Полиномиальный алгоритм проверки выполнения условия морфического образа расширенного максимального префиксного кода

Б. Ф. Мельников, А. А. Мельникова

Аннотация—**Максимальный префиксный код определяется обычным образом – «на основе изложенного в студенческих курсах». Расширенный максимальный префиксный код – это конечный язык, содержащий некоторый максимальный префиксный код в качестве подмножества (собственного или несобственного). Также обычным образом определяются (гомо)морфизмы, и, на их основе, инверсные морфизмы.**

В наших предыдущих публикациях были рассмотрены бесконечные итерации конечных языков – как омега-языки. Была сформулирована гипотеза о том, что бесконечные итерации двух заданных конечных языков совпадают тогда и только тогда, когда оба этих языка могут быть получены с помощью следующего алгоритма. Во-первых, выбирается некоторый новый алфавит; во-вторых, над этим алфавитом рассматриваются некоторые два расширенных максимальных префиксных кода; в-третьих, к этим двум расширенным максимальным префиксным кодам применяется некоторый (один и тот же в обоих случаях) морфизм – переводящий слова над новым алфавитом в слова над алфавитом исходным. При этом полученные морфические образы двух рассматриваемых расширенных максимальных префиксных кодов должны совпасть с двумя исходными (заданными) конечными языками.

(Более точно – было сформулировано несколько эквивалентных вариантов этой гипотезы, а также другая гипотеза, представляющая собой менее сильное утверждение, о котором имеет смысл говорить в случае невыполнения гипотезы первой.)

В настоящей статье решается задача проверки, может ли один язык быть получен с помощью подобного алгоритма, применённого к некоторому другому языку. Точнее, недетерминированный алгоритм для такой задачи тривиален (и приведён в одной из наших предыдущих публикаций) – здесь же мы приводим детерминированный, причём полиномиальный, алгоритм проверки выполнения этого условия. Таким образом, рассматриваемую в настоящей статье задачу можно считать шагом на пути проверки сформулированной выше гипотезы.

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

I. ВВЕДЕНИЕ, ОСНОВНЫЕ ОБОЗНАЧЕНИЯ, МОТИВАЦИЯ И ПОСТАНОВКА ЗАДАЧИ

В статье мы продолжаем тематику публикаций [1], [2], [3], [4]. В этих статьях мы исследовали свойства специального бинарного отношения $\Diamond$, впервые рассмотренного нами (в несколько иной интерпретации) ещё в 1993 г., в [5] – т. н. отношения эквивалентности в бесконечности. В дальнейших публикациях (некоторые ссылки будут приведены ниже) мы рассмотрели примеры применения этого отношения – причём эти примеры можно разделить на две группы:

- примеры необходимости его выполнения;
- примеры его использования – в различных областях теории формальных языков, дискретной математики и абстрактной алгебры.

Мы не стали разделять этот вводный раздел на несколько (как делаем почти всегда) – несмотря на достаточно большой его объём. Объяснить это можно так: для рассматриваемой здесь тематики все «составляющие части введения» – основные обозначения, мотивация и постановка задачи – получаются существенно связанными между собой. Применённое же деление раздела на *под*разделы – очень условное.

A. Оновные обозначения, префиксы и морфизмы

Сначала повторим определения бинарного отношения $\Diamond$ и связанных с ним понятий – близко к [3], [6] и др.

Для заданного конечного алфавита Σ («главного», «основного») и слова $u \in \Sigma^*$ над этим алфавитом:

- язык $\mathtt{pref}(u)$ определяется как множество префиксов слова u (включая несобственный, само u);
- язык $\mathtt{opref}(u)$ определяется как множество собственных префиксов слова u.

Для заданного языка $A \subseteq \Sigma^*$

$$\mathtt{pref}(A) = \bigcup_{u \in A} \mathtt{pref}(u);$$

аналогичным образом определяется и язык $\mathtt{opref}(A)$.

Если для конечных языков A и B (в наших статьях они практически всегда рассматриваются над этим «основным» алфавитом Σ) выполнено условие

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

то будем писать $A \trianglelefteq B$ (либо $B \trianglerighteq A$). Если одновременно выполнены условия $A \trianglelefteq B$ и $A \trianglerighteq B$, мы будем писать $A \Diamond B$.

(Специально отметим ещё раз разницу в обозначениях: в наших последних статьях с одной стороны, и, с другой стороны, в [5] и нескольких статьях 1990-х годов. Конечно же, в настоящей статье мы будем применять только обозначения, приведённые в этом разделе.)

Далее считаем, что все рассматриваемые языки – непустые и не содержащие ε. (Если иного не сказано в их

Статья получена 18 августа 2022 г.

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

Александра Александровна Мельникова, Димитровградский инженерно-технологический институт – филиал Национального исследовательского ядерного университета «МИФИ» (super-avahi@yandex.ru).

определениях – явно или неявно; а «неявно» – это, например, в случае задания языка как итерации некоторого другого.)

Для некоторого языка

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

мы рассматриваем алфавит

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

(причём если это не вызывает неоднозначностей, мы обычно пишем просто Δ), а для последнего – рассматриваем морфизм

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

задающийся следующим образом [1]:

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

Если A не обладает свойством префикса, то определённый здесь морфизм также будем называть *непрефиксным* [2].

B. Инверсные морфизмы и (расширенные) максимальные префиксные коды

Предмет настоящей статьи связан с более важной и более сложной задачей (более сложной, чем построение морфизма) – задачей построения *инверсного морфизма*. Мы её начали рассматривать в [6], [7], [8] (три части одной статьи) – и с возможным названием этой более сложной задачи совпадает *подзаголовок* одной из процитированных здесь частей. Отметим, что кратко эта задача была рассмотрена ещё в [9] (1995 г.).

Сформулируем условие этой более сложной задачи подробнее.

Сначала – такое естественное определение [3]. Для заданных конечного языка A, морфизма h_A и некоторого слова $u \in \Sigma^*$ рассмотрим *язык* [4]

$$h_A^{-1}(u) = \left\{ u_\Delta \in \Delta^* \,\middle|\, h_A(u_\Delta) = u \right\};$$

специально подчеркнём, что определённое понятие является *множеством*. Ту же самую конструкцию можно рассматривать и для некоторого языка (вместо слова u; пусть это язык B) – причём нас будут интересовать только конечные языки. А именно:

$$h_A^{-1}(B) = \bigcup_{u \in B} h_A^{-1}(u).$$

Теперь рассмотрим другое определение, не связанное с предыдущим. В нём «основной» алфавит – Δ. Над этим алфавитом множество максимальных префиксных кодов – как множество языков – во многих наших предыдущих работах обозначалось $\mathfrak{mp}(\Delta)$.

(Само определение «одного» максимального префиксного кода – также как некоторого языка – мы приводить не будем. Можно сказать, что это – код, который является максимальным и префиксным; на самом деле, этого будет достаточно, поскольку все эти понятия – из «студенческих курсов», т. е. в само́м названии уже содержится определение. Однако, с другой стороны, всё не столь тривиально – см. некоторые комментарии, включающие определение кодового индекса, в [6]. И там же, в [6], приведён простейший недетерминированный алгоритм получения любого максимального префиксного кода над заданным алфавитом – причём этот алгоритм можно рассматривать как ещё одно альтернативное определение.)

А множество языков, каждый из которых *содержит* некоторый максимальный префиксный код *в качестве подмножества* (возможно, несобственного), обозначается $\mathfrak{mp}^+(\Delta)$. Таким образом,

$$\mathfrak{mp}(\Delta) \subset \mathfrak{mp}^+(\Delta),$$

причём, конечно, для любого алфавита включение собственное. Мы будем называть каждый из таких языков *расширенным максимальным префиксным кодом* [5,6].

Вернёмся к рассмотрению «основного» алфавита, т. е. Σ, а также морфизмов вида

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

Пусть у нас есть некоторый язык

$$A_\Delta \in \mathfrak{mp}(\Delta);$$

тогда будем считать, что выполнено условие

$$h_A(A_\Delta) \in \mathfrak{mp}(A)$$

(мы таким образом определяем множество языков $\mathfrak{mp}(A)$ над рассматриваемым алфавитом Σ). Аналогично, для некоторого языка

$$A_\Delta \in \mathfrak{mp}^+(\Delta)$$

будем считать, что

$$h_A(A_\Delta) \in \mathfrak{mp}^+(A).$$

Таким образом:

- $\mathfrak{mp}^+(\Delta)$ – это множество языков, каждый из которых содержит в качестве подмножества некоторый максимальный префиксный код над алфавитом Δ;
- $\mathfrak{mp}^+(A)$ – это множество языков, каждый из которых является A-морфизмом некоторого языка из множества $\mathfrak{mp}^+(A)$; т. е. каждый из таких языков – специальный морфизм некоторого расширенного максимального префиксного кода.

Важно отметить, что в [6], [7], [8] приведены *недетерминированные бесконечные* алгоритмы построения всех языков множеств $\mathfrak{mp}(\Delta)$ и $\mathfrak{mp}^+(\Delta)$ – однако, конечно, эти алгоритмы не являются такими, которые могут удовлетворять заголовку настоящей статьи, т. е. её предмету. К описанию предмета мы сейчас и переходим.

[1] Неточностей во вводимых обозначениях нет, см. подробнее предыдущие публикации. Там же см. немного более подробное описание обозначений.

[2] Как правило, нас интересуют именно такие морфизмы.

[3] Во введении мы определения не нумеруем, поскольку они приведены в наших предыдущих публикациях. При этом нередко – существенно более подробно, с примерами и т. п.

[4] Состоящий, вообще говоря, не из одного, а из нескольких слов – а, возможно, являющийся пустым языком.

[5] Несмотря на приведённые в этом определении слова «содержит в качестве подмножества» – конечно, *не любой* язык можно назвать расширенным максимальным префиксным кодом над заданным алфавитом. Тривиальный пример – $\Delta = \{a, b\}$, а язык $\{a\}$. Этот язык действительно *был бы* расширенным максимальным префиксным кодом (и даже просто максимальным префиксным кодом) – но над алфавитом из одной буквы, а не над таким Δ.

[6] Термин «расширенный» введён нами. Он представляется более удачным, чем близкое по смыслу слово «дополненный».

C. Собственно постановка задачи

Итак, как следует из названия статьи, мы описываем *детерминированный полиномиальный алгоритм проверки выполнения условия морфического образа расширенного максимального префиксного кода* – то есть проверки условия $A \in \mathfrak{mp}^{+}(B)$ для заданных непустых конечных языков $A, B \subseteq \Sigma^{*}$. Этот алгоритм можно считать одной из важных *вспомогательных* задач для исследования сформулированного выше отошения $\Diamond$ на множестве непустых конечных языков.

Но, конечно, приведённая в предыдущем абзаце постановка задачи – очень краткая. Поэтому повторим (с некоторыми дополнениями) тот вариант постановки задачи, который был приведён в расширенной аннотации. Такое более полное описание постановки основано на рассмотренных в предыдущем подразделе определениях максимальных префиксных кодов, морфизмов и инверсных морфизмов.

В наших предыдущих публикациях были рассмотрены бесконечные итерации конечных языков – как омега-языки. При этом бесконечные итерации двух заданных конечных языков A и B совпадают тогда и только тогда, когда для них выполняется описанное выше бинарное отношение $\Diamond$:

$$A^{\omega} = B^{\omega} \quad \text{тогда и только тогда, когда} \quad A \Diamond B$$

Была сформулирована гипотеза о том, что это происходит тогда и только тогда, когда *оба* этих языка *могут быть получены* с помощью следующего алгоритма:

- во-первых, выбирается *некоторый*[7] новый конечный алфавит;
- во-вторых, над этим алфавитом рассматриваются *некоторые* два расширенных максимальных префиксных кода;
- в-третьих, к этим двум расширенным максимальным префиксным кодам применяется *некоторый* (причём обязательно *один и тот же в обоих случаях*) морфизм – переводящий слова над новым алфавитом в слова над алфавитом исходным.

При этом полученные морфические образы двух рассматриваемых расширенных максимальных префиксных кодов должны совпасть с двумя исходными конечными языками.

(Более точно. В предыдущих работах было сформулировано *несколько эквивалентных вариантов* этой гипотезы – и, кроме того, также была сформулирована другая гипотеза, представляющая собой менее сильное утверждение, о котором имеет смысл говорить в случае невыполнения гипотезы первой. В [8, разд. XIV] эти гипотезы были обобщены: это эквивалентные между собой варианты первой гипотезы, обозначенные ($\mathfrak{A}$), ($\mathfrak{A}'$), ($\mathfrak{A}''$) и ($\mathfrak{A}'''$), а также менее сильная – но при этом более сложная для формулировки – вторая гипотеза, обозначенная ($\mathfrak{AA}$).)

[7] При кратком описании этого алгоритма слово «некоторый» употребляется часто – и выделяется курсивом; аналогично – слова «могут быть получены», см. выше. Это связано с тем, что мы *заранее не можем знать*, какой именно объект (какое именно отображение) должно применяться.

Алгоритмы определения (построения) таких конкретных объектов (конкретных отображений) – это, по большому счёту, и есть комплекс задач, рассматриваемых в наших публикациях по этой тематике.

Вернёмся к подробному описанию постановки задачи. В настоящей статье решается задача проверки, *может ли один язык быть получен с помощью подобного алгоритма, применённого к некоторому другому языку*. Ещё точнее – поскольку недетерминированный алгоритм для такой задачи тривиален (он также был приведён в одной из наших предыдущих публикаций, [7, разд. XII][8]) – здесь мы приводим *детерминированный, причём полиномиальный*, алгоритм проверки выполнения этого условия. Таким образом, рассматриваемую в настоящей статье задачу можно считать шагом на пути проверки сформулированной гипотезы.

D. Мотивация

Итак, в настоящей статье мы продолжаем тематику, связанную со специальным бинарным отношением на множестве формальных языков (рассматриваемом в первую очередь на множестве итераций непустых конечных языков) – т. н. отношением эквивалентности в бесконечности. Выше мы также сформулировали более простое бинарное отношение на множестве языков – т. н. отношение покрытия, двойное применение которого равносильно применению отношения эквивалентности в бесконечности. В предыдущих публикациях мы рассмотатривали алгоритм проверки выполнения отношения покрытия, а также определили вспомогательные объекты, используемые для доказательства корректности этого алгоритма. С одним из подобных объектов – а именно, с расширенным максимальным префиксным кодом, точнее, с бесконечным множеством таких кодов, – мы и работаем в настоящей статье.

Также в предыдущих публикациях (прежде всего – в [1], [2]) мы определили ещё один вид таких объектов, т. н. бесконечные итерационные деревья. После этого мы объединили эквивалентные вершины этих деревьев (эквивалентность вершин рассматривается на основе определения таких деревьев) – фактически получая при этом детерминированный конечный автомат.

Сначала мы определяем именно такой конкретный автомат для двух заданных конечных языков (подлежащих проверке на выполнение отношения $\Diamond$) – т. н. автомат PRI; он, как уже было отмечено, *детерминированный*, определён *на множествах слов* – и каждое из этих множеств является подмножеством множества префиксов второго из заданных языков[9].

После этого мы определяем соответствующие такому автомату несколько вариантов специальных *недетерминированных* автоматов, фактически описывающих построение заданного итерационного дерева морфизма. Мы вводим совершенно иной объект – т. н. автомат NSPRI, – который также описывает построение итерационного дерева морфизма, но определён не на конечных множествах слов (конечных языках), а на (отдельных) словах. (Подробнее про автоматы NSPRI и PRI см. в [3], [4], [10] и др.)

Название настоящей статьи начинается со слов «полиномиальный алгоритм» – поэтому скажем немного о

[8] Более того, приведённый там недетерминированный алгоритм решает более сложную задачу – к ней мы вернёмся в заключении.

[9] Возможен также вариант, когда вместо множества префиксов используется множество суффиксов.

полиномиальности другого, связанного алгоритма, описанного нами в [11]. Основная идея доказательства того факта, что алгоритм *построения конечного автомата для проверки равенства бесконечных итераций двух конечных языков* полиномиален, такова. Естественным является построение для этой задачи детерминированного автомата – каждое состояние которого описывает целое множество возможных префиксов, оставшихся от различных разложений слова – т. е. разложений морфизма итерации первого заданного конечного языка по словам второго заданного конечного языка. Однако при этом мы работаем с множеством множеств возможных префиксов – что полиномиальность алгоритма делает невозможным. Поэтому мы строим *недетерминированный* автомат – определённый просто на множестве возможных префиксов [10]. При этом возникает вопрос – когда именно этот автомат даёт положительный результат. Мы решаем эту проблему тем, что требуем, чтобы получающийся недетерминированный автомат являлся бы аналогом всюду определённого детерминированного автомата [11].

(При этом условие того, что недетерминированный автомат является аналогом всюду определённого детерминированного автомата, *за полиномиальное время проверить невозможно* – см. подробнее [11]. Однако именно невозможность подобной проверки может дать доказательство невозможности проверки за полиномиальное время и условия выполнения отношения $\Cap$ для двух заданных конечных языков.)

В ближайшем будущем мы предполагаем опубликовать специальную статью, посвящённую именно мотивации к рассмотрению задач, связанных с алгоритмами проверки бинарного отношения $\Cap$.

E. Содержание статьи по разделам

Существенная часть статьи относится к подразделам настоящего раздела. В оставшейся её части мы рассмотрим:

- сам основной алгоритм статьи (раздел II) – при этом приведена также блок-схема алгорима;
- доказательство корректности этого алгоритма, в том числе его полиномиальности (раздел III) – впрочем, стоит отметить, что все эти доказательства несложные, и их можно считать просто следствиями описания алгоритма;
- а также небольшое заключение (раздел IV), в котором приведены только самые основные возможные направления дальнейших работ по рассматриваемой тематике.

II. ОПИСАНИЕ АЛГОРИТМА

Здесь мы приводим *формальное описание* основного алгоритма настоящей статьи; его блок-схема приведена на рис. 1.

Алгоритм II.1. *Проверка условия морфического образа расширенного максимального префиксного кода.*

Вход: конечные языки $A, B \subseteq \Sigma^*$, причём

$$A \neq \varnothing,\ A \not\ni \varepsilon,\ B \neq \varnothing \text{ и } B \not\ni \varepsilon.$$

Выход:

- `true` – если $A \in \mathfrak{mp}^+(B)$;
- `false` – в противном случае.

Метод.

Шаг 0. Инициализация:

- создать изменяемое в процессе работы алгоритма множество слов `requiritur`, при инициализации полагаем
$$\texttt{requiritur} := \{\varepsilon\};$$
- создать изменяемое в процессе работы алгоритма множество слов `probabile`, при инициализации полагаем
$$\texttt{probabile} := \varnothing;$$
- создать изменяемое в процессе работы алгоритма множество слов `discursum` [12], при инициализации полагаем
$$\texttt{discursum} := \varnothing.$$

Шаг 1. Если $\texttt{requiritur} = \varnothing$, то переход на шаг 12 (для окончания работы алгоритма).

Шаг 2. (Здесь $\texttt{requiritur} \neq \varnothing$.)
Если при этом

$$\texttt{requiritur} \cap \mathbf{pref}(A) \neq \varnothing,$$

то выход из алгоритма с ответом `false`.

Шаг 3. Рассмотрим множество

$$\texttt{tempore} := (\texttt{requiritur} \cup \texttt{probabile}) \cap \mathbf{pref}(A);$$

про него отметим следующее:

- оно заведомо непустое [13];
- при будущих выполнениях этого шага оно будет, вообще говоря, иным.

Шаг 4. Если $\texttt{tempore} \subseteq \texttt{discursum}$, то переход на шаг 12 (для окончания работы алгоритма).

Шаг 5. (Здесь $\texttt{tempore} \setminus \texttt{discursum} \neq \varnothing$.)
Среди слов множества $\texttt{tempore} \setminus \texttt{discursum}$ выбираем минимальное по длине (если таковых несколько – то выбираем любое из них); пусть это – слово u.

Шаг 6. $\texttt{discursum} :+ u$
(т. е. добавляем u в это множество).

Шаг 7. Если $u \notin \texttt{requiritur}$, то переход на шаг 11.

Шаг 8. (Здесь $u \in \texttt{requiritur}$.)
Если при этом также $u \in A$, то выполняем следующие действия:

- $\texttt{requiritur} :- u$
(т. е. удаляем u из этого множества);

[10] Повторим ещё раз, что *не* на множестве подмножеств таких возможных префиксов.

[11] Определение такого «аналога» очевидно. Если говорить неформально – должны быть заполнены все клетки таблицы, описывающей граф переходов (transition function) рассматриваемого недетерминированного автомата.

[12] В процессе работы алгоритма в это множество слова только добавляются.

[13] Действительно, мы знаем, что $\texttt{requiritur} \neq \varnothing$. При этом в `requiritur` нет слов, не принадлежащих $\mathbf{pref}(A)$. Значит, в одном лишь `requiritur` (даже не рассматривая `probabile`) хотя бы одно слово из множества $\mathbf{pref}(A)$ имеется.

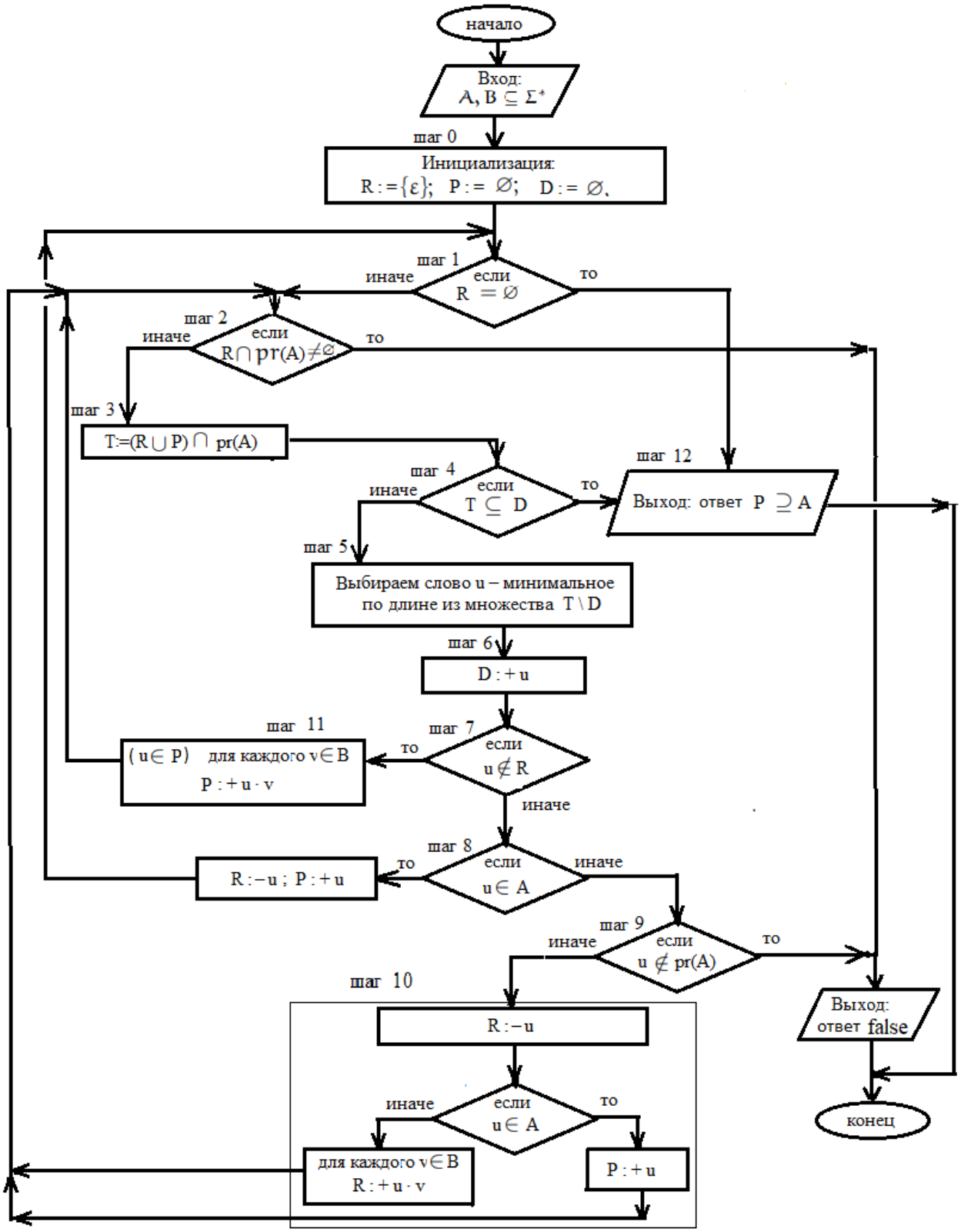

Рис. 1. Блок-схема основного алгоритма.

- $\texttt{probabile} :+ u$
  (т. е. добавляем u в это множество);
- переход на шаг 1.

Шаг 9. (Здесь $u \in \texttt{requiritur}$ и $u \notin A$.)
Если при этом $u \notin \mathrm{pref}(A)$, то выход из алгоритма с ответом `false`.

Шаг 10. (Здесь $u \in \texttt{requiritur}$ и $u \in \mathrm{opref}(A)$.)
Выполняем следующие действия:

- $\texttt{requiritur} :- u$
  (т. е. удаляем u из этого множества);
- если $u \in A$, то $\texttt{probabile} :+ u$
  (т. е. добавляем u в это множество);
- иначе (т. е. если $u \notin A$) для каждого $v \in B$

$$\texttt{requiritur} :+ u \cdot v$$

  (т. е. добавляем в это множество каждое из слов $u \cdot v$) [14];
- переход на шаг 2 [15].

Шаг 11. (Здесь $u \in \texttt{probabile}$.)
Выполняем следующие действия:

- для каждого $v \in B$ добавляем

$$\texttt{probabile} :+ u \cdot v$$

  (т. е. добавляем в это множество каждое из не входящих в `requiritur` слов $u \cdot v$);
- переход на шаг 2 [16].

Шаг 12. Выход из алгоритма с ответом

$$\texttt{probabile} \supseteq A$$

(т. е. `true` будет ответом тогда и только тогда, когда все слова исходного языка A помечены как `probabile`).

Конец описания алгоритма.

III. ДОКАЗАТЕЛЬСТВО КОРРЕКТНОСТИ АЛГОРИТМА

Утверждение 1: На любом этапе выполнения алгоритма II.1 для любого входящего в множество `requiritur` слова $u \in \Sigma^*$ выполнено следующее:

- $u \in B^*$ [17];
- при наличии нетривиальных представлений u в виде конкатенаций слов из B^* – т. е. представлений вида

$$\begin{gathered} u = v_1 v_2 \dots v_k, \\ \text{где } v_1, v_2, \dots, v_k \in B, \quad \text{причём } k \geqslant 2, \end{gathered} \quad (1)$$

  среди них существует по крайней мере одно такое представление (пусть далее именно оно и обозначено (1)), для которого выполнено следующее:

$$(\forall j : 1 \leqslant j < k)\ (v_1 v_2 \dots v_j \notin A).$$

[14] Понятно, что такое добавление – *основное действие*, причём, по-видимому, не только шага 10, но и всего описываемого алгоритма.

По этой же причине (описывается основное действие) мы не стали разделять этот шаг на несколько – а использовали запись, содержащую подшаги. Аналогично – на рис. 1.

[15] В рассматриваемой здесь ситуации можно строго доказать, что $\texttt{requiritur} \neq \varnothing$, поэтому возвращаться на шаг 1 необязательно.

[16] См. предыдущую сноску.

[17] Отметим, что с формальной точки зрения этот пункт опускать нельзя – поскольку нетривиальных представлений (см. следующий пункт) может вообще не существовать.

Доказательство. Вначале заметим, что «для удобства использования» мы сформулировали утверждение для подобных *нетривиальных* представлений слова – то есть входящие в множество `requiritur`

- как слово ε,
- так и слова множества B

в строгом и полном доказательстве требуют особого рассмотрения.

Результат этого рассмотрения является *базой индукции* – которую, как несложно видеть, мы можем фактически провести не по словам языка Σ^*, а *по словам языка* Δ^* [18]. А *шаг индукции* – это шаг 10 алгоритма: поскольку слова в множество `requiritur` добавляются («конкатенируются») *только на нём* и только для $u \notin A$, то и условие, аналогичное (1), будет выполняться для большего значения k (т. е. $k+1$ вместо k).

Конец доказательства утверждения.

Утверждение 2: Если выполнено [19] проверяемое в алгоритме условие $A \in \mathfrak{mp}^+(B)$, то на любом этапе выполнения для любого входящего в множество `requiritur` слова $u \in \Sigma^*$ существует некоторый язык $Q \in \mathfrak{mp}(B)$, такой что

$$\{u\} \cdot Q \subseteq A.$$

Доказательство также проводится индукцией по длине слова в (подразумеваемом) алфавите Δ. *База индукции* – это слово ε над алфавитом Δ, соответствующее его морфическому образу, слову ε над алфавитом Σ. Для ε требуемое условие выполняется согласно формулировки утверждения, в этом случае требуемый язык Q – это то подмножество $A \in \mathfrak{mp}^+(\Sigma)$, которое входит в множество языков $\mathfrak{mp}(\Sigma)$ (а такое подмножество имеется по определению множества языков $\mathfrak{mp}^+(\Sigma)$).

А *шаг индукции* следует из:

- несложного свойства максимальных префиксных кодов, заключающегося в том, что множество суффиксов любого слова, являющегося собственным префиксом какого-либо сло́ва, входящего в максимальных префиксный код [20], само образует максимальных префиксный код; выполнение этого свойства следует, например, из теории, приведённой в [12], [13] [21];
- а также описания шага 10 рассматриваемого алгоритма: на этом шаге мы заменяем слово, дописывая все возможные элементы языка B, т. е., с точки зрения инверсного морфизма и алфавита Δ, рассматриваем все варианты сло́ва – прообраза u (входящего в рассматриваемый максимальный префиксный код, элемент множества языков $\mathfrak{mp}(\Delta)$) с приписанными справа всеми буквами из Δ.

[18] Немного точнее – по их морфическим образам в Σ^*.

[19] Может, точнее сказать «если *будет* выполнено»? Ведь алгоритм описывается именно для проверки этого условия.

[20] Конечно же, нескольких таких слов – при мощности алфавита $|\Delta| \geqslant 2$.

[21] То, что префикс является собственным, – не является обязательным с точки зрения этой теории, однако важно для наших построений.

А возможность рассматривать в данном случае именно собственные префиксы можно считать следствием утверждения 1: *хотя бы одно* представления рассматриваемого слова u в виде конкатенации слов языка B не имеет в качестве собственного префикса – причём такого, который совпадает с конкатенацией меньшего числа этих слов, – некоторого слова «проверяемого» языка A.

(При этом важно отметить, что *требуемый* прообраз рассматриваемого слова u существует – например, вследствие утверждения 1.)

Конец доказательства утверждения.

Утверждение 3: На любом этапе выполнения алгоритма II.1 для любого входящего в множество `probabile` слова $u \in \Sigma^*$ выполнено следующее [22]:

- $u \in B^*$;
- для каждого из представлений u в виде конкатенаций слов из B^* – т. е. представления вида

$$u = v_1 v_2 \dots v_k, \quad \text{где } v_1, v_2, \dots, v_k \in B \ -$$

выполнено следующее:

$$(\exists j : 1 \leqslant j \leqslant k)\ (v_1 v_2 \dots v_j \in A). \qquad (2)$$

Доказательство. Рассмотрим ситуации добавления слов в множество `probabile`.

Во-первых, «перевод» слов в множество `probabile` из множества `requiritur` осуществляется (только) на шагах 8 и 10 – и в обеих этих ситуациях условие $u \in A$ даёт приведённое в формулировке утверждения представление для `probabile`, позволяющее считать выполненным условие (2).

Во-вторых, собственно добавления слов в множество `probabile` (без описанного выше «перевода») осуществляется только на шаге 11; он выполняется для $u \in$ `probabile`, что, также на основе формулировки утверждения, позволяет считать выполненным условие (2) уже для каждого слова вида $u \cdot v$.

Конец доказательства утверждения.

Утверждение 4: Алгоритм II.1 – полиномиально-временной относительно общей суммарной длины слов языка $A \cup B$.

Схема доказательства. Каждый из описанных выше шагов алгоритма (и их подшагов) представляет собой тривиальный вспомогательный алгоритм, выполняемый за полиномиальное время [23].

Общее число шагов алгоритма не превышает числа слов языка $\mathrm{pref}(A)$ – поскольку после обработки каждое такое слово на шаге 6 заносится в множество `discursum`, и, на основе проверки на шаге 5 (в нём мы рассматриваем язык `tempore` $\setminus$ `discursum`, т. е. без слов последнего множества), повторной обработки никакого слова произведено быть не может.

При этом, как несложно убедиться на основе элементарной математики, при задании некоторого $n \in \mathbb{N}$ как общей суммарной длины слов некоторого (незаданного) языка число возможных префиксов слов этого языка не превышает n – т. е. полиномиально; это даёт полиномиальность всего алгоритма.

Конец доказательства утверждения.

Объединяя утверждения 1 – 4, мы получаем следующую теорему.

Теорема 1: Алгоритм II.1 определяет, выполнено ли условие $A \in \mathfrak{mp}^+(B)$ – для заданных конечных языков $A, B \in \Sigma^*$. Он является полиномиально-временны́м алгоритмом – относительно общей суммарной длины слов языка $A \cup B$.

Доказательство. Если условие $A \subseteq B^*$ не выполнено, то также не может быть выполнено и условие $A \subseteq$ `probabile` – и этот факт мы обнаружим на шаге 12 [24], т. е. мы обязательно выйдем из алгоритма с нужным нам отрицательным ответом. Поэтому далее будем считать, что $A \subseteq B^*$.

В процессе работы алгоритма мы рассматриваем все слова языка

$$\mathrm{pref}(A) \cap B^* \ -$$

и, следовательно, если $A \in \mathfrak{mp}^+(B)$, то и все слова языка A. Каждое из этих слов (слов языка A), согласно утверждениям 2 и 3, будет в процессе работы алгоритма включено в множество `probabile`, следовательно, мы завершим работу с ответом `true` тогда и только тогда, когда при этом не останется нерассмотренных слов в текущем варианте множества `requiritur`. Очевидно, что отсутствие подобных слов по определениям морфизма и расширенного максимального префиксного кода даёт нужное нам условие $A \in \mathfrak{mp}^+(B)$.

И, согласно утверждению 4, все эти действия выполняются за полиномиальное время относительно общей суммарной длины слов языка $A \cup B$.

Конец доказательства теоремы.

IV. ЗАКЛЮЧЕНИЕ

Среди направлений дальнейших работ приведём только два самых основных [25]

Во-первых, необходимо описать *полиномиальный* алгоритм, решающий задачу построения языка $h_A^{-1}(B)$, т. е. алгоритм, строящий язык – инверсный морфизм для некоторого заданного конечного языка. (Эта задача уже была кратко сформулирована выше, в сноске 8; она там была названа «более сложной задачей».)

Во-вторых – и это, по-видимому, более важно – среди всех возможных вариантов таких инверсных морфизмов нужно построить – причём также, по возможности, за полиномиальное время – выбираемый по некоторому «естественному» количественному критерию оптимальный инверсный морфизм. Таким критерием может быть, например, суммарная длина слов морфического образа (в приведённом во введении определении это – суммарная длина слов языка A).

Конечно же, последняя задача не является тривиальной – что может быть продемонстрировано следующим очень простым примером. Пусть рассматривается даже не «сложный язык» (т. е. язык из $\mathfrak{mp}^+(A)$ для некоторого непрефиксного языка A), а нужный нас язык «с двумя одновременными упрощениями»:

- $\mathfrak{mp}$ – вместо $\mathfrak{mp}^+$,

[22] Ср. с формулировкой утверждения 1.

[23] Строго в этом можно убедиться, рассмотрев соответствующие компьютерные программы. Конечно, мы этого делать не будем: во-первых, все такие программы очень просты, а во-вторых, их слишком много.

[24] Даже если туда попадём. Если же не попадём – то всё очевидно, будет необходимый ответ `false`.

[25] Напомним, что в ближайшее время мы предполагаем опубликовать специальную статью, посвящённую именно мотивации к исследованию рассмотренных здесь задач.

- Σ (как множество 1-буквенных слов, а не как алфавит) – вместо некоторого, вообще говоря непрефиксного, $A \subseteq \Sigma^*$,

т. е. просто язык из множества $\mathfrak{mp}(\Sigma)$. Например – рассмотрим язык

$$\{a, ba, bb\} \quad \text{для} \quad \Sigma = \{a, b\}.$$

Тогда кроме очевидного морфизма, соответствующего максимальному префиксному коду

$$\{0, 10, 11\} \quad \text{над} \quad \Delta = \{0, 1\},$$
$$\text{где} \quad h(0) = a,\ h(1) = b,$$

можно рассматривать и морфизм

$$\{0, 1, 2\} \quad \text{над} \quad \Delta = \{0, 1, 2\},$$
$$\text{где} \quad h(0) = a,\ h(1) = ba,\ h(2) = bb;$$

конечно же, рассматривая инверсные морфизмы, построенные для этих морфизмов, в обоих случаях обозначенных h, второй вариант вряд ли можно назвать оптимальным.

Сто́ит также отметить, что вторую из сформулированных задач действительно можно рассматривать как отдельную – а не только как продолжение первой: для каждого возможного выбранного критерия необходимо описать соответствующий алгоритм выбора такого морфизма. (В [6] было отмечено, что все «естественные» критерии дают практически одинаковые частичные порядки – но, конечно, разные критерии могут давать разные алгоритмы выбора морфизма.)

Понятно, что эти задачи – существенно более сложные, чем задача, рассмотренная в настоящей статье: язык, обычно обозначаемый B, в них *не задан*, и требуется, среди прочего, его построить. И важно отметить, что краткие варианты формулировок этих задач, а также некоторый подходы к их решению, были приведены в трёх частях статьи [6], [7], [8].

Кроме этого, нужно описать полиномиальный алгоритм, решающий задачу построения языка $h_A^{-1}(B)$, т. е. алгоритм, строящий язык – инверсный морфизм для некоторого заданного конечного языка. (Эта задача была сформулирована выше в сноске 8, где названа «более сложной».)

А в самом конце описания направлений дальнейших работ напомним приведённый выше текст, связанный с автоматом PRI. Мы отмечали, что он определён на состояниях, являющихся подмножествами множества префиксов второго из заданных для его определения языков (обычно обозначавшегося в наших предыдущих публикациях B). При этом в сноске мы написали, что возможен также вариант, когда вместо множества префиксов используется множество суффиксов (этого же языка). К более подробному рассмотрению этих вопросов мы предполагаем вернуться в одной из наших ближайших публикаций. А именно, расматривая все возможные варианты первого из заданных для определения автомата PRI языков[26], можно свести большинство рассматривавшихся нами задач к таким ситуациям, когда оба заданных для определения автомата PRI языка (A и B) совпадают. Отметим, что в примерах предыдущих статей мы уже рассматривали несколько таких ситуаций ([10] и др.).

[26] В наших предыдущих публикациях он обычно обозначался A.

При этом важно отметить, что слова такого языка A можно ограничить по длине – возможность такого ограничения следует из приведённого в наших предыдущих публикациях описания функции переходов автомата PRI.

Список литературы

[1] 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.

[2] 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.

[3] Мельников Б. *Варианты конечных автоматов, соответствующих бесконечным итерационным деревьям морфизмов. Часть I* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 7. – P. 5–13.

[4] Мельников Б. *Варианты конечных автоматов, соответствующих бесконечным итерационным деревьям морфизмов. Часть II* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 10. – P. 1–8.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Мельников Б. *Подклассы класса контекстно-свободных языков (монография)*. – М., Изд-во Московского университета. – 1995. – 174 с. – ISBN 5-211-03448-1.

[10] Абрамян М., Мельников Б. *Алгоритмы преобразования конечных автоматов, соответствующих бесконечным итерационным деревьям* // Современные информационные технологии и ИТ-образование. – 2021. – Том 17. № 1. – С. 13–23.

[11] 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.

[12] Лаллеман Ж. *Полугруппы и комбинаторные приложения*. – М., Мир. – 1985. – 440 с.

[13] Грэхем Р., Кнут Д., Паташник О. *Конкретная математика. Математические основы информатики*. – М., Вильямс. – 2009. – 784 с.

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.

Александра Александровна МЕЛЬНИКОВА,
доцент Димитровградского инженерно-технологического института – филиала Национального исследовательского ядерного университета «МИФИ»
(https://diti-mephi.ru/),
email: super-avahi@yandex.ru,
mathnet.ru: personid=148963,
elibrary.ru: authorid=143351,
scopus.com: authorId=6603567251,
ORCID: orcidID=0000-0002-1658-6857.

A polynomial algorithm for checking the fulfillment of the condition of the morphic image of the extended maximal prefix code

Boris Melnikov, Aleksandra Melnikova

***Abstract*—The maximum prefix code is defined in the usual way, "based on the things stated in the student courses". An extended maximum prefix code is a finite language containing some maximum prefix code as a subset (proper or improper one). Also the (homo)morphism is defined in the usual way, and, based on it, the inverse morphism does.**

In our previous publications, infinite iterations of finite languages as om–languages have been considered. A hypothesis was formulated that infinite iterations of two given finite languages coincide if and only if both of these languages can be obtained using the following algorithm. First, some new alphabet is selected; secondly, some two extended maximal prefix codes are considered over this alphabet; third, to these two extended maximal prefix codes, some morphism is applied (the same in both cases), which translates words over the new alphabet into words over the original alphabet. At the same time, the obtained morphic images of the two considered extended maximal prefix codes should coincide with the two given finite languages.

(More precisely, some equivalent versions of this hypothesis have been formulated, as well as another hypothesis, which has a less strong statement, but which makes sense to talk about if the first hypothesis is not fulfilled.)

This paper solves the problem of checking whether one language can be obtained using a similar algorithm applied to some other language. More precisely, a non-deterministic algorithm for such a problem is trivial, and was given in one of our previous publications; here we also give a deterministic polynomial algorithm for checking the fulfillment of this condition. Thus, the problem considered in this paper can be considered a step towards verifying the formulated hypothesis.

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

References

[1] 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).

[2] 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).

[3] Melnikov B *Variants of finite automata corresponding to infinite iterative morphism trees. Part I* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 7. – P. 5–13 (in Russian).

[4] Melnikov B *Variants of finite automata corresponding to infinite iterative morphism trees. Part II* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 10. – P. 1–8 (in Russian).

[5] 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.

[6] *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).

[7] 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).

[8] 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).

[9] 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).

[10] Abramyan M., Melnikov B. *Algorithms of transformation of finite automata, corresponding to infinite iterative trees* // Modern information technologies and IT education. – 2021. – Vol. 17. No. 1. – P. 13–23. (in Russian).

[11] 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).

[12] Lallement G. *Semigroups and Combinatorial Applications*. – NJ, Wiley & Sons, Inc. – 1979. – 376 p.

[13] Graham R., Knuth D., Patashnik O. *Concrete Mathematics. A foundation for computer science*. – USA, Addison-Wesley Professional. – 1994. – xiv+657 p.

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.

Aleksandra MELNIKOVA,
Associated Professor of
Dimitrovgrad Engineering and Technology Institute –
Branch of National Research Nuclear University "MEPhI"
(https://diti-mephi.ru/),
email: super-avahi@yandex.ru,
mathnet.ru: personid=148963,
elibrary.ru: authorid=143351,
scopus.com: authorId=6603567251,
ORCID: orcidID=0000-0002-1658-6857.