# Полиномиальный алгоритм построения оптимального инверсного морфизма

Б. Ф. Мельников

*Аннотация*—**В настоящей статье мы продолжаем тематику нескольких наших предыдущих работ. В этой тематике мы рассматриваем две схожие задачи теории формальных языков: задачу извлечения корня заданной степени и задачу построения оптимального инверсного морфизма, где оптимальность можно определить как длину максимального слова языка, являющегося инверсным морфическим образом. Вместо рассмотрения всех подмножеств множества т. н. потенциальных корней, что всегда приводит к экспоненциальным алгоримам, мы в этих публикациях получали алгоритмы полиномиальные.**

**Для того, чтобы сформулировать вторую из этих задач удобным для построения полиномиальных алгоритмов образом, мы в предыдущих статьях рассмотрели несколько эквивалентных вариантов специальной гипотезы теории формальных языков. Одна из нескольких её эквивалентных формулировок может быть выражена следующим образом. Для двух конечных и не содержащих пустого слова языков мы можем записать необходимое и достаточное условие того, что итерация любого из этих языков принадлежит множеству префиксов другого языка, – причём записать его следующим образом. Существует некоторый новый алфавит, отличный от алфавита, над которым заданы исходные языки, а над этим алфавитом – два максимальных префиксных кода (вообще говоря, различных) и два языка, содержащих эти максимальные префиксные коды как подмножества. Кроме того, существует некоторый специальный морфизм, и исходные языки образуются путём применения этого морфизма к языкам, содержащих эти максимальные префиксные коды как подмножества.**

**В настоящей статье мы показываем, что при выполнении этой гипотезы возможен полиномиальный алгоритм построения оптимального инверсного морфизма. Причём, немного упрощая ситуацию, можно сказать, что такой алгоритм является специальным объединением нескольких полиномиальных же алгоритмов, рассмотренных в наших предыдущих статьях, – или несложных модификаций таких алгоритмов.**

*Ключевые слова*—**формальные языки, итерации языков, морфизмы, инверсные морфизмы, бинарные отношения, алгоритмы.**

## I. ВВЕДЕНИЕ

В настоящей статье мы продолжаем тематику наших предыдущих работ [1], [2], [3] (в первую очередь), а также [4], [5], [6], [7], [8], [9], [10]; сто́ит также отметить, что связанные вопросы приведены в [11], [12], [13]. В этой тематике мы рассматриваем две схожие задачи теории формальных языков:

- задачу (задачи) извлечения корня заданной степени
- и задачу (задачи) построения оптимального инверсного морфизма – где оптимальность можно определить, например, как длину максимального слова языка, являющегося инверсным морфическим образом.

И вместо рассмотрения всех подмножеств множества потенциальных корней (последние впервые определены нами в [1]), что всегда приводит к экспоненциальным алгоримам, мы, применяя получающиеся в некоторых ситуациях полурешётки таких подмножеств, получаем алгоритмы полиномиальные. Специально отметим, что в статьях [8], [9], [10] слово «полурешётка» вынесено в название.

К последнему добавим, что практически в каждой из процитированных работ есть раздел *мотивация* – и приведённые в этих разделах аргументы, конечно же, можно считать и мотивацией к описанию полиномиального алгоритма, рассматриваемого в настоящей статье.

Теперь немного подробнее про сами задачи. Для заданного языка $A \subseteq \Sigma^*$:

- извлечь корень $\mathfrak{n}$-й степени – это найти такой язык (такие языки) $B \subseteq \Sigma^*$, что
$$A = B^{\mathfrak{n}};$$
- построить оптимальный инверсный морфизм – это найти такой язык (такие языки) $D \subseteq \Delta^*$ (здесь $\Delta$ – некоторый новый алфавит) и морфизм
$$\mathfrak{h} : \Delta \to \Sigma^*,$$
что:
  - $D$ в качестве подмножества (возможно, несобственного) содержит некоторый максимальный префиксный код (над этим алфавитом $\Delta$),
  - и, кроме того, $A = \mathfrak{h}(D)$.

Мы рассматриваем не две задачи, а *две группы задач*, поскольку в обоих случаях имеется несколько вариантов для требуемого в задаче ответа:

- найти *любой* (корень из языка либо инверсный морфизм);
- найти *все*;
- найти *минимальный* (по некоторой норме);
- и т. п.

Однако несложно понять, что все эти варианты очень близки между собой.

Для того, чтобы сформулировать вторую из этих групп задач именно таким образом, в упомянутых статьях [8], [9], [10], а также в [14], мы рассмотрели несколько эквивалентных вариантов специальной гипотезы теории формальных языков, названной нами гипотезой (Ѫ). Одна из нескольких её эквивалентных формулировок, формулируемая согласно [14], может быть выражена следующим образом. Для двух конечных и не содержащих пустого слова $\varepsilon$ языков ($A$ и $B$) мы можем записать необходимое

Статья получена 28 марта 2023 г.

Борис Феликсович Мельников, Университет МГУ – ППИ в Шэньчжэне (bormel@smbu.edu.cn).

и достаточное условие того, что итерация любого из этих языков принадлежит множеству префиксов другого языка, – причём записать его следующим образом. Существует некоторый алфавит $\Delta$ (отличный от алфавита $\Sigma$, над которым заданы $A$ и $B$), над ним – два максимальных префиксных кода (вообще говоря, различных) и два языка ($A_\Delta$ и $B_\Delta$), содержащих эти максимальные префиксные коды как подмножества, а также некоторый морфизм $h : \Delta^* \to \Sigma^*$. При этом исходные языки $A$ и $B$ образуются путём применения этого морфизма к языкам $A_\Delta$ и $B_\Delta$.

В настоящей статье мы показываем, что при выполнении этой гипотезы возможен *полиномиальный* алгоритм построения оптимального инверсного морфизма. Причём, немного упрощая ситуацию, можно сказать, что такой алгоритм является специальным объединением нескольких полиномиальных же алгоритмов, рассмотренных в наших предыдущих статьях, – или несложных модификаций таких алгоритмов.

Приведём содержание статьи по разделам. *В разделе II* мы, в основном, повторяем термины, введённые нами ранее: без рассмотрения этих терминов настоящую статью, по-видимому, понимать сложнее. Наиболее важный из этих терминов – т. н. потенциальные корни, это просто возможные корни (любой степени) слов заданного языка. Как уже было отмечено выше, для всех рассматриваемых нами задач легко описываются экспоненциальные алгоритмы, заключающиеся в переборе подмножеств множества этих потенциальных корней.

В двух следующих разделах приведены описания вспомогательных алгоритмов: два алгоритма раздела *III* очень простые, а алгоритм *раздела IV* – небольшая модификация алгоритма, ранее рассмотренного нами в [2].

*В разделе V* описан главный алгоритм настоящей статьи; его основная часть заключается в итерационном выполнении рассматриваемых вспомогательных алгоритмов, и, на их основе, в переводе потенциальных корней – конкатенаций из одной рассматриваемой переменной-языка в другую переменую. При этом за каждый шаг такой итерации число слов рассматриваемого языка уменьшается – и из этого факта и полиномиальности каждого из «вспомогательных» алгоритмов следует полиномиальность всего алгоритма.

На основе описания главного алгоритма статьи *в разделе VI* приведён несложный пример его применения. Он «перекликается» с примерами, ранее рассматривавшимися в некоторых наших предыдущих статьях. *Раздел VII* – заключение, в нём приведено основное возможное направление дальнейших работ по рассмотренной в статье тематике.

## II. ПРЕДВАРИТЕЛЬНЫЕ СВЕ́ДЕНИЯ

В этом разделе мы, в основном, повторяем термины, введённые в предыдущих статьях по этой тематике: без рассмотрения этих терминов настоящую статью, по-видимому, понимать сложнее.

Начнём с максимальных префиксных кодов и морфизмов. Множество максимальных префиксных кодов над алфавитом $\Delta$ ([15, стр. 135, 144]) будем обозначать записью $\mathrm{mp}(\Delta)$. Возможный несложный *недетерминированный* алгоритм построения *любого* такого максимального префиксного кода – т. е. любого элемента множества $\mathrm{mp}(\Delta)$ – следующий.

**Недетерминированный алгоритм `All-MP`:** ***построение всех максимальных префиксных кодов.***

- (База.) Максимальным префиксным кодом является сам язык $\Delta$.
  Замечание 1. При этом мы рассматриваем буквы алфавита в качестве 1-буквенных слов.
  Замечание 2. В некоторых случаях удобнее в качестве базы рассматривать язык $\{e\}$ – но в нашей ситуации это делать нежелательно.
- (Шаг.) Если известно, что $A_\Delta \in \mathrm{mp}(\Delta)$, и при этом $u_\Delta \in A_\Delta$, то считаем, что

$$(A_\Delta \setminus \{u_\Delta\}) \cup \{u_\Delta\} \cdot \Delta \ \in \ \mathrm{mp}(\Delta).$$

*Конец формулировки алгоритма.*

Множество конечных языков над алфавитом $\Delta$, каждый из которых в качестве подмножества (возможно, несобственного) содержит некоторый максимальный префиксный код над $\Delta$ (т. е. какой-либо элемент множества $\mathrm{mp}(\Delta)$), будем обозначать $\mathrm{mp}^+(\Delta)$ – т. е. формально

$$\mathrm{mp}^+(\Delta) = \left\{ A_\Delta \subseteq \Sigma^* \,\middle|\, (\exists B_\Delta \subseteq A_\Delta)\,(B_\Delta \in \mathrm{mp}(\Delta)) \right\}.$$

Будем любой из языков множества $\mathrm{mp}^+(\Delta)$ называть расширенным максимальным префиксным кодом.

Морфизм (использование этого термина согласовано с [16]) – это отображение

$$h : \Delta^* \to \Sigma^*,$$

для которого:

- для каждой буквы $d \in \Delta$ её образ $h(d) \in \Sigma^*$ задаётся;
- а для каждого слова $d_1 d_2 \dots d_n \in \Delta^*$ полагаем

$$h(d_1 d_2 \dots d_n) = h(d_1)\, h(d_2) \dots h(d_n).$$

Нам будут очень важны морфизмы, *соответствующие конечным языкам*. То есть для некоторого языка

$$A \in \Sigma^*, \quad A = \{u_1, u_2, \dots, u_n\}$$

мы рассматриваем алфавит

$$\Delta_A = \{d_1, d_2, \dots, d_n\}$$

(если это не вызывает неоднозначностей, пишем обычно просто $\Delta$), а для него – морфизм

$$h_A : \Delta_A^* \to \Sigma^*,$$

задающийся следующим образом:

$$h_A(d_1) = u_1\,,\ h_A(d_2) = u_2\,,\ \dots,\ h_A(d_n) = u_n\,.$$

Будем называть такой морфизм $A$-морфизмом, не конкретизируя при этом порядок слов языка $A$.

Для рассматриваемых алфавита $\Delta$ и языка $A \subseteq \Sigma^*$ следующее множество языков над алфавитом $\Sigma$ будем обозначать $\mathrm{mp}(A)$:

$$\mathrm{mp}(A) = \left\{ B \subseteq \Sigma^* \,\middle|\, (\exists A_\Delta \in \mathrm{mp}(\Delta))\,(B = h_A(A_\Delta)) \right\}.$$

Аналогично,

$$\mathrm{mp}^+(A) = \left\{ B \subseteq \Sigma^* \,\middle|\, (\exists A_\Delta \in \mathrm{mp}^+(\Delta))\,(B = h_A(A_\Delta)) \right\}.$$

Заметим, что в обеих последнних формулах некоторое конкретное слово языка B записью $B = h_A(A_\Delta)$ определяется, вообще говоря, неоднозначно.

Перейдём к определению множества потенциальных корней языка конкретной степени $n \in \mathbb{N}$. Это множество обозначается

$$\overline{\sqrt[n]{A}} = \left\{ u \in \Sigma^* \,\middle|\, u^n \in A \right\}. \qquad (1)$$

При этом для рассматриваемой в настоящей статье задачи построения оптимального инверсного морфизма в равенстве (1) $n$ может быть любым подходящим натуральным числом (а не каким-то заранее заданным), поэтому мы обычно пишем

$$\overline{\sqrt[*]{A}} = \left\{ u \in \Sigma^* \,\middle|\, (\exists k \in \mathbb{N})\,(u^k \in A) \right\};$$

специально отметим, что равенство $k = 1$ допускается. Также заанее отметим, что построение множества всех потенциальных корней может быть выполнено с помощью несложного полиномиального алгоритма – о нём будет сказано далее в тексте статьи.

Теперь рассмотрим приведённые в предыдущих статьях определения, связанные с бинарным отношением $\lessdot\!\!\cdot\!\!\gtrdot$. Если для конечных языков A и B выполнено условие

$$(\forall u \in A^*)\ (\exists v \in B^*)\ (u \in \mathrm{opref}(v)),$$

то будем писать $A \lessdot B$ (либо $B \gtrdot A$). Если одновременно выполнены условия $A \lessdot B$ и $A \gtrdot B$, то будем писать $A \lessdot\!\!\cdot\!\!\gtrdot B$. Наоборот, в случае, когда уже известно, что условие $A \gtrdot B$ не выполнено, мы в случае выполнения условия $A \lessdot B$ можем также писать $A \lhd B$ (либо $B \rhd A$).

К этим обозначениям приведём такие замечания.

- Во-первых, обозначение $A \lhd B$ в наших предыдущих работах определялось двумя разными способами – однако здесь мы не будем ни приводить второй возможный способ, доказывать эквивалентность обоих.
- Во-вторых, можно очевидным образом объяснить и обозначение $A \lhd\!\cdot\!\rhd B$ (соответствующее ему условие выполнено, например, для пары $A = \{a\}$, $B = \{b\}$). В одной из наших предыдущих статей по поводу этого обозначения была фраза «однако вряд ли это обозначение будет когда-либо востребовано»; мы его не будем использовать и в настоящей статье – однако легко понять, что его всё же можно здесь применить, например – для комментариев к основному алгоритму.

В настоящей работе к этим обозначениям (рассматривавшимся нами и ранее) добавим два следующих. будем писать $B \Rsh A$ (или, соответственно, $B \nRsh A$) в случаях, когда:

- во-первых, $B \gtrdot A$ (или, соответственно, $B \rhd A$);
- во-вторых, вдобавок к предыдущему, $A \in \mathrm{mp}^+(B)$.

Конкретное определение возможного частичного порядка на множестве языков может быть дано несколькими способами (см. [9] и др.) – но оно будет применяться в настоящей статье, и поэтому для строгости изложения определим один конкретный возможный вариант. Более того, аналогично [9] оформим это определение в виде алгоритма.

**Определение-алгоритм `Part-Order`:** ***возможный частичный порядок на множестве языков.***

**1.** Рассматриваем произвольный непустой конечный язык; пусть это

$$A = \{u_1, u_2, \ldots, u_n\},$$

где

$$|u_1| \leqslant |u_2| \leqslant \cdots \leqslant |u_n|.$$

Последовательность

$$(|u_n|, |u_{n-1}|, \ldots, |u_2|, |u_1|)$$

будем обозначать записью $\mathrm{Le}(A)$ [1]; при этом для непустых последовательностей считаем

$$\mathrm{He}(A) = |u_n|,$$
$$\mathrm{Ta}(A) = \mathrm{Le}(A \setminus \{u_n\}) = (|u_{n-1}|, \ldots, |u_2|, |u_1|).$$

В приведённых записях последовательность $\mathrm{Ta}(A)$ может быть пустой, обозначаем таковую $()$.

**2.** Для двух конечных языков A и B:

- если $A = B = ()$, полагаем $A \simeq B$ [2];
- иначе если $A = ()$, полагаем $A \prec B$;
- иначе если $B = ()$, полагаем $B \prec A$;
- иначе если $\mathrm{He}(A) < \mathrm{He}(B)$, полагаем $A \prec B$;
- иначе ответ ($A \simeq B$, либо $A \prec B$, либо $A \succ B$) выбираем так же, как для последовательностей $\mathrm{Ta}(A)$ и $\mathrm{Ta}(B)$ соответственно.

Очевидно, что приведённый алгоритм зацикливаться не может. Поэтому на множестве языков определяется частичный порядок – [17], [18], [19]. Обозначения $\preccurlyeq$ и $\succcurlyeq$ при необходимости также употребляем обычным образом: ($\prec$ либо $\simeq$) и ($\succ$ либо $\simeq$) соответственно.

*Конец формулировки определения.*

Также мы для языка $B \subseteq \Sigma^*$ и слова $u \in \Sigma^*$ (обычно $u \in B$) будем употреблять следующее обозначение:

$$B_{-u} = B \setminus \{u\}.$$

В конце раздела приведём одну из нескольких эквивалентных формулировок вышеупомянутой гипотезы (Ѫ) – [14] и др.

**Гипотеза Ѫ .** Для любых конечных языков $A, B \subseteq \Sigma^*$ выполнено следующее. Эквивалентность $A \lessdot\!\!\cdot\!\!\gtrdot B$ выполняется тогда и только тогда, когда существует язык $\mathcal{D} \subseteq \Sigma^*$, такой что при рассмотрении соответствующего алфавита $\Delta$ (где $|\Delta| = |\mathcal{D}|$) существуют расширенные максимальные префиксные коды

$$A_\Delta, B_\Delta \subseteq \Delta^*,$$

такие что

$$A = h_{\mathcal{D}}(A_\Delta), \quad B = h_{\mathcal{D}}(B_\Delta).$$

Ниже такой язык $\mathcal{D}$ будем иногда называть *минимальным* языком (для заданного языка A, либо эквивалентного ему языка B). □

[1] Le от “length”, He от “head”, Ta от “tail”.

[2] Можно «упростить» запись, написав здесь «если $A = B$ …». Всё определение при этом будет эквивалентно приведённому.

## III. ОПИСАНИЯ ДВУХ ПРОСТЫХ ВСПОМОГАТЕЛЬНЫХ АЛГОРИТМОВ

Для обоих алгоритмов, приведённых в этом разделе, очевидны как состоятельность, так и полиномиальность. Однако мы оформим их строго, а также приведём доказательства полиномиальности [3].

**Первый вспомогательный алгоритм `Pot-Roots`: *определение потенциальных корней.***

*Вход:* непустой конечный язык $A \subseteq \Sigma^*$, причём

$$A = \{u_1, u_2, \ldots, u_n\},$$

а для каждого $i \in \overline{1,n}$

$$u_i = a_{i,1}a_{i,2}\ldots a_{i,l_i}.$$

*Выход:* язык $B \subseteq \Sigma^*$, такой что

$$B = \overline{\sqrt[*]{A}}$$

(фактически $B \supseteq A$), состоящий из всех потенциальных корней языка $A$, т. е. из всех таких $v \in \Sigma^*$, что

$$\left(\exists k \in \mathbb{N}\right)\left(v^k \in A\right).$$

*Метод.* Для каждого $i \in \overline{1,n}$ рассматриваем все такие $l \in \mathbb{N}$, что $l_i \vdots l$. Для каждого такого $l$ проверяем выполнение условия

$$\left(a_{i,1}a_{i,2}\ldots a_{i,l}\right)^{l_i/l} = u_i\,,$$

и при положительном ответе включаем слово

$$v = a_{i,1}a_{i,2}\ldots a_{i,l}$$

в строимое множество потенциальных корней $B$.

*Конец описания алгоритма.*

**Утверждение 1:** Описанный алгоритм `Pot-Roots` выполняется за полиномиальное время.

*Доказательство.* Каждая проверка каждого потенциального корня выполняется за 1 просмотр всего рассматриваемого слова $u_i$, а общее число потенциальных корней не превышает длину этого слова. Поэтому алгоритм для каждого из $u_i$ квадратичный, т. е. и описанный алгоритм `Pot-Roots` полностью выполняется за полиномиальное время. □

**Второй вспомогательный алгоритм `Del-Concat`: *удаление потенциальных корней, являющихся конкатенацией нескольких других потенциальных корней.***

*Вход:* непустой конечный язык $B \subseteq \Sigma^*$, являющийся множеством потенциальных корней (для некоторого «исходного» языка, обработанного алгоритмом `Pot-Roots`).

*Выход:* язык $C \subseteq B$, такой что

$$C = \left\{u \in B \;\middle|\; (\forall v_1, v_2, \ldots, v_k \in B,\ k \geqslant 2)\ (u \neq v_1v_2\ldots v_k)\right\}.$$

*Замечание 1.* Для того, чтобы легко было доказать полиномиальность описываемого алгоритма `Del-Concat`, мы применяем тот же простейший приём, который несколько раз использовался в [2], [20]: в качестве «главного действия» мы рассматриваем не варианты выбора конкретного «короткого» потенциального корня, а движение по буквам «длинного» потенциального корня. Кроме того, в связи с тем, что этот приём уже был применён несколько раз, мы вместо подробного формального описания алгоритма приведём только его схему – включающую в себя всё необходимое для строгого доказательства полиномиальности – если в нём возникнет необходимость.

*Замечание 2.* Не ограничивая общности будем считать, что алгоритм разбивает все потенциальные корни на два подмножества:

- корни, представимые в виде конкатенации двух и более других потенциальных корней;
- корни, которые не могут быть представлены в таком виде.

*Схема метода.* Рассматриваем очередное слово $u_i$ исходного языка $B$ – при этом сразу отмечая, что общую суммарную длину всех слов этого языка можно считать равной размерности задачи, т. е. значения, на основе которого и определяется сложность алгоритма. Для этого слова заводим равный его длине (плюс 1) булевский массив первоначально заполнив его всеми `false` – только 0-й элемент полагаем равным `true`. По смыслу этот массив отвечает на вопрос, представим ли префикс рассматриваемого слова нужной длины в виде конкатенации слов исходного языка $B$.

Это была «база» алгоритма; а его «шагом» является рассмотрение очередного префикса слова $u_i$ (очередного элемента массива) и заполнение значениями `true` новых элементов – тех, для которых удаётся выбрать подходящие слова из $B$. Поэтому если в результате работы алгоритма значение его последнего элемента окажется равным `true` – то исключаем из $B$ рассматриваемое слово $u_i$.

Заметим ещё, что выполняемое в процессе работы алгоритма уменьшение числа элементов языка $B$ на результат работы алгоритма не влияет.

*Конец описания алгоритма.*

**Утверждение 2:** Описанный алгоритм `Del-Concat` выполняется за полиномиальное время.

*Доказательство.* К изложенному выше обоснованию полиномиальности добавим следующее. Мы выбираем слово (которое пытаемся разложить), потом двигаемся по его буквам и отмечаем, где есть возможные концы других слов языка $B$; на каждой букве (а их не более $N$, где $N$ – размерность задачи, т. е. сумма длин всех слов рассматриваемого языка) мы делаем не более $N$ «дополнительных» шагов для формирования таких отметок концов слов. Таким образом, обработка одного слова занимает не более $N^2$ подобных операций, а поскольку общее количество слов в языке также не превышает $N$, то общая работа алгоритма `Del-Concat` не превышает $N^3$ операций. □

## IV. ПРИМЕНЯЕМАЯ МОДИФИКАЦИЯ ГЛАВНОГО ВСПОМОГАТЕЛЬНОГО АЛГОРИТМА

Как уже отмечалось во введении, мы совсем немного модифицируем алгоритм, приведённый в [2] – см. блок-схему на рис. 1.

[3] Отметим, что в предыдущих публикациях мы нередко приводили фрагменты компьютерных программ, подтверждающих полиномиальность. Для второго из алгоритмов этого раздела некоторые опубликованные ранее фрагменты компьютерных программ подходят без изменений.

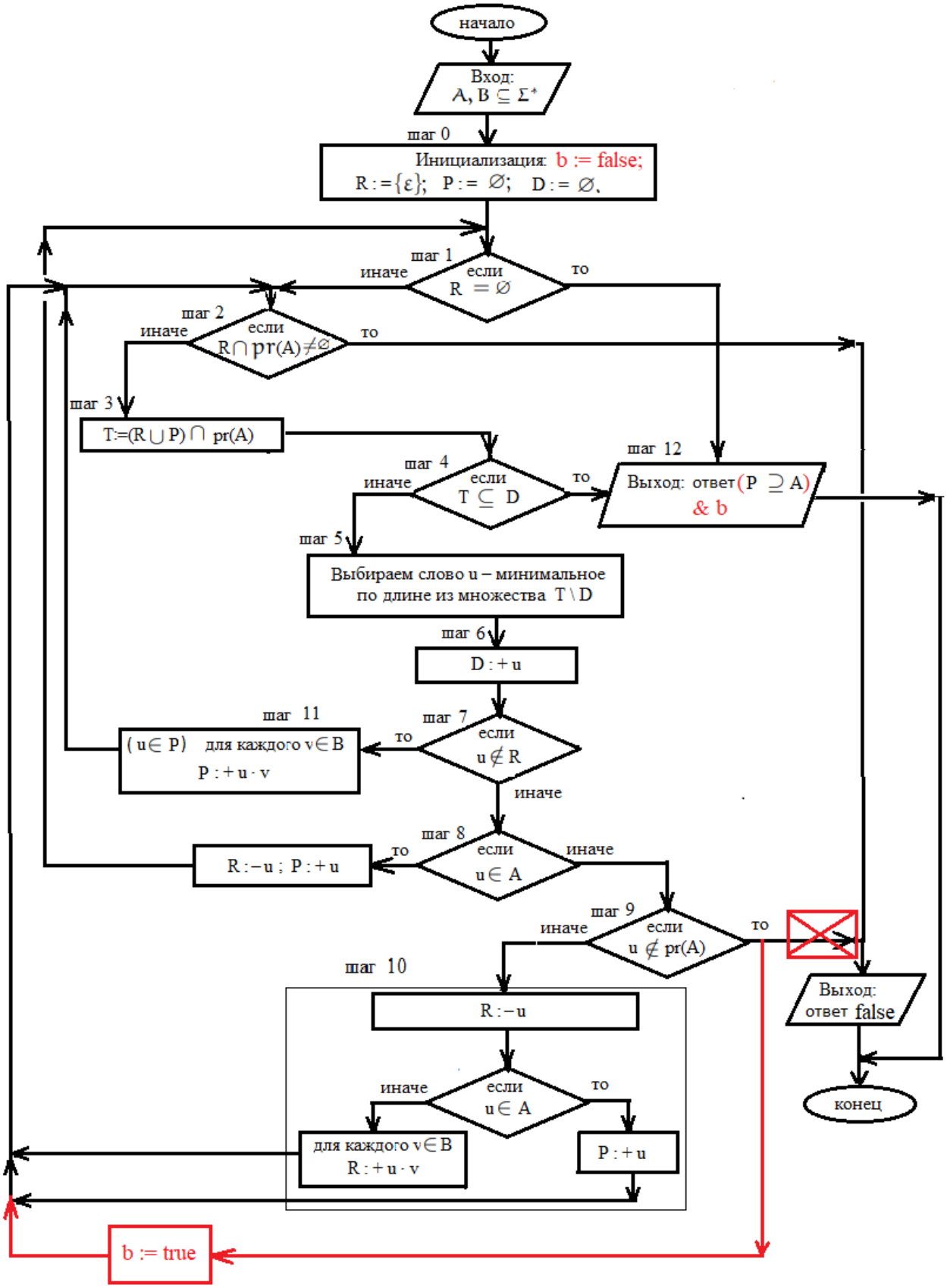


Рис. 1. Блок-схема главного вспомогательного алгоритма. «Без красного цвета» показана версия алгоритма, опубликованная ранее в [2], – где этот алгоритм рассматривался как основной предмет статьи; там он отвечал на вопрос, выполняется ли условие $A \in \mathrm{mp}^+(B)$ – из чего, в частности, следовало условие $B \rhd A$, которое ближе к материалу настоящей статьи.
Красным цветом на рисунке показаны внесённые изменения и добавления. Показан вариант алгоритма, отвечающий на вопрос, выполняется ли условие $B \nRightarrow A$. При необходимости проверки условия $B \Rightarrow A$ «красную» переменную $\mathfrak{b}$ и все связанные с нею действия можно не рассматривать, а изменение по сравнению с алгоритмом из [2] заключается только в перенаправлении одного из двух переходов после шага 9.

Разница (между двумя вариантами алгоритма – а можно считать, что и тремя) состоит в том, что в тех модификациях, которые нужны нам в настоящей статье, мы:

- либо дополнительно *требуем* наличия хотя бы одного *проверяемого* слова, *не принадлежащего* языку $\mathtt{pref}(A^*)$, – при одновременном выполнении всех остальных условий алгоритма из $\mathtt{pref}(A^*)$; в приведённых на рис. 1 обозначениях такой возможный «выход за пределы языка префиксов» [4] обозначается равным `true` значением переменной `b`; такой вариант алгоритма проверяет, выполняется ли условие $B \not\Rightarrow A$;
- либо не требуем, но *разрешаем* наличие такого слова; такой вариант алгоритма проверяет, выполняется ли условие $B \not\Rightarrow A$.

И, понятно, в [2] рассматриваемый вариант алгоритма был назван *основным* (поскольку в его описании и подробном рассмотрении заключался предмет той статьи) – мы же почти совпадающий с ним вариант называем вспомогательным.

Изменения и добавления, внесённые в предыдущую версию алгоритма, на рисунке 1 показаны красным цветом. При этом «без красного цвета» читается версия алгоритма, опубликованная ранее в [2], – где этот алгоритм рассматривался как основной предмет статьи; там он отвечал на вопрос, выполняется ли условие $A \in \mathfrak{mp}^+(B)$ – из чего, в частности, следовало условие $B \not\Rightarrow A$, которое ближе к материалу настоящей статьи.

Полностью (независимо от цвета) показан вариант алгоритма, отвечающий на вопрос, выполняется ли условие $B \not\Rightarrow A$. При необходимости проверки условия $B \not\Rightarrow A$ «красную» переменную `b` и все связанные с нею действия можно не рассматривать, а изменение по сравнению с алгоритмом из [2] заключается только в перенаправлении одного из двух переходов после шага 9.

Таким образом, можно понять, что изменения и дополнения совсем невелики – и поэтому мы в настоящей статье не приводим формального описания двух вариантов этого алгоритма.

К сказанному добавим такое замечание: на новой «красной» ветке вычислений, соответствующей варианту «то» из шага 9, действительно можно не делать присваивания `R:-u` (осуществляемого на «агрегатном» шаге 10): мы ведь во время выполнения алгоритма всё равно рассматриваем только такие слова, которые входят в язык $\mathtt{pref}(A^*)$.

Доказательство полиномиальности приведённого здесь алгоритма полностью совпадает с доказательством из [2]. Также почти совпадает и доказательство его корректности – конечно, с учётом замечаний, описанных выше в настоящей статье.

## V. ОПИСАНИЕ ОСНОВНОГО АЛГОРИТМА

Предлагаемый общий алгоритм заключается в выполнении алгоритма `Pot-Roots` (построении множества потенциальных корней исходного языка), а далее – в итерационном выполнении алгоритма `Del-Concat` (удаление потенциальных корней – конкатенаций, точнее, перевод их в другую переменную-язык) и «главного вспомогательного» алгоритма. При этом за каждый шаг итерации число слов рассматриваемого языка (языка B дальнейших обозначениях) уменьшается – и из этого факта и полиномиальности каждого из «вспомогательных» алгоритмов следует полиномиальность всего алгоритма; немного подробнее см. далее.

Перед описанием основного алгоритма сделаем ещё одно замечание. После завершения алгоритма `Del-Concat` в рассматриваемом множестве потенциальных корней (согласно дальнейшим обозначениям – C) могут остаться лишние элементы [5]; примеры на эту темы были приведены в наших предыдущих публикациях ([8], [9], [10] и др.), см. также в настоящей статье ниже, раздел VI. При этом обязательно выполнено условие $C \Rrightarrow A$ (то есть возможно и $C \triangleright A$, однако $C \not\triangleleft A$ – что в нашей ситуации равносильно $C \lhd A$ – невозможно).

Итак, *лишние* потенциальные корни могут остаться (конкретнее – остаться в языке B после перенесения некоторых из них в язык C) – но именно потому, *что мы предполагаем выполнение гипотезы* (Ⱨ) [6], у нас имеется *наименьшее* подходящее подмножество множества потенциальных корней (подробнее см. [10]). Иными словами – такие потенциальные корни могут остаться вследствие того, что их подмножества образуют полурешётку по пересечению [7]. И, также вследствие этого факта, возможен простой алгоритм определения таких корней – т. е. алгоритм, выполняющийся путём *последовательной* их проверки, не связанной с перебором множества подмножеств.

Вследствие вышесказанного мы в этом алгоритме проверяем каждое слово языка C – не является ли оно лишним? При этом любое *лишнее* слово таково, что его наличие приводит к невыполнению условия $A \cup \{u\} \not\Lsh A$ (из чего, в частности, следует, что $(A \cup \{u\}) \triangleright A$). Таким образом, мы будем использовать *специальную модификацию* алгоритма, описанного в [2]. Отметим, что о возможности подобной модификации в той статье было сказано явно – а в настоящей статье он приведён в разделе IV. Можно сказать, что сама модификация заключается в *полном* выполнении предыдущего алгоритма (т. е. алгоритма из [2]) [8] – но при этом в процессе его работы ищем хотя бы одно слово, не входящее в язык $\mathtt{pref}(A^*)$ – либо просто допускаем возможность существования таких слов.

На основе вышеизложенного опишем основной алгоритм – см. его блок-схему на рис. 2 в конце статьи.

**Основной алгоритм:** ***построение оптимального инверсного морфизма в случае выполнения гипотезы*** **(Ⱨ).**

*Вход:* непустой конечный язык $A \subseteq \Sigma^*$.

*Выход:* язык $\mathcal{D} \subseteq \Sigma^*$, такой что $A \in \mathfrak{mp}^+(\mathcal{D})$, причём среди всех возможных языков $D \subseteq \Sigma^*$, обладающих

[4] Подобный «выход» был запрещён в «немодифицированной версии алгоритма»: мы же там проверяли условие $\mathfrak{mp}^+$.

[5] Специально отметим, что слово *лишние* здесь пишем без кавычек.

[6] В нашем случае лучше рассматривать «основной вариант» этой гипотезы, т. е. именно (Ⱨ), «без штрихов».

[7] Наличие полурешётки по объединению очевидно, причём независимо от выполнения / невыполнения гипотезы (Ⱨ) – см. процитированные здесь работы.

[8] Здесь слово «полном» это означает, что мы не прерываемся в процессе его выполнения из-за специально описываемых «неожиданных ситуаций».

свойством $A \in \mathrm{mp}^+(D)$ [9], строимый язык $\mathcal{D}$ – минимальный согласно норме, получаемой на основе алгоритма `Part-Order`.

*Замечание 1.* Как было отмечено в [9], мы можем применять самые разные «естественные» варианты нормы на множестве конечных языков. Приведённая здесь (в описании выхода алгоритма) норма – один из таких возможных вариантов.

*Замечание 2.* Мы действительно можем говорить «минимальный согласно норме»: ведь согласно гипотезе (Ⱥ), таковым является искомый язык. Он (согласно этой гипотезе) есть, но мы его не знаем – его построение и есть предмет рассматриваемого алгоритма в частности и всей статьи вообще.

*Метод.*

*Шаг 1.* Применяем алгоритм `Pot-Roots` для построения языка $B \subseteq \Sigma^*$ – множества потенциальных корней языка $A$.

*Шаг 2.* Применяем алгоритм `Del-Concat` для построения языка $C \subseteq \Sigma^*$ – языка корней, представимых в виде конкатенации двух и более других потенциальных корней.

*Шаг 3.* В языке $B$ оставляем только слова, не вошедшие в $C$.

*Шаг 4.* Если $B = \varnothing$, то выход из алгоритма с ответом $C$.

*Шаг 5.* Рассматривая все слова $u \in B$, ищем любое из них, чтобы было выполнено условие

$$(B_{-u} \cup C) \not\Subset A;$$

выбираем первое возможное такое слово, далее оно также будет обозначаться $u$. Если такое слово не найдено, то выход из алгоритма с ответом $B$.

*Шаг 6.* $B := B \cup C \setminus \{u\}$ и переход на шаг 2.

*Конец описания алгоритма.*

Из всего сказанного выше выводится следующее основное утверждение статьи.

**Утверждение 3:** При выполнении гипотезы (Ⱥ) описанный алгоритм корректен – то есть он даёт такой минимальный язык $\mathcal{D} \subseteq \Sigma^*$, что

$$\mathcal{D} \in \mathrm{mp}^*(A);$$

или, что в нашей ситуации равносильно предыдущему,

$$\mathcal{D} \Subset A.$$

(Как и ранее, $A$ – заданный язык.)

*Доказательство.* Пусть здесь $D$ – минимальное множество слов (некоторое подмножество множества потенциальных корней) [10], выбираемое согласно гипотезе (Ⱥ); согласно этой же гипотезе, если для некоторого языка $D' \subseteq \Sigma^*$ верно $D' \Subset D$, то $D' \subseteq D^*$.

Рассмотрим некоторое слово $u \in C$, обозначим $C' = C \setminus \{u\}$. Условие $C' \Subset D$ невозможно: согласно гипотезе (Ⱥ), при этом должно было бы выполняться условие $u \in D^*$, а все такие $u$ были ранее удалены – поскольку, согласно сделанным обозначениям, $B \supseteq D$. Значит, если условие $C \Subset D$ не выполняется, то возможен лишь вариант

$$C \rhd C \setminus \{u\}.$$

А подобные слова $u$ удаляются на шаге 6 алгоритма, причём, согласно изложенному выше в настоящем доказательстве, возможно *последовательное* их удаление. □

**Утверждение 4:** Описанный основной алгоритм выполняется за полиномиальное время.

*Доказательство.* За каждый шаг итерации число слов рассматриваемого языка (языка $B$ дальнейших обозначениях) уменьшается – и из:

- этого факта;
- полиномиальности каждого из «вспомогательных» алгоритмов;
- полиномиального ограничения числа проверок потенциальных корней для «главного вспомогательного алгоритма» –

следует полиномиальность всего алгоритма. □

## VI. НЕСЛОЖНЫЙ ПРИМЕР

Рассматриваемый в настоящем разделе несложный пример «перекликается» с примерами, ранее рассматривавшимися в [3], [8], [9], [10] и некоторых других наших предыдущих статьях; см. также монографии [21], [22].

А именно, в указанных публикациях для языка

$$F = \{a, bab\} \tag{2}$$

мы рассматривали *в качестве исходного (заданного)* язык

$$E = F^2 = \{aa, abab, baba, babbab\}$$

и для последнего получали такое множество потенциальных квадратных корней:

$$\overline{\sqrt[2]{E}} = \{a, ab, ba, bab\} = F \cup \{ab, ba\}.$$

Здесь же для того же языка $F$ (2) будем рассматривать эквивалентный ему [11] язык

$$A = \{aa, abab, babababa, bab\}.$$

Действительно, эквивалентность $A \Subset F$ следует, например, из того, что

$$h_F^{-1}(A) = \{00, 01, 1010, 1\}$$

(для некоторого нового алфавита $\Delta = \{0, 1\}$ и соответствующего ему $F$-морфизма; $F$, как и ранее, задаётся согласно (2)) – а последний язык входит в множество языков $\mathrm{mp}^+(\Delta)$.

Множество потенциальных корней [12] здесь таково:

$$B = \overline{\sqrt[*]{A}} = A \cup \{a, ab, ba, baba\}.$$

Рассмотрим *возможный вариант* работы описанного выше алгоритма – мы уже фактически провели его шаг 1

[9] Таким свойством обладает, в частности, сам язык $A$. То есть множество языков, обладающих таким свойством, заведомо непустое.

[10] Мы *пока* этого множества не знаем – и можно сказать, что при выполнении гипотезы (Ⱥ) именно рассматриваемый в настоящей статье алгоритм строит это множество.

[11] Конечно, имеется в виду язык, эквивалентный с точки зрения бинарного отношения $\Subset$.

[12] Напомним, что в задаче построения оптимального инверсного морфизма – в отличие от задачи извлечения корня из языка – мы рассматриваем корни любой возможной степени.

(на который возвращаться не будем), причём используем те же самые обозначения переменных-языков.

*Первая итерация.* На шаге 2 (точнее, на первом его применении) мы определяем, например, такие представления слов языка B:

$$\begin{gathered} aa = a \cdot a; \\ abab = ab \cdot ab; \\ babababa = ba \cdot ba \cdot ba \cdot ba; \\ baba = ba \cdot ba. \end{gathered} \tag{3}$$

Замечания:

- здесь и далее слова рассматриваются не в алфавитном порядке, а в том, в котором они появляются в рассматриваемых нами записях языков – в частности, языка B;
- при наличии нескольких вариантов представления мы приводим только один из них – этого достаточно;
- остальные слова в виде конкатенаций не представляются.

Таким образом, после выполнения шагов 2 и 3 получаем:

$$\begin{gathered} B = \{ bab, a, ab, ba \}; \\ C = \{ aa, abab, babababa, baba \}. \end{gathered}$$

Далее мы получаем непустое значение на шаге 4, после чего на шаге 5 можем выбрать одно из следующих значений для слова $u$ (подробности применения вспомогательного алгоритма проверки выполнения условия $B \not\Rightarrow A$ приводить не будем):

$$u = ab \quad \text{или} \quad u = ba;$$

отметим, что примерно те же результаты получались при рассмотрении примеров из [8], [9], [10]. Будем считать, что выбрано слово $u = ab$, поэтому далее на шаге 6 получаем новое значение языка B:

$$B = \{ aa, abab, babababa, bab, a, ba, baba \}.$$

*Вторая итерация.* Для вновь полученного языка B при повторном выполнении шага 2 алгоритма каждое из слов, входящих в (3), также допускает представление в виде конкатенаций – однако одно из представлений должно быть иным:

$$abab = a \cdot bab.$$

Поэтому после выполнения шагов 2 и 3 получаем:

$$\begin{gathered} B = \{ bab, a, ba \}; \\ C = \{ aa, abab, babababa, baba \}. \end{gathered}$$

На шаге 5 снова можно выбрать $ba$, и на шаге 6 получаем новое значение языка B:

$$B = \{ aa, abab, babababa, bab, a, baba \}.$$

*Третья итерация.* Разложения слов вновь полученного языка B те же самые, поэтому после выполнения шагов 2 и 3 получаем:

$$\begin{gathered} B = \{ bab, a \}; \\ C = \{ aa, abab, babababa, baba \}. \end{gathered}$$

Здесь на шаге 5 мы ничего выбрать уже не можем, поэтому последняя версия языка B – который, как и можно было понять заранее, совпадает с F (2) – является языком-ответом.

## VII. ЗАКЛЮЧЕНИЕ

Как было сказано во введении, мы приводим *только одно* возможное направление дальнейших работ по рассмотренной в статье тематике. И, конечно, это направление можно назвать основным.

А именно, мы настоящей статьёй завершили доказательство существования *полиномиального* алгоритма проверки условия эквивалентности $A \Lsh\!\Rsh B$ для двух заданных конечных языков – при выполнении гипотезы (Ѫ). Поэтому при наличии доказательства *неполиномиальности* этой проверки (что может быть выполнено путём построения *за полиномиальное время* автомата NSPRI, соответствующего паре заданных языков – см. подробнее [6], [7] и др.) мы получим, что выполнение гипотезы (Ѫ) эквивалентно выполнению равенства P = NP.

## VIII. БЛАГОДАРНОСТИ

Работа частично поддержана грантом научной программы китайских университетов “Higher Education Stability Support Program” (раздел “Shenzhen 2022 – Science, Technology and Innovation Commission of Shenzhen Municipality”).

## Список литературы

[1] Melnikov B., Korabelshchikova S., Dolgov V. *On the task of extracting the root from the language* // International Journal of Open Information Technologies. – 2019. – Vol. 7. No. 3. – P. 1–6.

[2] Мельников Б., Мельникова А. *Полиномиальный алгоритм проверки выполнения условия морфического образа расширенного максимального префиксного кода* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 12. – P. 1–9.

[3] Мельников Б., Мельникова А. *О задачах извлечения корня из заданного конечного языка* // International Journal of Open Information Technologies. – 2023. – Vol. 11. No. 5. – P. 1–14.

[4] Melnikov B. *The equality condition for infinite catenations of two sets of finite words* // International Journal of Foundation of Computer Science. – 1993. – Vol. 4. No. 3. – P. 267–274.

[5] Алексеева А., Мельников Б. *Итерации конечных и бесконечных языков и недетерминированные конечные автоматы* // Вектор науки Тольяттинского государственного университета. – 2011. – № 3 (17). – С. 30–33.

[6] Мельников Б., Мельникова А. *Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть I* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 4. – P. 1–11.

[7] Мельников Б., Мельникова А. *Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть II* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 5. – P. 1–11.

[8] Мельников Б. *Полурешётки подмножеств потенциальных корней в задачах теории формальных языков. Часть I. Извлечение корня из языка* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 4. – P. 1–9.

[9] Мельников Б. *Полурешётки подмножеств потенциальных корней в задачах теории формальных языков. Часть II. Построение инверсного морфизма* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 5. – P. 1–8.

[10] Мельников Б. *Полурешётки подмножеств потенциальных корней в задачах теории формальных языков. Часть III. Условие существования решётки* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 7. – P. 1–9.

[11] Melnikov B., Melnikova A. *Some properties of the basis finite automaton* // The Korean Journal of Computational and Applied Mathematics (Journal of Applied Mathematics and Computing). – 2002. – Vol. 9. No. 1. – P. 135–150.

[12] Melnikov B., Sciarini-Guryanova N. *Possible edges of a finite automaton defining a given regular language* // The Korean Journal of Computational and Applied Mathematics (Journal of Applied Mathematics and Computing). – 2002. – Vol. 9. No. 2. – P. 475–485.

[13] Мельников Б., Сайфуллина М. *О некоторых алгоритмах эквивалентного преобразования недетерминированных конечных автоматов* // Известия высших учебных заведений. Математика. – 2009. – № 4. – С. 67–72.

[14] Мельников Б., Мельникова А. *Применение лепестковых конечных автоматов для проверки выполнения частного случая гипотезы Зуи (для заданного конечного языка)* // International Journal of Open Information Technologies. – 2023. – Vol. 11. No. 3. – P. 1–11.
[15] Лаллеман Ж. *Полугруппы и комбинаторные приложения*. – М., Мир. – 1985. – 440 с.
[16] Саломаа А. *Жемчужины теории формальных языков*. – М., Мир. – 1986. – 159 с.
[17] Скорняков Л. (ред.) *Общая алгебра. Том 1*. – М., Наука. – 1990. – 592 с.
[18] Хаусдорф Ф. *Теория множеств*. – М., УРСС. – 2007. – 304 с.
[19] Гуров С. *Булевы алгебры, упорядоченные множества, решетки: определения, свойства, примеры*. – М., Либроком. – 2013. – 221 с.
[20] Мельников Б., Мельникова А. *Полиномиальный алгоритм построения конечного автомата для проверки равенства бесконечных итераций двух конечных языков* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 11. – P. 1–10.
[21] Мельников Б. *Подклассы класса контекстно-свободных языков (монография)*. – М., Изд-во Московского университета. – 1995. – 174 с. – ISBN 5-211-03448-1.
[22] Мельников Б. *Регулярные языки и недетерминированные конечные автоматы (монография)*. – М., Изд-во Российского государственного социального ун-та. – 2018. – 179 с. – ISBN 978-5-7139-1355-7.

Борис Феликсович МЕЛЬНИКОВ,
профессор Университета МГУ – ППИ в Шэньчжэне
(http://szmsubit.ru/),
email$_1$: bormel@smbu.edu.cn,
email$_2$: bf-melnikov@yandex.ru,
mathnet.ru: personid=27967,
elibrary.ru: authorid=15715,
scopus.com: authorId=55954040300,
ORCID: orcidID=0000-0002-6765-6800.

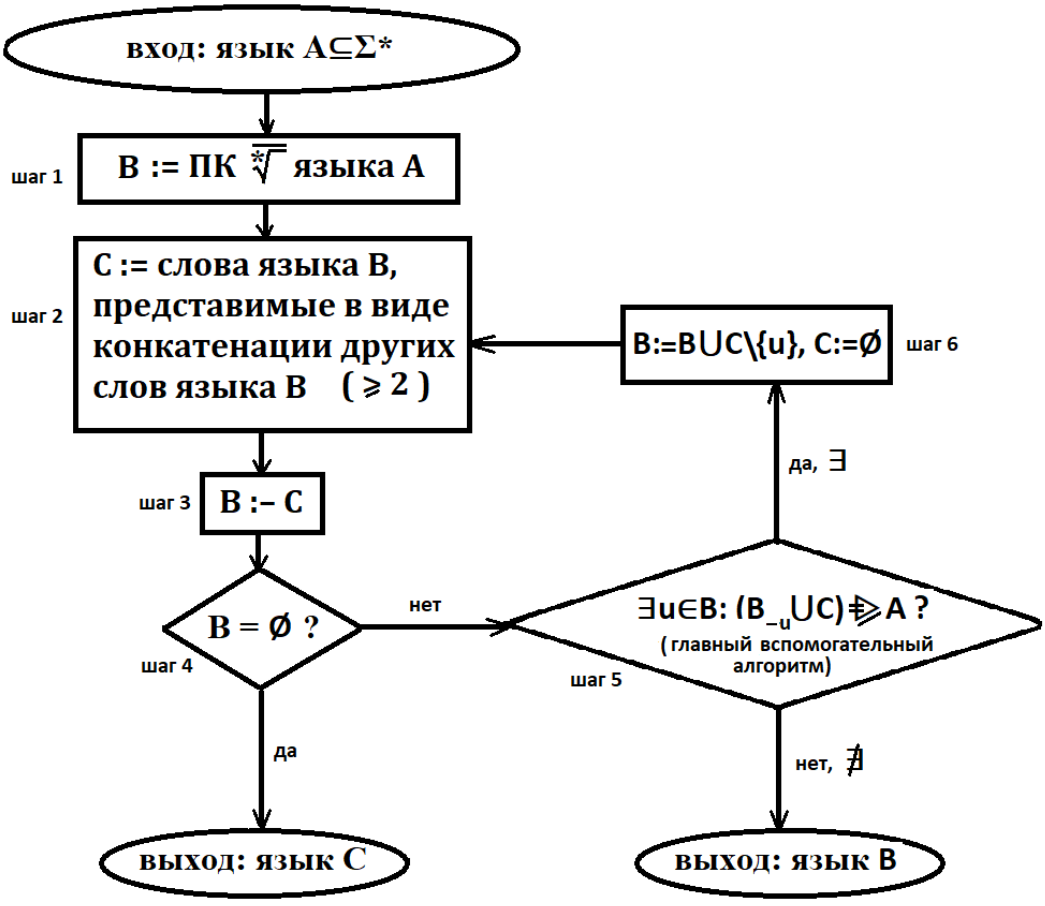


Рис 2. Блок-схема основного алгоритма.

# A polynomial algorithm for constructing the optimal inverse morphism

Boris Melnikov

***Abstract*—In this paper, we continue the theme of some our previous works. In this topic, we consider two similar problems of the formal languages theory: the problem of extracting the root of a given degree and the problem of constructing an optimal inverse morphism, where optimality can be defined as the length of the maximum word of a language that is an inverse morphic image. Instead of considering all subsets of the set of so-called potential roots, which always leads to exponential algorithms, we obtained some various polynomial algorithms in these publications.**

**In order to formulate the second of these problems in a way convenient for constructing polynomial algorithms, we have considered some equivalent versions of the special hypothesis of the formal languages theory in previous papers. One of its equivalent formulations can be expressed as follows. For two finite languages not containing the empty word, we can consider the necessary and sufficient condition that the iteration of any of both languages belongs to the set of prefixes of the other language, and write it as follows. There is some new alphabet, different from the alphabet over which the source languages are set, and above this alphabet there are two maximum prefix codes (generally speaking, different ones) and two languages containing these maximum prefix codes as subsets. In addition, there is some special morphism, and the source languages should be formed by applying this morphism to the languages containing these maximal prefix codes as subsets.**

**In this paper, we show that when this hypothesis is fulfilled, a polynomial algorithm for constructing an optimal inverse morphism is possible. Moreover, simplifying the situation a little, we can say that such an algorithm is a special combination of several polynomial algorithms considered in our previous papers, or simple modifications of such algorithms.**

***Keywords*—formal languages, iterations of languages, morphisms, inverse morphisms, binary relations, algorithms.**

## References

[1] Melnikov B., Korabelshchikova S., Dolgov V. *On the task of extracting the root from the language* // International Journal of Open Information Technologies. – 2019. – Vol. 7. No. 3. – P. 1–6.

[2] Melnikov B., Melnikova A. *A polynomial algorithm for checking the fulfillment of the condition of the morphic image of the extended maximal prefix code* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 12. – P. 1–9 (in Russian).

[3] Melnikov B., Melnikova A. *On the problems of extracting the root from a given finite language* // International Journal of Open Information Technologies. – 2023. – Vol. 11. No. 5. – P. 1–14 (in Russian).

[4] Melnikov B. *The equality condition for infinite catenations of two sets of finite words* // International Journal of Foundation of Computer Science. – 1993. – Vol. 4. No. 3. – P. 267–274.

[5] Alekseeva A., Melnikov B. *Iterations of finite and infinite languages and nondeterministic finite automata* // Vector of Science of Togliatti State University. – 2011. – No. 3 (17). – P. 30–33 (in Russian).

[6] Melnikov B., Melnikova A. *Infinite trees in the algorithm for checking the equivalence condition of iterations of finite languages. Part I* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 4. – P. 1–11 (in Russian).

[7] Melnikov B., Melnikova A. *Infinite trees in the algorithm for checking the equivalence condition of iterations of finite languages. Part II* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 5. – P. 1–11 (in Russian).

[8] Melnikov B. *Semi-lattices of the subsets of potential roots in the problems of the formal languages theory. Part I. Extracting the root from the language* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 4. – P. 1–9 (in Russian).

[9] Melnikov B. *Semi-lattices of the subsets of potential roots in the problems of the formal languages theory. Part II. Constructing an inverse morphism* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 5. – P. 1–8 (in Russian).

[10] Melnikov B. *Semi-lattices of the subsets of potential roots in the problems of the formal languages theory. Part III. The condition for the existence of a lattice* // International Journal of Open Information Technologies. – 2022. – Vol. 10. No. 7. – P. 1–9 (in Russian).

[11] Melnikov B., Melnikova A. *Some properties of the basis finite automaton* // The Korean Journal of Computational and Applied Mathematics (Journal of Applied Mathematics and Computing). – 2002. – Vol. 9. No. 1. – P. 135–150.

[12] Melnikov B., Sciarini-Guryanova N. *Possible edges of a finite automaton defining a given regular language* // The Korean Journal of Computational and Applied Mathematics (Journal of Applied Mathematics and Computing). – 2002. – Vol. 9. No. 2. – P. 475–485.

[13] Melnikov B., Sayfullina M. *On some algorithms for equivalent transformation of nondeterministic finite automata* // News of higher educational institutions. Mathematics. – 2009. – No. 4. – P. 67–72 (in Russian).

[14] Melnikov B., Melnikova A. *The use of petal finite automata to verify the fulfillment of a special case of the Zyu hypothesis (for a given finite language)* // International Journal of Open Information Technologies. – 2023. – Vol. 11. No. 3. – P. 1–11 (in Russian).

[15] Lallement G. *Semigroups and Combinatorial Applications*. – NY, John Wiley & Sons. – 1979. – 376 p.

[16] Salomaa A. Jewels of Formal Language Theory. Computer Science Press, Rockville, Maryland (1981). – 144 p.

[17] Skornyakov L. (Ed.) *General Algebra. Vol. 1*. – Moscow, Nauka. – 1990. – 592 p. (in Russian).

[18] Hausdorff F. *Grundzüge der Mengenlehre*. – Grundzüge der Mengenlehre, von Veit. – 1914. – ISBN 978-0-8284-0061-9. (Reprinted by Chelsea Publishing Company in 1949.)

[19] Gurov S. *Boolean algebras, ordered sets, lattices: definitions, properties, examples*. – Moscow, Librokom. – 2013. – 221 p. (in Russian).

[20] Melnikov B., Melnikova A. *A polynomial algorithm for constructing a finite automaton for checking the equality of infinite iterations of two finite languages* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 11. – P. 1–10 (in Russian).

[21] Melnikov B. *Subclasses of the context-free languages class (monograph)*. – Moscow, Moscow State University Ed. – 1995. – 174 p. – ISBN 5-211-03448-1 (in Russian).

[22] Melnikov B. *Regular languages and nondeterministic finite automata (monograph)*. – Moscow, Russian Social State University Ed. – 2018. – 179 p. – ISBN 978-5-7139-1355-7 (in Russian).

Boris MELNIKOV,
Professor of Shenzhen MSU – BIT University, China
(http://szmsubit.ru/),
email$_1$: bormel@smbu.edu.cn,
email$_2$: bf-melnikov@yandex.ru,
mathnet.ru: personid=27967,
elibrary.ru: authorid=15715,
scopus.com: authorId=55954040300,
ORCID: orcidID=0000-0002-6765-6800.