$C_1 = l_2 l_5 l_{18}, \qquad C_2 = l_3 l_4, \qquad C_3 = l_7 l_9,$$

$$C_4 = l_6 l_8 l_{10}, \qquad C_5 = l_{12} l_{13} l_{14}, \qquad C_6 = l_{13} l_{15} l_{16}.)$$

Now, for every vertex p on α_0 that lies on a loop, let us define some order $C_1^p, C_2^p, \ldots, C_m^p$ for the simple loops including p (for example, for p_4 in Fig. 3, say,

$$C_1 = l_{18} l_2 l_5, \qquad C_2 = l_7 l_9, \qquad C_3 = l_6 l_8 l_{10}).$$

Let $X = \{x_1, x_2, \ldots, x_{m_1}\}$ be the X-alphabet of $\mathfrak{A}$ and $Y = \{y_1, y_2, \ldots, y_{m_2}\}$ be the corresponding alphabet of $\mathfrak{N}$. Let $G_1^0 = (g_1, g_2, \ldots, g_{m_1})$ and $G_2^0 = (g'_1, g'_2, \ldots, g'_{m_2})$

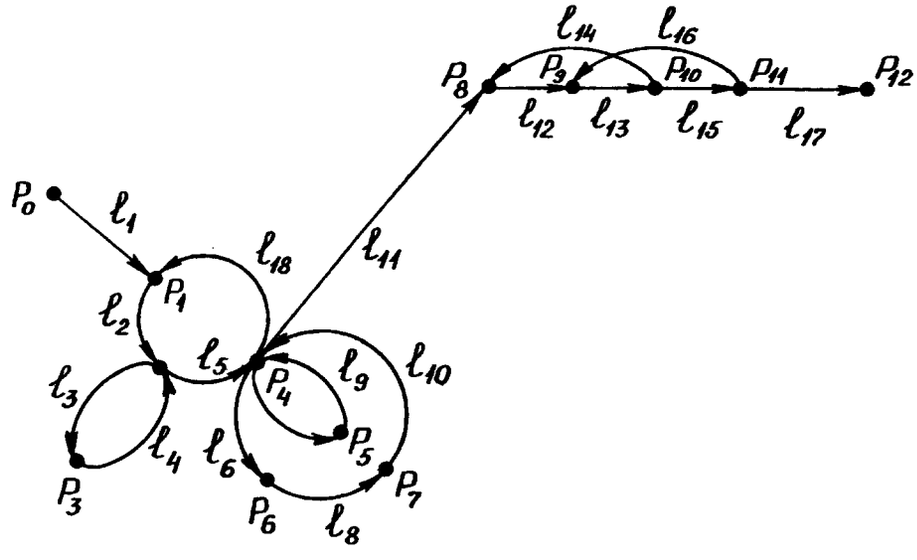

Fig. 3.

be two trains of expressions $g \in \mathrm{LF}$. Let $\alpha = d_1 d_2 \dots d_e$ be an arbitrary path in $\mathfrak{A}(\alpha_0)$ with initial vertex p. Using g_i as a value of x_i, $1 \leqslant i \leqslant m_1$, in the initial vertex p of α, our intention is to define an element in LF that is the value of x_i at the end of α. By the way, we will define values in LF for all variables associated with arcs in α. Then it will be possible to associate with α some sequences of A-symbols and expressions in LF that are, in some sense, similar to $u(\alpha)$.

So let us define, for every $j \in \{1, 2, \dots, m_1\}$, $g_j(d_0, G_1^0) = g_j$ and for every arc d_r, $1 \leqslant r \leqslant e$,

$$g_j(d_r, G_1^0) \underset{\mathrm{Df}}{=} \begin{cases} g_j(d_{r-1}) + 1 & \text{if the operator } x_j \leftarrow x_j + 1 \text{ is associated with } d_r, \\ c & \text{if the operator } x_j \leftarrow c \text{ is associated with } d_r, \\ g_i(d_{r-1}) & \text{if the operator } x_j \leftarrow x_i \text{ is associated with } d_r \text{ and no operator before } x_j \leftarrow x_i \text{ in the sequence associated with } d_r \text{ has the left part } x_i, \\ g_i(d_{r-1}) + 1 & \text{if } x_j \leftarrow x_i + 1 \text{ is associated with } d_r \text{ and no operator before } x_j \leftarrow x_i + 1 \text{ in the sequence associated with } d_r \text{ has the left part } x_i, \\ g_i(d_r) & \text{if the operator } x_j \leftarrow x_i \text{ is associated with } d_r \text{ and some operator with the left part } x_i \text{ associated with } d_r \text{ precedes } x_j \leftarrow x_i, \\ g_i(d_r) + 1 & \text{if } x_j \leftarrow x_i \text{ in the above case is replaced by } x_j \leftarrow x_i + 1, \\ g_j(d_{r-1}) & \text{otherwise.} \end{cases}$$

Now, set

$$b(d_r, G_1^0) \underset{\mathrm{Df}}{=} \begin{cases} \Lambda & \text{if } d_r \text{ is a } \Lambda\text{-arc}, \\ a & \text{if } d_r \text{ is an } A\text{-arc and } a = \omega(d_r), \\ c + g_j(d_r, G_1^0) & \text{if } d_r \text{ is an } X\text{-arc and } \omega(d_r) = c + x_j. \end{cases}$$

(Recall that $\omega(d)$ is the label associated with d.) Finally, set

$$\bar{b}(\alpha, G_1^0) = b(d_1, G_1^0) b(d_2, G_1^0) \dots b(d_e, G_1^0).$$

Define also

$$G_1(\alpha, G_1^0) = (g_1(d_e, G_1^0), g_2(d_e, G_1^0), \dots, g_{m_1}(d_e, G_1^0)).$$

In a similar way, for an arbitrary path β in $\mathfrak{N}$ with the initial vertex q, $\bar{b}(\beta, G_2^0)$ and $G_2(\beta, G_2^0)$ can be defined.

Now, let q be a vertex in $\mathfrak{N}$, p and p' be two consecutive vertices on the path α_0, G_1^0, G_2^0 be two trains defined as above. We will define an auxiliary procedure $\mathrm{PROC}_1(p, p', q, G_1^0, G_2^0)$ that, given arbitrary p, p', q, G_1^0, G_2^0, constructs two finite trees R and T and sets $G_1(\alpha, G_1^0)$ and $G_2(\beta, G_2^0)$ for paths α in R, respectively paths β in T. The procedure also glues together initial vertices and leaves (final vertices) in, respectively, R and T. Vertices and arcs in R and T will correspond to vertices and arcs in respectively, $\mathfrak{A}$ and $\mathfrak{N}$.

Procedure $\mathrm{PROC}_1(p, p', q, G_1^0, G_2^0)$. Glue together p and q.

Let α be the fragment of α_0 from p to p'. Find $\bar{b}(\alpha, G_1^0) = a_1 a_2 \ldots a_4$. Then search for all paths β in $\mathfrak{N}$ with the initial state q such that

$$\bar{b}(\beta, G_2^0) = \bar{b}(\alpha, G_1^0) \tag{1}$$

holds. If no such β is found, then let R be the vertex p and T be the vertex q; p in R and q in T are marked as *dead* vertices. Otherwise, let $R = \alpha$ and T be the tree with the initial vertex q that contains all found paths β (note that the set of all different β is finite since the automaton $\mathfrak{N}$ contains no Λ-loops). Find $G_1(\alpha, G_1^0)$ and $G_2(\beta, G_2^0)$ for all β in T. Glue together the leaf in R with the leaves in T. Stop the procedure.

Now, let q be a state of $\mathfrak{N}$, and let p be a vertex on α_0 in $\mathfrak{A}$ such that at least one loop contains it. We will define an auxiliary procedure $\mathrm{PROC}_2(p, q, G_1^0, G_2^0)$ which, given arbitrary p, q, G_1^0, G_2^0, constructs two finite trees R and T, trains $G_1(\alpha, G_1^0)$ for paths α in R with the ends in leaves, and trains $G_2(\alpha, G_2^0)$ for paths β in T with the ends in leaves; some vertices in R and T may be glued together.

PROC_2 is the main tool of our proof, so we will describe this procedure firstly using one simple example. The example is chosen so that all main ideas of the procedure can be demonstrated.

So, let the fragment of the automaton $\mathfrak{A}$ reachable from p be the following automaton $\mathfrak{A}'$ (Fig. 4). Let the fragment of $\mathfrak{N}$ reachable from q be the following automaton $\mathfrak{N}'$ (Fig. 5).

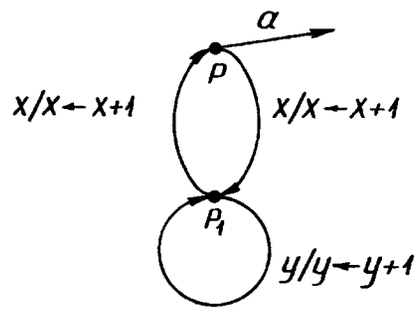

Fig. 4.

Let also $G_1 = (0, 2)$ and $G_2 = (0, 2)$ (we omit superscripts in $G_1^{(n)}$ and $G_2^{(n)}$ when describing only the first stage of the procedure).

First, we will try to explain the general idea of the procedure. In the first stage the procedure finds the linear fragments of $\mathfrak{N}'$ corresponding (i.e., defining the same word) to the shortest path from p to p' in $\mathfrak{A}'$. Then the procedure passes the loop from p_1 to p_1 in $\mathfrak{A}'$ one time and finds the corresponding extensions of the paths in $\mathfrak{N}'$—the loops with initial vertices q_1 and q_2 will be these extensions. But now the trains of linear forms G_1 in the vertices corresponding to the beginning and the end of one pass around the loop in Fig. 6 are equal. Similarly, the trains G_2 in the

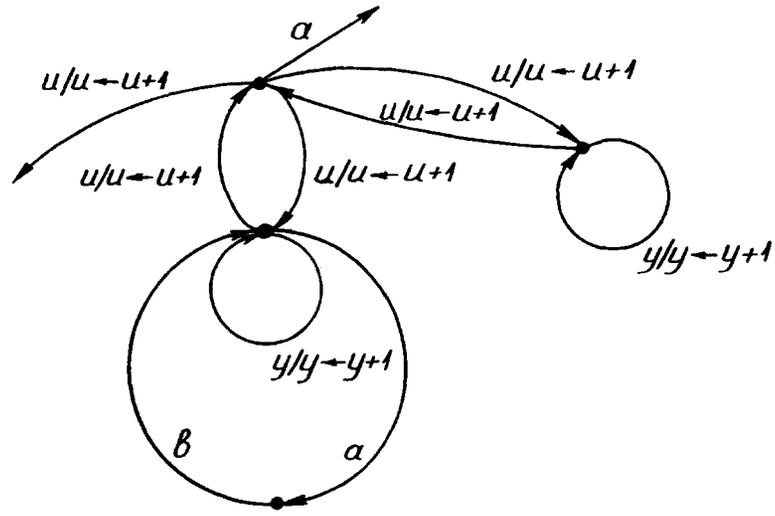

Fig. 5.

vertices corresponding to the beginnings q_1, q_2 and the ends q_1, q_2 of similar loops are equal. In this case, the procedure changes G_1 and G_2 to $(0, 2+z_1)$, where z_1 is a variable defined by the path from p_1 to p_1 in $\mathfrak{A}'$ and the paths q_1q_1 and q_2q_2 in $\mathfrak{N}$ (the coefficient 1 in $1 \cdot z_1$ shows that the variable y changes its value for 1 on the paths p_1p_2, q_1q_1, q_2q_2). Then the procedure removes the fragments corresponding to the loop in Fig. 1 from the trees constructed above. Further, the procedure acts similarly.

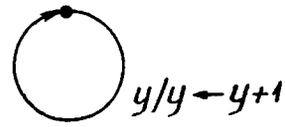

Fig. 6.

Now we will describe the procedure's working more explicitly by using our example.

The procedure acts on $\mathfrak{A}'$ and $\mathfrak{N}'$ as follows:

Step 1. PROC_2 applies $\text{PROC}_1(p, p_1, q, G_1, G_2)$. The following trees $R^{(1)}$ and $T^{(1)}$ (Fig. 7) will be obtained. The vertex $p^{(1)}$ also will be glued together with $q_1^{(1)}$, $q_2^{(1)}$, $q_3^{(1)}$. $G_1(p^{(1)}) = G_1(p)$ and $G_2(q_i^{(1)}) = G_2(q)$, $i = 1, 2, 3$.

Step 2. PROC_2 is (recursively) applied to all trains $(p^{(1)}, q_i^{(1)}, G_1(p^{(1)}), G_2(q_i^{(1)}))$, $i = 1, 2, 3$. The trees $R^{(2)}$ and $T^{(2)}$ (Fig. 8) will be obtained. The vertex $p^{(2)}$ will be

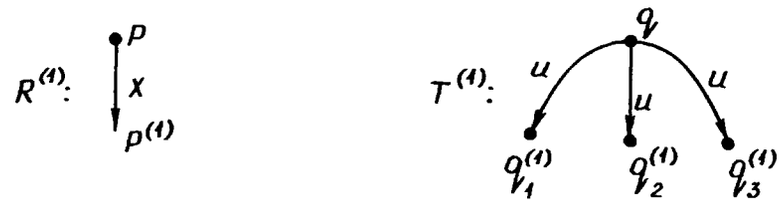

Fig. 7.

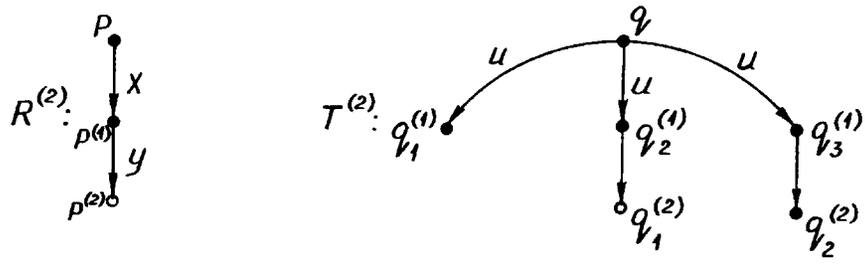

Fig. 8.

glued together with $q_1^{(2)}$ and $q_2^{(2)}$, but $q_1^{(1)}$ will be marked as dead. $G_1(p^{(2)}) = G_1(p_1)$ and $G_2(q_1^{(2)}) = G_2(q_2^{(2)}) = G_2(q_2^{(1)}) = G_2(q_3^{(1)})$ again, but in this case, these equalities are used in order to:

(a) remove the path $p^{(1)}p^{(2)}$ from $R^{(2)}$ and identify $p^{(2)}$ with $p^{(1)}$;

(b) remove the paths $q_2^{(1)}q_1^{(2)}$ and $q_3^{(1)}q_2^{(2)}$ from $T^{(2)}$ and identify $q_1^{(2)}$ with $q_2^{(1)}$ and $q_2^{(2)}$ with $q_3^{(1)}$ (*remark*: therefore, the trees in Fig. 7 will be obtained again);

(c) change $G_1(p^{(1)})$, $G_2(q_2^{(1)})$, $G_2(q_3^{(1)})$ to $(0, z_1+2)$, where z_1 is a new variable.

Step 3. PROC_1 is applied to the trains $(p^{(1)}, p_1, q_2^{(1)}, G_1(p^{(1)}), G_2(q_i^{(1)}))$, $i=2, 3$. Now, the trees $R^{(3)}$ and $T^{(3)}$ (Fig. 9) will be obtained, and the vertex $p^{(3)}$ will be glued together with $q_i^{(3)}$, $i=1, 2, 3, 4, 5, 6$. We also have:

$$\begin{aligned} &G_1(p^{(3)}) = (0, 2+z_1) = G_1(p^{(1)}), \\ &G_2(q_i^{(3)}) = (0, 2+z_1) = G_2(q_2^{(1)}) = G_2(q_3^{(1)}), \quad i=1, 2, 3, 4, 5, 6. \end{aligned} \tag{2}$$

The vertices $p^{(1)}$ and $p^{(3)}$ correspond to one vertex in $\mathfrak{A}'$, and similarly, $q_i^{(3)}$, $i=1$, 2, 3, 4, 5, 6 and $q_2^{(1)}$, $q_3^{(1)}$ correspond to one vertex in $\mathfrak{N}'$. In this case, in view of the equalities (2), the procedure

(a) removes the path $p^{(1)}p^{(3)}$ from $R^{(3)}$ and identifies $p^{(3)}$ with $p^{(1)}$;

(b) removes the paths $q_2^{(1)}q_1^{(3)}$, $q_2^{(1)}q_2^{(3)}$, $q_2^{(1)}q_3^{(3)}$, $q_3^{(1)}q_4^{(3)}$, $q_3^{(1)}q_5^{(3)}$, $q_3^{(1)}q_6^{(3)}$ from $T^{(3)}$ and identifies $q_1^{(3)}$, $q_2^{(3)}$, $q_3^{(3)}$ with $q_2^{(1)}$ and $q_4^{(3)}$, $q_5^{(3)}$, $q_6^{(3)}$ with $q_3^{(1)}$;

(c) changes $G_1(p^{(1)})$, $G_2(q_2^{(1)})$, $G_2(q_3^{(1)})$ to $(2z_2, 2+z_1)$, where z_2 is defined by the train of the paths in $\mathfrak{A}'$ and $\mathfrak{N}'$ that correspond to the train of the paths removed

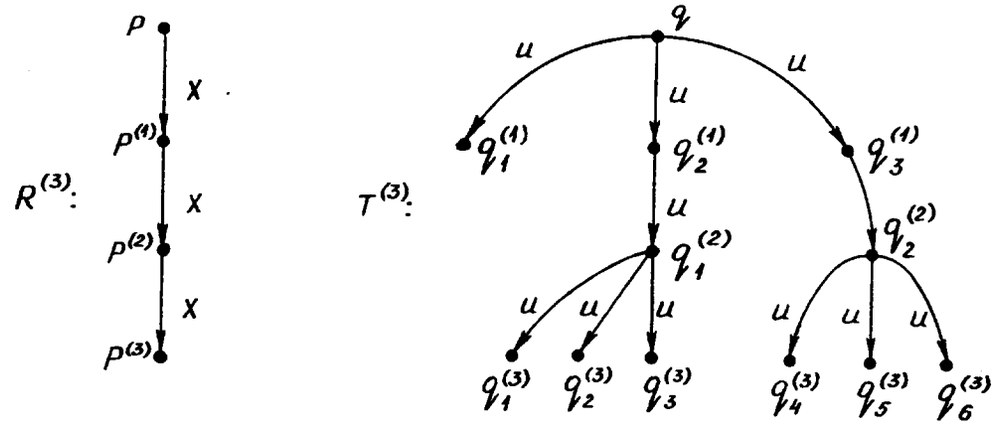

Fig. 9.

in (a) and (b). The coefficient 2 shows that the variable x is increased by 2 on all removed paths.

Therefore, the trees in Fig. 7 are obtained again, but there are new G_1 and G_2 for their leaves.

Steps 4, 5, 6. The similar sequence of actions is applied to the trees obtained above (Steps 1, 2, and 3). After Step 6, we will again obtain trees like in Fig. 7. Moreover, in this case,

$$G_1(p^{(1)}) = G_2(q_2^{(1)}) = G_2(q_3^{(1)}) = (2z_2, 2+z_1) \tag{3}$$

as after the Step 3. In view of the equalities of the trees and equalities (3), the procedure stops its work.

Now we will give the formal definition of PROC_2.

Procedure $\mathrm{PROC}_2(p, q, G_1, G_2)$. Let k denote the number of simple loops passing through p. Assume that, before a step n,

(a) the trees R_{n-1} and T_{n-1} have been constructed;

(b) every living (not-dead) leaf in T_{n-1} is glued together with a single leaf in R_{n-1};

(c) the leaves of the subtrees $R_j, j<n$, and $T_j, j<n$, in R_{n-1} and T_{n-1} respectively, are marked;

(d) some leaves in R_{n-1} and T_{n-1} are marked as dead;

(e) for every path α in R_{n-1} (T_{n-1}) with living end, the train $G_1^{n-1}(\alpha)$ (respectively $G_2^{n-1}(\alpha)$) is defined.

Step n. Perform the procedure described below for every $i \leqslant k$, every living leaf $\bar{q}$ in T_{n-1}, and the leaf $\bar{p}$ in R_{n-1} glued together with $\bar{q}$.

Procedure $\mathrm{PR}(\bar{q}, \bar{p}, i)$. Let $\bar{\alpha}$ be the path in R_{n-1} that leads from the root to $\bar{p}$, and let $\bar{\beta}$ be the path in T_{n-1} leading from the root to $\bar{q}$. The vertex $\bar{p}$ in R_{n-1} evidently corresponds to $\bar{p}$ in $\mathfrak{A}$. Similarly for $\bar{q}$. Find the first vertex p' (p_1 in our example above) after $\bar{p}$ on c_i^p such that p' lays on at least one loop equal to no $C_1^p, C_2^p, \ldots, C_k^p$ (in our example it will be p_1). Compute $\mathrm{PROC}_1(\bar{p}, p', \bar{q}, G_1^{n-1}(\bar{\alpha}), G_2^{n-1}(\bar{\beta}))$. Then, some trees R, T (see Fig. 7) and the trains $G_1(\alpha, G_1^{n-1}(\bar{\alpha}))$ and $G_2(\beta, G_2^{n-1}(\bar{\beta}))$ are obtained. If, computing PROC_1, paths β satisfying (1) have not been found, then stop $\mathrm{PROC}_2(p, q, G_1^0, G_2^0)$, and set $R_{n,i} =_{\mathrm{Df}} R_{n-1}$, $T_{n,i} =_{\mathrm{Df}} T_{n-1}$. Otherwise, set $R_{n,i}^{(1)} =_{\mathrm{Df}} R$ $T_{n,i}^{(1)} =_{\mathrm{Df}} T$. Then, for each path α in $R_{n,i}^{(1)}$ with end in a leaf $\tilde{p}$, set $G_1^n(\tilde{p}) =_{\mathrm{Df}} G_1(\alpha, G_1^{n-1}(\bar{\alpha}))$ and, for each path β in $T_{n,i}^{(1)}$ with the end in a leaf $\tilde{q}$, set $G_2^n(\tilde{q}) =_{\mathrm{Df}} G_2(\beta, G_2^{n-1}(\bar{\beta}))$. Further, for every $\tilde{p}$ mentioned above and for every $\tilde{q}$ glued with $\tilde{p}$, perform $\mathrm{PROC}_2(\tilde{p}, \tilde{q}, G_1^n(\tilde{p}), G_2^n(\tilde{q}))$ (again, some vertices in $\mathfrak{A}$ and $\mathfrak{N}$ correspond to $\tilde{p}$ and $\tilde{q}$; in our example above, this stage is Step 2 for instance). Now, some trees, say

$$R = R(\tilde{p}, \tilde{q}, G_1^n(\tilde{p}), G_2^n(\tilde{q}))$$

(for example, the path $p^{(1)}p^{(2)}$ in Fig. 8) and trains $G_1^n(s) = G_1(\alpha', G_1^n(\tilde{p}))$ for the paths α' in R with ends in the leaves s are obtained. Also, some trees, say

$$T = T(\tilde{p}, \tilde{q}, G_1^n(\tilde{p}), G_2^n(\tilde{q}))$$

(respectively, the paths $q_2^{(1)}q_1^{(2)}$ and $q_3^{(1)}q_2^{(2)}$ in Fig. 8) and trains $G_2^n(t) = G_2(\beta', G_2^n(\tilde{q}))$ for paths β' in T with ends in the leaves t are obtained. Identify the roots of all $R(\tilde{p}, \tilde{q}, G_1^n(\tilde{p}), G_2^n(\tilde{q}))$ with p in $R_{n,i}^{(1)}$ and the roots of all $T(\tilde{p}, \tilde{q}, G_1^n(\tilde{p}), G_2^n(\tilde{q}))$ with $\tilde{q}$ in $T_{n,i}^{(1)}$. Denote the trees obtained by the above action by $R_{n,i}^{(2)}$ and $T_{n,i}^{(2)}$, respectively.

Further, find the next vertex p'' after p' on the loop C_i^p such that at least one loop going through p'' is equal to no $C_1^p, C_2^p, \ldots, C_k^p$. Compute

$$\mathrm{PROC}_1(p', p'', q', G_1^n(p'), G_2^n(q'))$$

for every leaf p' in $R_{n,i}^{(2)}$ and for every leaf q' in $T_{n,i}^{(2)}$ glued together with q' (actually, perform PROC_1 for p' and q' in $\mathfrak{A}$ and $\mathfrak{N}$ corresponding to p' in $R_{n,i}^{(2)}$ and q' in $T_{n,i}^{(2)}$). Now, the trees $R(p', q', G_1^n(p'), G_2^n(q'))$ (see Fig. 9) and the trains $G_1^n(s) = G_1(\alpha, G_1^n(p'))$ are obtained for the paths α' with ends in the leaves s of the above trees. Also the trees $T(p', q', G_1^n(p'), G_2^n(q'))$ (again, see Fig. 9) and the trains $G_2^n(t) = G_2(\beta', G_2^n(q'))$ are obtained for the paths β' with ends in the leaves t.

Identify the roots of all $R(p', q', G_1^n(p'), G_2^n(q'))$ with p' in $R_{n,i}^{(2)}$, and the roots of all $T(p', q', G_1^n(p'), G_2^n(q'))$ with q' in $T_{n,i}^{(2)}$. Denote the obtained trees by $R_{n,i}^{(3)}$ and $T_{n,i}^{(3)}$, respectively. Now perform $\mathrm{PROC}_2(\tilde{p}, \tilde{q}, G_1^n(\tilde{p}), G_2^n(\tilde{q}))$ for every leaf $\tilde{p}$ in $R_{n,i}^{(3)}$ and every leaf $\tilde{q}$ in $T_{n,i}^{(3)}$ glued together with $\tilde{p}$. Then, repeating the above actions, define $R_{n,i}^{(4)}$, $T_{n,i}^{(4)}$, and the trains $G_1^n(p')$, $G_2^n(q')$ for all leaves p' in $R_{n,i}^{(4)}$ and all the leaves q' in $T_{n,i}^{(4)}$ glued together with p'.

Then, define similarly

$$p''', R_{n,i}^{(5)}, T_{n,i}^{(5)}, R_{n,i}^{(6)}, T_{n,i}^{(6)}, \ldots, p^{(k)}, R_{n,i}^{(2k-1)}, T_{n,i}^{(2k-1)}, R_{n,i}^{(2k)}, T_{n,i}^{(2k)}, \ldots.$$

Finally, for some m, $p^{(m)}$ will coincide with p. In this case, find only $R_{n,i}^{(2m-1)}$, $T_{n,i}^{(2m-1)}$ and $G_1^n(\tilde{p})$, $G_2^n(\tilde{q})$ for leaves $\tilde{p}$ in $R_{n,i}^{(2m-1)}$ and $\tilde{q}$ in $T_{n,i}^{(2m-1)}$. Define $R_{n,i}^{(2m-1)} = R_{n,i}$, $T_{n,i}^{(2m-1)} = T_{n,i}$. Stop $\mathrm{PR}(\bar{q}, \bar{p}, i)$.

Procedure $\mathrm{PROC}_2(p, q, G_1, G_2)$ (*continued*). Now, for every i, identify the root of $R_{n,i} = R_{n,i}(\bar{p}, \bar{q})$ with $\bar{p}$ in R_{n-1} and the root of $T_{n,i} = T_{n,i}(\bar{p}, \bar{q})$ with $\bar{q}$ in T_{n-1}. Denote the obtained trees by R_n and T_n.

Now, let p' be an arbitrary leaf in R_n. Let α be the path from p to p' in R_n. Let $\beta_1, \beta_2, \ldots, \beta_r$ be the paths in T_n that go from the root to the leaves $q_1', q_2', \ldots, q_r'$ glued together with p'. Check whether there is a vertex p'' on α such that

(a) p'' is a leaf in some of the trees R_j, $j < n$;

(b) p'' is glued together with vertices $q_1'', q_2'', \ldots, q_s''$ in T_n such that $q_1'', q_2'', \ldots, q_s''$ and $q_1', q_2', \ldots, q_r'$ correspond to the same states in $\mathfrak{N}$; and

(c) the following equalities hold

$$G_1^j(p'') = G_1^j(p'), \tag{4}$$

$$\forall m \leqslant r \; \exists k \leqslant s: (G_2^j(q_m') = G_2^j(1_k'')). \tag{5}$$

If such a p'' is not found, then check whether R_n coincides with R_s for some $s < n$. If such an s exists, then stop the procedure, otherwise go to step $n+1$.

Now, if a vertex p'' is found, let p'' be among the vertices nearest to p' in α. Let $\bar{\alpha}$ denote the fragment of α from p'' to p'. Let $\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_r$ denote the fragments of $\beta_1, \beta_2, \ldots, \beta_r$ such that their beginnings are glued together with p'' and their ends are glued together with p'. Remove the fragment $\bar{\alpha}$ from R (leaving the part of $\bar{\alpha}$ that contains paths to other leaves) and glue together (identify) the vertices p' and p''. Then, in a similar way, remove fragments $\bar{\beta}_m$, $m = 1, 2, \ldots, r$, from every β_m (we performed these actions in our example in Steps 2 and 3).

Note now that $\bar{\alpha}$ and $\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_r$ correspond to paths $\tilde{\alpha}$ in $\mathfrak{A}$ and $\tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r$ in $\mathfrak{N}$, respectively (consisting of the same arcs). Associate some variables z $(= z(\tilde{\alpha}, \tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r))$ with the train $(\tilde{\alpha}, \tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r)$. For every $j \in \{1, 2, \ldots, m_1\}$, let g_j be the jth element in $G_1^n(p')$. Two cases are possible for the variable x_j:

Case 1: there is a number $n_0 \in \mathbb{N}$ such that, for every $n \in \mathbb{N}$, if n_1 is the value of x_j at the beginning of $\tilde{\alpha}$, then $n_1 + n_0$ is the value of x_j at the end of $\tilde{\alpha}$;

Case 2: the value of x_j at the end of $\tilde{\alpha}$ does not depend on its value at the beginning of $\tilde{\alpha}$ (this means, in fact, that a sequence of operators $x_{j_1} := c_1$, $x_{j_2} := x_{j_1}, \ldots, x_j := x_{j_s}$ changes the value of x_j when passing through α).

It is not difficult to see that Cases 1 and 2 can be checked effectively. If Case 1 holds and cz is not a member of g_j, then exchange g_j to $g_j + cz$.

Similarly, regarding the behaviour of all variables g_j', $j = \{1, 2, \ldots, m_2\}$, exchange $G_2^n(q')$ for all vertices q' in T_n glued together with p'.

Further, if R_n coincides with R_j for some $j < n$, then stop the procedure. Otherwise, go to step $n+1$.

This ends the definition of $\mathrm{PROC}_2(p, q, G_1, G_2)$.

Proof of Theorem 4.1 (*continued*). Now, let $R = \bigcup_{n \in \mathbb{N}} R_n$ and $T = \bigcup_{n \in \mathbb{N}} T_n$.

Lemma 4.2. *For every* p, q, G_1, G_2, $\mathrm{PROC}_2(p, q, G_1, G_2)$ *terminates.*

Proof. It follows from the definition of PROC_2 that every new member cz in an arbitrary form g in

$$\bigcup_{n \in \mathbb{N}} \Bigg(\bigcup_{\substack{(p', q') \text{ is a pair} \\ \text{of leaves glued} \\ \text{together on } R_n \text{ and } T_n}} (G_1^n(p') \cup G_2^n(q')) \Bigg) \tag{6}$$

is defined by a train $(\tilde{\alpha}, \tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r)$ of paths $\tilde{\alpha}$ in $\mathfrak{A}$ and $\tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r$ in $\mathfrak{N}$ such that $\tilde{\alpha}$ is a loop in $\mathfrak{A}(\alpha_0)$, $\tilde{\beta}_1, \tilde{\beta}_2, \ldots, \beta_r$ are loops in $\mathfrak{N}$, and words defined by $\tilde{\alpha}$ correspond (see (1)) to words defined by $\tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r$.

Evidently, the trains $(\tilde{\alpha}, \tilde{\beta}_1, \ldots, \tilde{\beta}_r)$ mentioned above are the shortest in the following sense: if $\tilde{\alpha}$ is a concatenation of two consecutive fragments $\tilde{\alpha}^1$ and $\tilde{\alpha}^2$, if every $\tilde{\beta}_i$, $1 \leq i \leq r$, is a concatenation of consecutive fragments $\tilde{\beta}_i^1, \tilde{\beta}_i^2$, and if $(\tilde{\alpha}^1, \tilde{\beta}_1^1, \tilde{\beta}_2^1, \ldots, \tilde{\beta}_r^1)$, $(\tilde{\alpha}^2, \tilde{\beta}_1^2, \tilde{\beta}_2^2, \ldots, \tilde{\beta}_r^2)$ (some $\tilde{\beta}_i^1$ may be equal) are of the above type, then the trains $(\tilde{\alpha}^1, \tilde{\beta}_1^1, \tilde{\beta}_2^1, \ldots, \tilde{\beta}_r^1)$ and (or) $(\tilde{\alpha}^2, \tilde{\beta}_1^2, \beta_2^2, \ldots, \tilde{\beta}_r^2)$, but not $(\tilde{\alpha}, \tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r)$ may specify a variable z. Here we use the fact that the vertex p''

nearest to p with the necessary properties is to be taken on the path from the root of R_n to p' (see the definition of PROC_2). Now it can be easily seen that the set of members cz used in the linear forms of (6) is finite. Hence, there is only a finite number of different trains $G_1^n(p')$ and $G_2^n(q')$, $n \in \mathbb{N}$.

Now, let γ be a sufficiently long path from the root in R. Let $p_1, p_2, \ldots, p_j, \ldots$ be the vertices on this path corresponding to the same vertex in $\mathfrak{A}$ (say, p). There are paths $\beta_1, \beta_2, \ldots, \beta_r, \ldots$ in T corresponding to γ such that, for each j, there is a vertex q_j^r in β_r glued together with p_j. For every j, the finite set $Q_j = \{q_j^1, q_j^1, \ldots, q_j^r, \ldots\}$ corresponds to a finite set $\bar{Q}_j$ in the graph $\mathfrak{R}$. If j is a sufficiently long path in R, then, obviously, i and j exist such that

(a) $\bar{Q}_i = \bar{Q}_j$;

(b) $G_1^{n_1}(p_i) = G_2^{n_2}(p_j)$, where n_1 and n_2 are such that p_i is a leaf in R_{n_1} and p_j is a leaf in R_{n_2};

(c) for every r,

$$G_2^{n_1}(q_j^r) = G_2^{n_2}(q_j^r).$$

In this case, the fragment of γ from p_i to p_j has to be removed. From these considerations it follows that all paths in R are of bounded length. Therefore, $R_n = R_j$ for some j and $n > j$. Thus we have obtained the necessary assertion. □

Proof of Theorem 4.1 (*continued*). Note that T is also a finite tree.

Now, we return to the construction of the procedure checking the inclusion $U(S) \subseteq L(\mathfrak{R})$.

Let p_0 and q_0 be the initial states in $\mathfrak{A}$ and $\mathfrak{R}$ respectively. Denote by p_1 the first vertex from p_0 on α_0 laying on the loop. Further, denote by p_2 the first vertex on the fragment of α_0 with beginning p_1 such that simple loops passing through p_2 differ from simple loops containing p_1. In a similar way, define $p_3, p_4, \ldots, p_k$ (we have p_1, p_2, p_4, p_8, p_9 in Fig. 3). Let p_{k+1} be the end of α_0.

Define $G_1(p_0) = (0, 0, \ldots, 0)$ (m_1 times) and $G_2(q_0) = (0, 0, \ldots, 0)$ (m_2 times). Perform $\mathrm{PROC}_1(p_0, p_1, q_0, G_1(p_0), G_2(q_0))$. Then the trees R^1, T^1 and the trains $G_1(p_1)$, $G_2(q_1)$ for leaves p_1 in the tree R^1 and the leaves q_1 in T^1 glued together with p_1 are obtained. Perform $\mathrm{PROC}_3(p_1, q, G_1(p_1), G_2(q))$ for every q in $\mathfrak{R}$ corresponding to leaves in T^1, and corresponding trains $G_2(q)$.

Now, the trees $R^2(p_1, q)$ and $T^2(p_1, q)$ and corresponding trains $G_1(p')$ and $G_2(q')$ for leaves p' of $R^2(p_1, q)$ and leaves q' of $T^2(p_1, q)$ glued together with p' are obtained. Identify the roots of all $R^2(p_1, q)$ with the leaf corresponding to p_1 in R. Identify the roots of all $T^2(p_1, q)$ with the leaves corresponding to q in T^1. Denote the obtained trees by R^2 and T^2.

Now, let, for an arbitrary $i < k$, R^{2i}, T^{2i}, and $G_1(p)$, $G_2(q)$ for leaves in R^{2i} and leaves q in T^{2i} glued with p be defined. Compute $\mathrm{PROC}_1(p_i, p_{i+1}, G_1(p), G_2(q))$ for every p in $\mathfrak{A}$ corresponding to a leaf p in R^{2i}, and for corresponding q in $\mathfrak{R}$. Then we obtain $R^{2i+1}(p_{i+1}, q)$, $T^{2i+1}(p_{i+1}, q)$, and $G_1(p'')$, $G_2(q')$ for leaves p' in $R^{2i+1}(p_{i+1}, q)$ and leaves q' in $T^{2i+1}(p_{i+1}, q)$ glued together with q'. Identify the

roots of all trees $R^{2i+1}(p_{i+1}, q)$ with the leaf in R^{2i} corresponding to p_{i+1}. Identify the roots of all $T^{2i+1}(p_{i+1}, q)$'s with the leaves q in T^{2i} glued together with p_{i+1}. Denote the obtained trees by R^{2i+1} and T^{2i+1}.

Perform now $\mathrm{PROC}_2(p_{i+1}, q, G_1(p_{i+1}), G_2(q))$ for all vertices q in $\mathfrak{N}$ corresponding to leaves in T^{2i+1} glued together with p_{i+1} in R^{2i+1}. Then trees $R^{2i+2}(p_{i+1}, q)$, $T^{2i+2}(p_{i+1}, q)$ and corresponding trains $G_1(p')$, $G_2(q')$ for the leaves p' in $R^{2i+2}(p_{i+1}, q)$ and the leaves q' in $T^{2i+2}(p_{i+1}, q)$ glued together with p' are obtained. For every leaf q in T^{2i+1}, identify the root of $T^{2i+2}(p_{i+1}, q)$ with q and the root of $R^{2i+2}(p_{i+1}, q)$ with the leaf in R^{2i+1} glued together with q. Denote the obtained trees by R^{2i+2} and T^{2i+2} respectively.

Define $R = R^{2k+3}$, $T = T^{2k+3}$.

Now, let $z_1, z_2, \ldots, z_e$ be the variables contained in the linear forms of $G_1(p)$, $G_2(q)$ for all vertices p in R and q in T. Let α be an arbitrary branch in R. We will associate a path $\alpha(k_1, k_2, \ldots, k_e)$ in $\mathfrak{A}$ with arbitrary $k_1, k_2, \ldots, k_e \in \mathbb{N}$ as follows.

Let z_j correspond to a loop C_j in $\mathfrak{A}$ (see the definition of PROC_2). Among all the loops c_j, $1 \leqslant j \leqslant e$, select $C^1_{j_1}, C^1_{j_2}, \ldots, C^1_{j_{n_1}}$ such that they are fragments of no other C_j. From the description of PROC_1 it follows that there are vertices corresponding to the beginning (and, therefore, to the final) vertices of some loops $C \in \bar{C}^1 = \{C^1_{j_1}, C^1_{j_2}, \ldots, C_{j n_1}\}$ in α. Insert into α the loop $C^1_{j_1}$ k_{j_1} times in corresponding vertices, the loop $C^1_{j_2}$ k_{j_2} times, and so on; if the beginning vertex of $C^1_{j_i}$ is in α more than once, then insert $C^1_{j_1}$ so that the common number of its copies in α is equal to k_{j_1}. So a path α^1 in $\mathfrak{A}$ is obtained. Now, select the set

$$\bar{C}^2 = \{C^2_{j_1}, C^2_{j_2}, \ldots, C^2_{j_{n_2}}\} \subseteq \{C_j\}_{1 \leqslant j \leqslant e} \setminus \bar{C}_1$$

of the loops C_j that are fragments of no other loop in $\{C_j\}_{1 \leqslant j \leqslant e} \setminus \bar{C}_1$. In a similar way, insert every $C^2_{j_i} \in \bar{C}^2$ into α^1 k_{j_i} times in corresponding places. Denote the path obtained above by α^2. Similarly, define $\alpha^3, \alpha^4, \ldots, \alpha^m, \ldots$. Finally, some path α^m in $\mathfrak{A}$ will be obtained. Denote this path by $\alpha(k_1, k_2, \ldots, k_e)$.

Clearly, in general, $\alpha(k_1, k_2, \ldots, k_e)$ may be not uniquely defined. Hence, we have to regard the set $[\alpha(k_1, k_2, \ldots, k_e)]$ of all the possible paths $\alpha(k_1, k_2, \ldots, k_e)$ that may be obtained by the above procedure.

In a similar manner, one can associate the set of paths $[\beta(k_1, k_2, \ldots, k_e)]$ in $\mathfrak{N}$ with every branch β in T. Note only that in this case, a whole set $\{C_1, C_2, \ldots, C_s\}$ of loops in $\mathfrak{N}$ with a common beginning (and final) vertex corresponds to every variable z_j (recall that every variable z is associated with a train $(\tilde{\alpha}, \tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r)$ of paths $\tilde{\alpha}$ in $\mathfrak{A}$, and $\tilde{\beta}_1, \tilde{\beta}_2, \ldots, \tilde{\beta}_r$ in $\mathfrak{N}$), and the common number of copies of loops $C_1, C_2, \ldots, C_s$ to be inserted into $\beta(k_1, k_2, \ldots, k_e)$ is equal to k_j.

Note now that if the ends of paths α in R and β in T are glued together, then paths $\alpha(k_1, k_2, \ldots, k_e)$ and $\beta(k_1, k_2, \ldots, k_e)$ may be chosen *consistently* in the following sense: insertions of a loop C' into α and loops $C_1, C_2, \ldots, C_s$ into β associated with one variable z are made in vertices glued together during the process of constructing R and T.

Let us return now to the test for checking $U(S) \subseteq L(\mathfrak{N})$,

Assume that for some $\bar{a} \in S$, $u(\bar{\alpha}) \notin L(\mathfrak{N})$. The procedure of constructing trees R and T associates with $\bar{\alpha}$ some path α in R and some paths $\beta_1, \beta_2, \ldots, \beta_r$ in T such that their vertices are consecutively glued together with vertices on α. It is not difficult to see that $\bar{\alpha} \in [\alpha(k_1, k_2, \ldots, k_e)]$ for some $k_1, k_2, \ldots, k_e \in \mathbb{N}$.

Now, let G_1 be the train of m_1 forms $0 \in \mathrm{LF}$, and $G_2 = (0, 0, \ldots, 0)$ (m_2 times). Let

$$\bar{b}(\alpha, G_1) = a_1 a_2 \ldots a_{r(\alpha)},$$

and, for every path β in T,

$$\bar{b}(\beta, G_2) = b_1(\beta) b_2(\beta) \ldots b_{r(\beta)}(\beta).$$

Let $z_1, z_2, \ldots, z_e$ be the variables contained in all forms $g \in G_1(p) \cup G_2(q)$ for all vertices p in R and q in T.

Now we are going to define a procedure that, given an α (and, therefore, $\bar{\alpha}$), effectively constructs a path

$$\alpha(k_1, k_2, \ldots, k_e) \in [\alpha(k_1, k_2, \ldots, k_e)]$$

such that, for every β in T and $\beta(k_1, k_2, \ldots, k_e)$ in $\mathfrak{N}$ consistent with $\alpha(k_1, k_2, \ldots, k_e)$

$$\alpha(k_1, k_2, \ldots, k_e) \neq \beta(k_1, k_2, \ldots, k_e).$$

The procedure searches for paths β in T such that $\bar{b}(\alpha, G_1) = \bar{b}(\beta, G_2)$. Let a path β in T be found such that the length of $\bar{b}(\beta, G_2)$ is equal to r and, for all $j \leq r$, $b_j(\beta) = a_j$. If there are $\gamma_1, \gamma_2, \ldots, \gamma_k$ in T with $\bar{b}(\gamma_i, G_2)$ of length $r+1$ that continue β and have $b_{r+1}(\gamma) = a_{r+1}$, then continue the procedure using the paths $\gamma_1, \gamma_2, \ldots, \gamma_k$ instead of β. If there is no such γ in T, then, for each γ in T, the two following cases are, evidently, possible:

(1) $b_{r+1}(\gamma)$ is an A-letter;

(2) $b_{r+1}(\gamma) \in \mathrm{LF}$.

For γ with $b_{r+1}(\gamma) \in A$ we have, evidently,

$$u(\alpha(k_1, k_2, \ldots, k_e)) \neq u(\beta(k_1, k_2, \ldots, k_e))$$

for all $k_1, k_2, \ldots, k_e$ and all consistent $\alpha(k_1, k_2, \ldots, k_e)$ and $\beta(k_1, k_2, \ldots, k_e)$. For all paths γ with $b_{r+1}(\gamma) \in \mathrm{LF}$ we use the following evident assertions.

Assertion 4.3. *Let $f_0, f_1, \ldots, f_p \in \mathrm{LF}$, $z_1, z_2, \ldots, z_e$ be the variables contained in one of $f \in \{f_0, f_1, \ldots, f_p\}$ at least. Let $f_j(\bar{\alpha})$ be the value of f_j on an e-tuple $\bar{k} = (k_1, k_2, \ldots, k_e)$, $k_j \in \mathbb{N}$, $1 \leq j \leq e$. Let f_0 be equal to no $f_1, f_2, \ldots, f_p$. Then a $\bar{k}$ can be effectively found such that*

$$(\forall j \leq p)\,(f_0(\bar{k}) \neq f_j(\bar{k})).$$

Proof of Theorem 4.1 (*continued*). Using Assertion 4.3 one can effectively find an e-tuple $\bar{k}$ such that, for every γ that continues β, the value of a_{r+1} on $\bar{k}$ is not equal to the value of $b_{r+1}(\gamma)$ on $\bar{k}$. Clearly, then $u(\alpha(k_1, \ldots, k_e)) \neq u(\gamma(k_1, \ldots, k_e))$ also holds for all consistent $\alpha(k_1, \ldots, k_e)$ and $\gamma(k_1, \ldots, k_e)$.

Assume now that, for branch α in R, all paths $\gamma_1, \gamma_2, \ldots, \gamma_m$ in T are found such that their ends $q_1, q_2, \ldots, q_m$ are glued together with the end p of α, and

$$\bar{b}(\alpha, G_1) = \bar{b}(\gamma_j, G_2), \quad 1 \leqslant j \leqslant m.$$

Clearly, $u(\alpha(k_1, \ldots, k_e)) = u(\gamma_j(k_1, \ldots, k_e)$ holds for every j, $1 \leqslant j \leqslant m$, for every e-tuple $(k_1, \ldots, k_e)$ of values of the variables $z_1, z_2, \ldots, z_e$ contained in the forms in

$$\bar{b}(\alpha, G_1), \qquad \bar{b}(\gamma_j, G_2), \quad 1 \leqslant j \leqslant m,$$

and for every consistent path $\alpha(k_1, \ldots, k_e)$ in $\mathfrak{A}$ and $\gamma_j(k_1, \ldots, k_e)$ in $\mathfrak{N}$. On the other hand, there is no δ in T such that

$$u(\alpha(k_1, \ldots, k_e)) = u(\delta(k_1, \ldots, k_e))$$

holds for any $k_1, \ldots, k_e$. However, the path α in R is taken so that, for some numbers $k_1, \ldots, k_e$ and for some $\alpha(k_1, \ldots, k_e)$,

$$\bar{\alpha} = \alpha(k_1, \ldots, k_e).$$

From this reasoning it follows that $q_1, q_2, \ldots, q_m$ correspond to nonaccepting states of $\mathfrak{N}$.

Consider now some path $\alpha(\bar{1})$ in $\mathfrak{A}$, where $\bar{1} = (1, 1, \ldots, 1)$ (e times). Clearly, $u(\alpha(\bar{1})) \in U(S)$. On the other hand, $u(\gamma_j(\bar{1})) \notin L(\mathfrak{N})$ for each $j \in \{1, 2, \ldots, m\}$ and for each $\gamma_j(\bar{1})$ consistent with $\alpha(\bar{1})$. However, $u(\gamma) = u(\alpha(\bar{1}))$ holds for no γ in $\mathfrak{N}$ differing from $\gamma_j(\bar{1})$, $1 \leqslant j \leqslant m$. Therefore, if $u(\bar{\alpha}) \in U(S) \setminus L(\mathfrak{N})$, then $u(\alpha(\bar{1})) \in U(S) \setminus L(\mathfrak{N})$.

The procedure constructed above shows that to check the inclusion $U(S) \subseteq L(\mathfrak{N})$, it is sufficient to check whether $u(\alpha(k_1, \ldots, k_e)) \in L(\mathfrak{N})$

(1) for all branches α in R;

(2) for all e-tuples $(k_1, \ldots, k_e)$ of values of the variables contained in the linear forms in $\bar{b}(\alpha, G_1)$, $\bar{b}(\gamma, G_2)$, $\gamma \in T$, where all $k_i \leqslant \tilde{k}$ for some constant $\tilde{k}$ effectively defined by R, T, $G_1(p)$, $G_2(q)$; and finally,

(3) for all possible $\alpha(k_1, \ldots, k_e) \in [\alpha(k_1, \ldots, k_e)]$.

So the inclusion $U(S) \subseteq L(\mathfrak{N})$ can be effectively checked. The theorem is proved. □

Corollary 4.4. *The emptiness and equivalence problems are decidable for languages in* $\mathscr{L}(\mathscr{A})$.

5. Intersection and difference of the languages in $\mathscr{L}(\mathscr{A})$

Theorem 5.1. *The class* $\mathscr{L}(\mathscr{A})$ *is closed under* $\cap$ and $\setminus$.

The proof is based on a procedure similar to the one in the proof of Theorem 4.1, constructing trees R, T, and trains $G_1(p)$, $G_2(q)$. The reader can easily make the necessary modifications himself.

6. The problem of synthesis

The task of a synthesis is to trace the correct program $P \in \mathscr{P}$ using a finite set of examples from $V(P)$. Obviously, this problem is solvable by an exhaustive search.

Unfortunately, algorithms of exhaustive search are clearly unpractical. In [1, 2, 3] polynomial algorithms synthesizing programs by one sufficiently long example are presented. In our case, because of the possible branching in sample computations, it is clearly impossible. On the other hand, the operations: * and +, which are in fact a tool for representing regularities like arithmetical progressions, allow us to hope that an effective synthesis algorithm for programs $p \in \mathscr{P}$ by one example is possible if some additional information is given. One possibility is to synthesize programs from one long example if approximate positions of the boundaries of the loops and the relations between the value sof the same variable in the sample computation are given. The authors hope to investigate this problem in a forthcoming paper.

Acknowledgment

The authors are very grateful to the unknown referee for many useful suggestions.

References

[1] J.M. Barzdin, Some rules of inductive inference and their use for program synthesis, *Information Processing 83* (North-Holland, Amsterdam, 1983) 333–338.

[2] A.N. Brazma, Inductive inference of programs containing while loops, *Elektron. Informationsverarb. Kybernetik*, to appear.

[3] J.M. Barzdin, On inductive synthesis of programs, Lecture Notes in Computer Science **122** (Springer, Berlin, 1981) 234–254.

[4] Y. Kadratoff, A class of functions synthesized from a finite number of examples and LISP program scheme, *Internat. J. Comput. Inform. Sci.* **8**(6) (1979) 489–521.

[5] A.W. Bierman and R. Krischnaswamy, Constructing programs from example computations, *IEEE Trans. Software Engrg.* **SE-2**(2) (1976) 141–153.

[6] P.D. Summers, A methodology for LISP program construction from examples, *J. ACM* **24**(2) (1977) 161–175.

[7] B.A. Trakhtenbrot and J. M. Barzdin, *Finite Automata* (North-Holland, Amsterdam, 1973).