Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть I

Б. Ф. Мельников, А. А. Мельникова

***Аннотация*—В настоящей статье мы возвращаемся к тематике, связанной с одним важным бинарным отношением на множестве формальных языков (рассматриваемом в первую очередь на множестве итераций непустых конечных языков) – отношению эквивалентности в бесконечности. Прежде всего мы рассматриваем примеры применения этого отношения (как примеры необходимости его выполнения, так и примеры использования) в различных областях теории формальных языков, дискретной математики и абстрактной алгебры. Для упрощения рассмотрения эквивалентности в бесконечности мы формулируем более простое бинарное отношение на множестве языков – отношение покрытия, двойное применение которого равносильно применению отношения эквивалентности в бесконечности. Далее мы рассматриваем алгоритм проверки выполнения отношения покрытия, после чего определяем вспомогательные объекты, используемые как для доказательства корректности этого алгоритма, так и для других задач теории формальных языков. В качестве одного из комментариев к алгоритму мы приводим соответствующую ему компьютерную программу, рассматриваем примеры её работы для конкретных входных языков, после чего формулируем определения связанных с ней объектов – в частности, определение бесконечных деревьев отношения покрытия. С их помощью мы доказываем корректность алгоритма проверки выполнения отношения покрытия, а также оцениваем сложность этого алгоритма.**

***Ключевые слова*—формальные языки, итерации языков, бинарные отношения, бесконечные деревья, алгоритмы.**

I. ВВЕДЕНИЕ

Публикацией этой статьи мы надеемся начать *цикл статей* по тематике, которую рассматривали, в основном, в 1993–2000 г.г.: [1], [2], [3], [4], [5], [6], [7] и др. В этих статьях мы рассматривали *важное отношение эквивалентности* на множестве формальных языков, а также эквивалентное ему отношение эквивалентности (фактически просто равенство) на множестве ω-языков, – и *применение свойств* этих отношений в различных областях теоретической информатики и алгебры. Однако впоследствии по этой тематике у нас практически не было публикаций: статьи [8], [9], [10], [11] являются немногочисленными исключениями и, кроме того, всё-таки не связаны с ней непосредственно. Итак, мы к этой тематике возвращаемся.

Статья получена 29 января 2021 г.

Борис Феликсович Мельников, Университет МГУ – ППИ в Шэньчжэне (bf-melnikov@yandex.ru).

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

Больше всего тема настоящей статьи связана с [5]. Ниже мы повторяем приведённый в [5] алгоритм (мы при этом несколько улучшаем формулировки и доказательства вспомогательных утверждений, а также, что важнее, исправляем замеченные опечатки). Кроме того, мы приводим иллюстрирующие алгоритм рисунки, а также соответствующую ему компьютерную программу [1]. Но, главное, мы рассматриваем тот алгоритм с новой точки зрения: приступаем к рассмотрению *бесконечных итерационных деревьев*, которые появляются в нескольких моделях теоретической информатики – причём для, казалось бы, совершенно разных объектов (среди них назовём недетерминированные конечные автоматы и бинарные отношения). Кроме [5], этот алгоритм стóит также сравнить с материалом, приведённым в [12, стр. 177], см. подробнее ниже.

Итак, в настоящей статье мы рассматриваем бинарные отношения *покрытия в бесконечности* (пишем $\overset{\subseteq}{\infty}$) [2] и *эквивалентности в бесконечности* (пишем $\overset{\sim}{\infty}$); в принципе эти отношения можно рассматривать для любых бесконечных языков – но мы обычно (и, в частности, в настоящей статье) рассматриваем их только для языков вида A^*, где A – некоторый конечный (и обычно непустой) язык. Отношение $\overset{\subseteq}{\infty}$ для языков A^* и B^* выполняется в том случае, когда для любого слова языка A^* существует некоторое слово языка B^*, такое что первое из этих слов является префиксом (возможно, несобственным) второго. Отношение $\overset{\sim}{\infty}$ выполняется, когда вдобавок к этому выполнено и «зеркальное» условие (со взаимным обменом A и B); оно, очевидно, является отношением эквивалентности.

Как следует из уже сказанного выше, основной предмет статьи – это не столько алгоритм проверки выполнения бинарного отношения покрытия в бесконечности (конкретно – выполнения условия $A^* \overset{\subseteq}{\infty} B^*$ для заданных непустых конечных языков A и B) сколько начало рассмотрения относящихся к этому алгоритму вспомогательных объектов, в частности – бесконечных деревьев отношения покрытия. Мы описываем связь таких деревьев с алгоритмом проверки отношения покрытия и

[1] При этом можно отметить следующее. По-видимому, практически всегда программа, иллюстрирующая описываемый в некоторой статье алгоритм, приводится для ещё одного пояснения того, что алгоритм действительно «может быть реализован» либо «может быть выполнен за приемлемое время». Мы же приводим свою программу для совершенно иной цели: наш алгоритм является достаточно сложным – и программа является в первую очередь *большим комментарием* к нему.

[2] Ниже иногда будем писать кратко: «отношение покрытия».

его доказательством, а также получаем оценку сложности этого алгоритма.

Структура статьи такова. В разделе II приведены некоторые причины нашего интереса к рассмотрению алгоритма проверки отношения покрытия – причём некоторые из них связаны с нашими предыдущими публикациями, а некоторые – не связаны с ними. Мы рассматриваем примеры применения как этого отношения, так и его частного случая, отношения эквивалентности в бесконечности, – причём как примеры необходимости их выполнения, так и примеры их использования в различных областях теории формальных языков, дискретной математики и абстрактной алгебры. В разделе III рассматриваются основные обозначения и некоторые нестандартные соглашения по их использованию.

В разделе IV мы описываем очень важный вспомогательный алгоритм – алгоритм проверки условия

$$A^* \overset{\subseteq}{\infty} B^*$$

В разделе V мы приводим дополнительные обозначения и пояснения, необходимые для дальнейшего доказательства корректности этого алгоритма. По-видимому, среди рассматриваемых в этом разделе объектов наиболее важным является бесконечное *дерево морфизма* – на основе которого мы далее определим более сложный объект – ещё одно бесконечное дерево. А в разделе VI мы приводим описание соответствующей компьютерной программы – вместе с результатами её выполнения для простого случая; повторим, что мы рассматриваем эту программу как *особый комментарий* к приведённому алгоритму.

Весь дальнейший материал мы включаем во вторую часть настоящей статьи. В ней в разделе VII мы рассматриваем «развитие» дерева морфизма – другое специальное бесконечное дерево; особо отметим ещё раз, что оно не является рассмотренным ранее деревом морфизма – а является более сложным объектом. Мы называем его *деревом отношения покрытия* – что полностью отражает его смысл. Также в разделе VII мы приводим и новые комментарии к основному алгоритму настоящей статьи. На основе дерева отношения покрытия мы в разделе VIII продолжаем рассмотрение соответствующей компьютерной программы – вместе с результатами её выполнения для более сложного случая, чем в разделе VI.

В разделе IX мы приводим несколько вспомогательных утверждений, после чего в разделе X *доказываем корректность* алгоритма проверки условия покрытия. В разделе XI кратко рассматриваются вопросы сложности этого алгоритма. А в заключении (раздел XII) мы кратко перечисляем полученные результаты и формулируем направления дальнейшей работы, связанные с рассмотренными в настоящей статье.

II. «МОТИВАЦИЯ»

В этом разделе мы перечислим некоторые причины нашего интереса к рассмотрению алгоритма проверки отношения покрытия – причём некоторые из них связаны с нашими предыдущими публикациями, а некоторые – не связаны с ними. Мы рассматриваем примеры применения как этого отношения, так и его частного случая, отношения эквивалентности в бесконечности, – причём как примеры необходимости их выполнения, так и примеры их использования в различных областях теории формальных языков, дискретной математики и абстрактной алгебры.

По понятным причинам мы начинаем «обсуждение мотивации» с наших публикаций (они ближе к рассматриваемой здесь тематике) – а потом рассмотрим про связанные работы других авторов.

Само отношение $\overset{\sim}{\infty}$ и некоторые его свойства мы впервые рассматривали в [2] – но ещё до того, в [1], был рассмотрен один из частных случаев этого отношения (его можно назвать «префиксным»). Кроме того, было рассмотрено применение этого частного случая при решении (также частных случаев) проблемы эквивалентности однозначных скобочных грамматик – что необходимо в некоторых задачах, полезных в системах автоматизации построения компиляторов.

Отметим по этому поводу, что в общем виде проблема эквивалентности скобочных грамматик, конечно, неразрешима – что хорошо показано, например, ещё в [13]. При этом именно с помощью простых скобочных грамматик в [13] доказывается неразрешимость проблемы эквивалентности для всего класса контекстно-свободных языков. Но у этого подкласса существуют важные подклассы – однозначные и, в качестве подкласса последнего, детерминированные КС-языки. Как было отмечено в одной из наших предыдущих публикаций, автором доказательства о *разрешимости* проблемы равенства детерминированных КС-языков – иными словами, разрешимости проблемы эквивалентности детерминированных магазинных автоматов – «был объявлен» Ж. Сьёнизерг [14], несмотря на то, что пока никто не обнаружил ошибок ни в доказательстве, приведённом в серии статей Л. Станевичене [15], [16], ни в почти завершённом доказательстве В. Мейтуса [17]. Однако про разрешимость проблемы эквивалентности в «промежуточном» классе – классе однозначных КС-языков – до сих пор не известно ничего. Одна из причин этого такая: в отличие от детерминированности, чёткого критерия однозначности нет, а возможные формулировки определения однозначности *не являются конструктивными*.

Итак, после статьи [2] мы продолжали публиковать статьи уже не по критериям выполнения отношения $\overset{\sim}{\infty}$, а по *применению* этого отношения (и этих критериев); сами варианты применения могут быть условно разделены на две примерно равные части – «алгебраическую» и «формально-языковую»[3]. К первой, «алгебраической» части, при этом относятся:

- [3] – где решены задачи, в которых сформулированы условия эквивалентности некоторых алгебраических систем;
- [6] – где рассматриваются критерии коммутирования в глобальном надмоноиде свободного моноида (супермоноиде[4]) и применение этих критериев в специальных алгебраических системах;

[3] А к «критериям выполнения», т.е. к [2], при таком делении на варианты можно добавить только статью [5].

[4] Мы считаем название «супермоноид» неудачным – но подробно это пояснять не будем (поскольку это не по теме статьи); но для упрощения текста всё-таки будем далее это название употреблять. Отметим ещё, что именно название “supermonoid” обычно употребляется в таком смысле в литературе на английском языке.

- [8] – где описываются подмоноиды супермоноида, отличные от префиксного и суффиксного, а также решаются связанные с супермоноидом специальные задачи;
- [8] – где мы впервые рассматриваем связь итераций конечных языков с конечными автоматами [5];
- [10] – где мы рассматриваем связанную задачу, извлечение корня из языка [6];

А ко второй, «формально-языковой» части, относятся:

- уже кратко прокомментированная выше статья [1];
- [4] – где специальные расширения класса КС-языков позволяют описывать реальные языки программирования, в частности, языки запросов простых реляционных баз данных; при этом на основе критериев выполнения отношения $\widetilde{\infty}$ мы в частных случаях можем решить проблему эквивалентности двух таких описаний;
- [7] – где мы специальным образом описываем некоторые грамматические структуры КС-языков, причём эти описания можно рассматривать как обобщение приведённых в [4]; при этом мы также показываем возможность применения критериев выполнения отношения $\widetilde{\infty}$;
- [11] – где мы рассматриваем как класс скобочных языков (т. е. возвращаемся к задачам из [1]), так и как класс простых скобочных языков [7].

Таким образом, дальнейшие работы по этой тематике автоматически приводят к продвижению по всем перечисленным темам.

Среди работ других авторов, связанных с этой тематикой, можно, конечно, отметить уже процитированные [14], [15], [16], [17]. Кроме этого, упомянем недавние статьи [18] (где, согласно нашей терминологии, рассматриваются варианты представления ω-слова как элемента ω-итерации конечного языка), а также [19], [20] (где, как и в нашей процитированной выше работе [10], рассматривается задача извлечение корня из языка).

III. ПРЕДВАРИТЕЛЬНЫЕ СВЕ́ДЕНИЯ (ОСНОВНЫЕ ОБОЗНАЧЕНИЯ)

В этом разделе рассматриваются основные обозначения и соглашения по их использованию; при этом нужно отметить, что некоторые из этих обозначений – нестандартные.

Близко к применявшемуся в [21], будем использовать такие соглашения про алфавиты:

- Σ – «основной» алфавит, его буквы – a, b и т. д., иногда с индексами;
- Δ – «вспомогательный» алфавит, его буквы – либо 0 (не всегда), 1, 2 и т. д., либо d_1, d_2, …, d_n.

Также согласно [21] приведём подробное определение морфизма [8]. Морфизмом далее будет называться отображение вида

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

в котором каждая буква любого слова над алфавитом Δ заменяется на некоторое заранее заданное слово над алфавитом Σ.

Специально определим *конкретный* морфизм h_A (зависящий от заданного конечного языка $A \subseteq \Sigma^*$). Для конечного языка

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

будем рассматривать конкретный алфавит

$$\Delta = d_1,\, d_2,\, \ldots,\, d_n\,;$$

соответствующий морфизм h_A задаётся следующим определением [9]:

$$h_A(d_i) = u_i \quad \text{для всех} \quad i \in \overline{1,n}.$$

Ниже для морфизмов указанного будем употреблять (явно и неявно) функцию-прообраз h^{-1} – т. н. инверсный морфизм. В одной из предыдущих публикаций мы отмечали, что инверсный морфизм не является, вообще говоря, однозначной функцией; в частности существуют слова из Σ^*, не имеющие прообраза. Однако можно рассматривать прообраз некоторого слова как язык, а также прообраз языка как объединение прообразов всех его слов – т. е. подобный подход корректен.

Подробнее это формулируется следующим образом. Несмотря на то, что для любого $u \in \Delta^*$ запись $h(u)$ согласно [21] определяет слово, для любого $v \in \Sigma^*$ запись $h^{-1}(v)$ определяет *множество слов* (язык) над алфавитом Δ:

$$h^{-1}(v) = \{u \in \Delta^* \,|\, h(u) = v\};$$

при этом определяемое множество может быть и пустым. А язык $h^{-1}(B)$ для любого $B \subseteq \Sigma^*$ определяется следующим образом:

$$h^{-1}(B) = \bigcup_{v \in B} h^{-1}(v)$$

[5] Понятно, что эта связь очевидна – мы здесь имеем в виду общие задачи для этих двух формализмов и сведе́ние задач из одного формализма к задачам из другого.

[6] По нашему мнению, положительный ответ на вопрос о существовании полиномиально-временно́го алгоритма, решающего эту задачу, есть «небольшой шаг на пути к доказательству равенства P=NP». И наоборот, отрицательный ответ на вопрос о существовании такого алгоритма – «небольшой шаг на пути к доказательству неравенства P≠NP».

[7] Казалось бы, второй из этих классов – более простой… Однако:

- во-первых, ещё из студенческих курсов теории трансляции известно, что соответствующие магазинные автоматы для второго класса языков более сложные – [13] и др.;
- во-вторых, как обычно в подобных ситуациях, бо́льшая простота в описании формализма позволяет решать более сложно формулируемые задачи.

[8] Это – ключевой термин нашей статьи; поэтому нужно отметить, что в различных математических справочниках этим термином называют близкие – но всё-таки немного разные – классы отображений. Без конкретных ссылок на источники приведём следующие два определения:

- сохраняющее структуру отображение одной математической структуры на другую того же типа;
- термин, используемый для обозначения элементов произвольной категории, играющих роль отображений множеств друг в друга, гомоморфизмов групп, колец, алгебр, непрерывных отображений топологических пространств и т. п.

Казалось бы, второе определение обобщает первое (что ожидаемо, т. к. оно относится к определениям из теории категорий) – но слов «сохраняющее структуру» в нём не хватает (при том, что, повторим, речь идёт о теории категорий); а эти слова, с нашей точки зрения, являются ключевыми. Итак, мы будем использовать только простой частный случай этих определений – согласно тексту, приведённому нами.

[9] Здесь

$$\overline{(m,n)} = \{\, m,\, m+1,\, \ldots,\, n-1,\, n \,\}$$

в случае $m \leqslant n$, либо $\varnothing$ в случае $m > n$.

(и также может быть пустым) [10].

Введём ещё несколько обозначений (некоторые из них – нестандартные), применяемых далее в настоящей статье.

Если

$$v = a_1 a_2 \dots a_n \,,$$

и при некотором $m \leqslant n$

$$u = a_1 a_2 \dots a_m$$

(допускаем возможность $m = 0$, т. е. $u = e$), то u называется префиксом слова v; это записывается

$$u \in \mathrm{pref}\,(v)\,.$$

Таким образом, $\mathrm{pref}\,(v)$ – множество всех префиксов слова v. Если $m < n$ (т.е. $u \neq v$), префикс называется собственным; этот факт будем записывать так:

$$u \in \mathrm{opref}\,(v)\,.$$

Префиксы заданной длины $k \geqslant 0$ того же слова u будем обозначать $\mathrm{pref}_k(u)$.

Бинарное отношение $\overset{\subseteq}{\infty}$, которое мы уже начали обсуждать выше, для языков A^* и B^* строго определяется так [11]:

$$A^* \overset{\subseteq}{\infty} B^* \quad \Longleftrightarrow \quad (\forall u \in A^*)\,(\exists v \in B^*)\,(u \in \mathrm{pref}\,(v)).$$

Двойное применение отношения $\overset{\subseteq}{\infty}$ даёт отношение эквивалентности $\overset{\sim}{\infty}$:

$$A^* \overset{\sim}{\infty} B^* \quad \Longleftrightarrow \quad (A^* \overset{\subseteq}{\infty} B^*) \;\;\&\;\; (B^* \overset{\subseteq}{\infty} A^*).$$

В [2] показано, что вместо этих отношений можно рассматривать включение/равенство ω-итераций:

$$A^* \overset{\subseteq}{\infty} B^* \;\Leftrightarrow\; A^\omega \subseteq B^\omega \quad \text{и} \quad A^* \overset{\sim}{\infty} B^* \;\Leftrightarrow\; A^\omega = B^\omega.$$

Ещё отметим, что ω-языки были рассмотрены в нашей недавней публикации [22] – однако в уже опубликованной части статьи ω-итерации конечных языков ещё не рассматривались.

$\mathcal{P}(A)$ – булеан (степень) множества A, т. е. множество всех его подмножеств.

$\|L\|_{\max}$ – максимальная из длин слов конечного языка L.

Ещё некоторые обозначения будут вводиться по мере необходимости.

IV. ВАЖНЫЙ ВСПОМОГАТЕЛЬНЫЙ АЛГОРИТМ: ОБОЗНАЧЕНИЯ И ОПИСАНИЕ

Итак, опишем очень важный вспомогательный алгоритм – алгоритм проверки условия

$$A^* \overset{\subseteq}{\infty} B^*.$$

Как мы уже отмечали во введении, мы в этом и нескольких следующих разделах фактически повторяем алгоритм, приведённый в [5] – но при этом:

- несколько улучшаем формулировки и доказательства вспомогательных утверждений;
- исправляем замеченные опечатки;
- приводим иллюстрирующие алгоритм рисунки (без которых, по-видимому, понять алгоритм гораздо сложнее);
- приводим соответствующую алгоритму компьютерную программу – которую сто́ит рассматривать в качестве *комментария* к алгоритму.

При описании алгоритма будем использовать естественные обозначения $:=$ («положить равным») и $:+$ («добавить»). Эти обозначения употребляются для всех используемых в алгоритме переменных: переменных-слов, переменных-множеств, переменных-функций.

Алгоритм 1:

Вход: Языки $A = \{\, u_1, \dots, u_n \,\}$ и B.

Выход: 1 – если $A^* \overset{\subseteq}{\infty} B^*$;
0 – в противном случае.

Некоторые замечания к описанию алгоритма. Введём вспомогательный алфавит

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

Слова в этом алфавите и языки над ним будут обычно иметь нижний индекс $_\Delta$. Функция

$$F : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Sigma^*)$$

определяется следующим образом:

$$F(C) = \{\, v \in \mathrm{opref}\,(B) \,|\, (\exists u \in B^*)\,(uv \in C) \,\}\,.$$

Ниже будут использоваться:

- переменные-множества L, H следующего вида:
$$L \subseteq \Delta^*\,, \quad H \subseteq \mathcal{P}(\Sigma^*)\,;$$
- переменные-функции p, s (не всюду определённые) следующего вида:
$$p \subseteq \Delta^* \times \mathcal{P}(\Sigma^*)\,, \quad s \subseteq \Delta^* \times \Delta^*\,;$$
функции, как видно из этой записи, задаются указанием множества пар (аргумент – значение) [12];
- а также переменная-слово $w \in \Delta^*$.

Если алгоритм даёт ответ 0, то будем при этом фиксировать элемент $w_\Delta \in \Delta^*$, такой что

$$h_A(w_\Delta) \notin \mathrm{pref}\,(B^*)\,.$$

Описание.

Шаг 0. $L := \varnothing$,
$H := \{\, \{e\} \,\}$,
$p(e) := \{e\}$,
функция $p(u_\Delta)$ при $u_\Delta \neq e$ и функция $s(u_\Delta)$ при любом u_Δ не определены.

Шаг 1. Выбрать какое-либо слово из таких $u_\Delta \in \Delta^*$, что определено значение $p(u_\Delta)$ и не определены $p(u_\Delta d_i)$ при всех $i \in \overline{1,n}$.

Шаг 2. [13] Для всех $i \in \overline{1,n}$ выполнить следующие действия (будем здесь обозначать $p_i = p(u_\Delta d_i)$).

[10] В одной из наших предыдущих публикаций мы обсуждали т. н. задачу Ж.-Э. Пина: для любого морфизма h прообраз любого регулярного языка B (т.е. язык $h^{-1}(B)$) регулярен. Это связано с одним из шагов описываемого далее алгоритма.

[11] В настоящей статье мы рассматриваем толко языки вида A^*. Но и для произвольных языков подобное отношение можно определить аналогично – что, конечно, верно и для рассматриваемого далее отношения эквивалентности $\overset{\sim}{\infty}$.

[12] Этот способ достаточно часто используется в теории формальных языков. Те же самые соглашения мы будем использовать и в дальнейшем тексте.

[13] По поводу действий, описанных на этом шаге, важное замечание про инверсные морфизмы было приведено выше в разделе III.

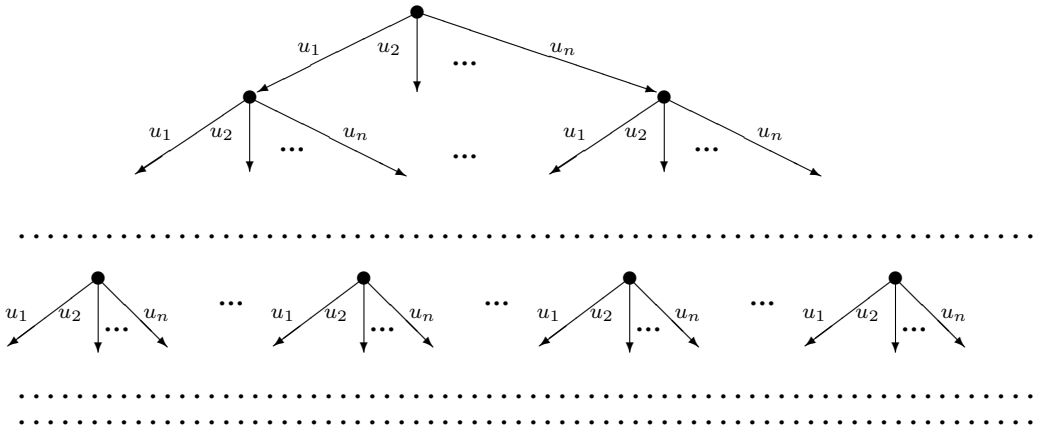

Рис. 1. Абстрактное дерево морфизма.

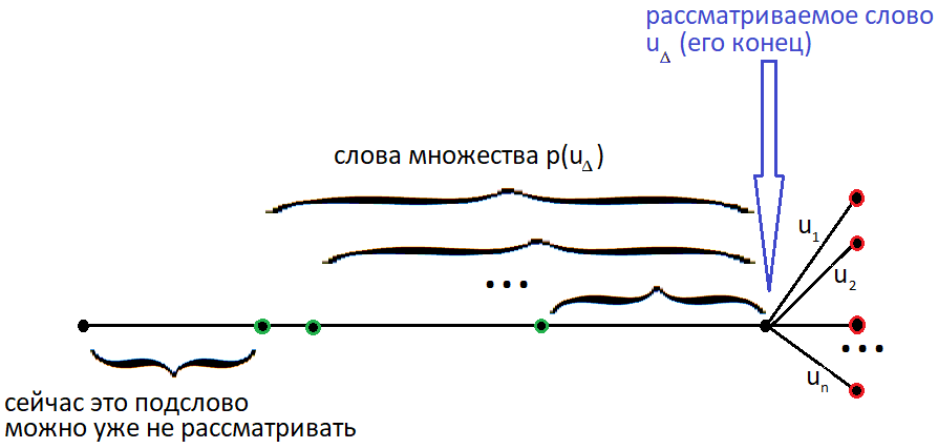

Рис. 2. Выбор слова u_Δ на шаге 1 алгоритма и модификация соответствующего языка $p(u_\Delta)$ для дальнейших действий.

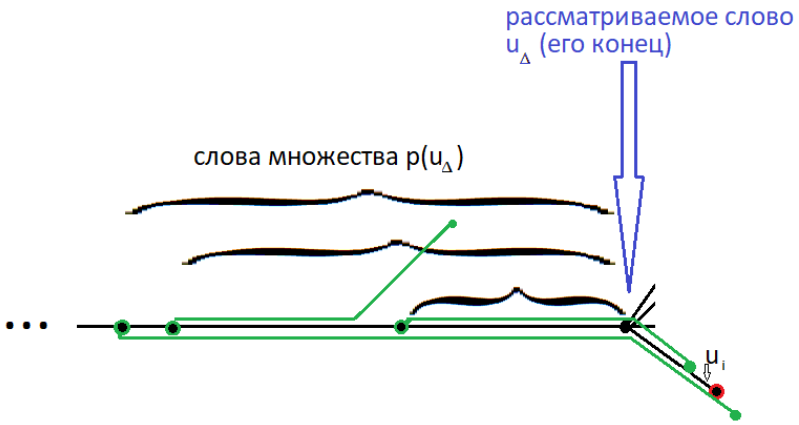

Рис. 3. Варианты продолжений слов на шаге 2 алгоритма.

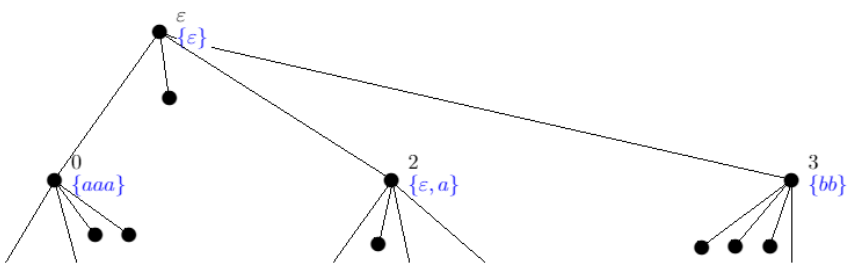

Рис. 4. Результаты работы алгоритма (начало).

Подшаги шага 2:

2.1. $p_i := F\big(p(u_{\Delta})u_i\big)$.

2.2. Если $(\exists C \in H)\,(C \subseteq p_i)$,
то $L :+ \{\, u_{\Delta} d_i \,\}$ и $s(u_{\Delta}) := p^{-1}(C)$,
иначе $H :+ p_i$ и $s(u_{\Delta} d_i) := u_{\Delta} d_i$.

2.3. Если $p_i = \varnothing$, то *выход* с ответом 0 и зафиксированным элементом $u_{\Delta} d_i$.

Шаг 3. $w := u_{\Delta}$.

Шаг 4. Если $(\forall i \in \overline{1,n})\,(w d_i \in L)$,
то $L :+ \{\, w \,\}$,
иначе *переход* на шаг 1.

Шаг 5. Если $w = e$,
то *выход* из алгоритма с ответом 1.

Шаг 6. $w := \mathrm{pref}_{|w|-1}(w)$ (т.е. мы «отрезаем» от слова w одну букву справа) и *переход* на шаг 4.

Конец описания алгоритма.

V. ВСПОМОГАТЕЛЬНЫЙ АЛГОРИТМ: ДОПОЛНИТЕЛЬНЫЕ ОБОЗНАЧЕНИЯ И ПОЯСНЕНИЯ

Предположим, что к некоторым конечным множествам $A, B \subset \Sigma^*$ применён алгоритм 1. Будем употреблять все встречающиеся в его описании обозначения и в последующем тексте. Рассмотрим также функции

$$S \subseteq \Delta^* \times \Delta^*\,, \quad P \subseteq \Delta^* \times \mathcal{P}(\Sigma^*)\,,$$

заданные следующим образом:

$$S(e) = e\,,$$

$$\big(\forall u_{\Delta} \in \Delta^*\big)\,\big(\forall d \in \Delta\big)\,\big(S(u_{\Delta} d) = s(S(u_{\Delta})d)\big)\,;$$

$$(\forall u_{\Delta} \in \Delta^*)\,(P(u_{\Delta}) = p(S(u_{\Delta})))\,.$$

Поясним смысл введённых нами объектов (переменных-множеств, переменных-функций и др.) – причём не только описанных в алгоритме 1, но и функций S и P. Для этого сначала рассмотрим *дерево морфизма*

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

т. е. бесконечное ориентированное дерево, из каждой вершины которого выходят дуги, помеченные словами языка A (см. рис. 1). Это дерево в чём-то аналогично дереву конечного автомата, рассматривавшемуся в [12, стр. 177] [14].

Будем говорить, что слово $u \in \Sigma^*$ *покрыто*, если оно является префиксом какого-либо слова из B^*. Само слово

$$w = v_1 \dots v_m$$

(где $v_i \in B$ для всех $i \in \overline{1,m}$) при этом назовём *покрытием* слова u, если

$$u \in \mathrm{pref}\,(w)\,, \quad \text{но} \quad u \notin \mathrm{pref}\,(v_1 \dots v_{m-1})\,.$$

Для некоторого u все слова v из множества $\mathrm{opref}\,(B)$, такие что слова uv являются его покрытиями, обозначим

$$F\big(\{\, u \,\}\big)$$

(такая запись употреблена вследствие того, что область определения функции F – не Σ^*, а $\mathcal{P}(\Sigma)$). Для произвольного множества C положим

$$F(C) = \bigcup_{u \in C} F\big(\{\, u \,\}\big)\,.$$

Ниже мы покажем, что условие

$$A^* \subseteq B^*$$

выполняется тогда и только тогда, когда все слова из A^* покрыты. Однако проверить, покрыто ли всё бесконечное множество слов над алфавитом Δ, мы не можем, поэтому в алгоритме неявно употребляется специальное отношение эквивалентности, заданное на Σ^* (т. е. на вершинах рассмотренного нами дерева морфизма): два слова $u, v \in \Sigma^*$ эквивалентны по этому отношению, если

$$F\big(\{\, u \,\}\big) = F\big(\{\, v \,\}\big)\,.$$

Из предыдущего следует, что для каждого класса эквивалентности по этому отношению достаточно проверить, покрыто ли какое-нибудь одно слово из рассматриваемого класса, – это и делается в алгоритме.

Продолжим описание обозначений. L – множество тех слов из Δ^*, для которых уже проверено, покрыты ли слова $h_A(L)$. H – совокупность множеств $F\big(\{\, u \,\}\big)$ для всех $u \in h_A(L)$.

Равенство $s(u_{\Delta}) = v_{\Delta}$ означает, что слова $h_A(u_{\Delta})$ и $h_A(v_{\Delta})$ принадлежат одному классу эквивалентности по описанному выше отношению. В $p(u_{\Delta})$ мы записываем некоторое множество, содержащее

$$F\big(h_A(u_{\Delta})\big)$$

в качестве подмножества, причём такое, что оно само является значением функции

$$F\big(h_A(v_{\Delta})\big)$$

для некоторого другого слова $v_{\Delta} \in \Delta^*$.

Ниже для рассматриваемого в некоторый момент работы алгоритма слова u_{Δ} мы для языка $p(u_{\Delta})$ иногда будем употреблять неформальное название «множества хвостов»:

- на рис. 2 – это слова, отмеченные «лежащими» фигурными скобками; при этом начала этих слов отмечены там зелёными кружка́ми.
- то же самое на рис. 4 (начало работы алгоритма): там это языки (множества), показанные синим цветом.

Заглавные буквы S и P обозначают функции, аналогичные обозначаемым соответствующими строчными, но определённые всюду, где это возможно, а не только для слов u_{Δ}, обрабатываемых описанным алгоритмом. И, конечно, в процессе работы алгоритма для всех возможных $u_{\Delta} \in \Delta^*$ функции S и P не строятся – но приводятся доказательства существования их и возможности построить их значения для любого конкретного $u_{\Delta} \in \Delta^*$, такого, что слово $h(u_{\Delta})$ покрыто.

VI. ОПИСАНИЕ СООТВЕТСТВУЮЩЕЙ КОМПЬЮТЕРНОЙ ПРОГРАММЫ

Программа выполнена на языке Си++. Сначала (сразу после рисунков с текстом программы и результатами её работы) приведём к ней общие замечания.

[14] Как было отмечено во введении, во второй части настоящей статьи будет рассмотрено некоторое обобщение дерева морфизма. Кроме того, в одной из ближайших публикаций мы собираемся рассмотреть другое специальное бесконечное дерево – ещё более сложный объект, связанный как с задачей, рассматриваемой в настоящей статье, так и с недетерминированными конечными автоматами.

```
class Language {
protected:
    int nWords;
    string Words[maxWords]; // каждое слово образовано из букв 'a' и 'b'
public:
    Language() { nWords = 0; }
    void AddWord(string s);
        // при добавлении слова проверяем несовпадение с уже имеющимися
    int KolWords() { return nWords; }
    string operator[] (int i) { return i<nWords ? Words[i] : "*ERR*"; }
private:
    void DeleteWord(int N);
public:
    void ConcatWord(string S);
        // добавляем слово S ко всем словам языка A (над алфавитом Sigma) -
        // в конец каждого слова
    void EraseSleva(Language& lB); // шаг 1 алгоритма
    void DeleteDaleko(Language& lB);
        // оставляем те слова, которые являются собственными префиксами слова из B
    friend ostream& operator<<(ostream& os, Language& l);
};
```

Рис. 5. Класс для работы с конечным языком (множеством слов) над алфавитом Σ.

```
void Language::EraseSleva(Language& lB) {
    for (int i=0; i<nWords; i++) {
    // цикл по словам "основного" языка (A) - от которого "отрезаем"
        string sOld = Words[i];
        for (int j=0; j<lB.KolWords(); j++) { // по словам "отрезаемого" языка B
            string sErase = lB[j];
            string sNew;
            if (!::EraseSleva(sOld,sErase,sNew)) continue;
            AddWord(sNew);
        }
    }
}
```

Рис. 6. Метод для «отрезания начала» у всех слов конечного языка. (Про применение `continue` см. замечание в тексте статьи.)

```
void Language::DeleteDaleko(Language& lB) {
    for (int i=nWords-1; i>=0; i--) {
    // цикл по словам "основного" языка (A) - из которого удаляем
        bool bDel = true;
        for (int j=0; j<lB.KolWords(); j++) { // по словам второго языка (B)
            if (!OPrefix(Words[i],lB[j])) continue;
            bDel = false;
            break;
        }
        if (bDel) DeleteWord(i);
    }
}
```

Рис. 7. Метод для удаления «лишних» слов (к которым ничего нельзя приписать) у конечного языка.

```
class Tails: public Language {
private:
    string sDelta;
        // слово из алфавита Delta, состоящего из букв 0,1,...N-1
        // (N - количество слов в A)
public:
    Tails() : Language() { sDelta = ""; }
    void InitEmpty() { sDelta = ""; nWords = 1; Words[0] = ""; }
    void ConcatLetterByInt (int I); // добавляем к слову из Delta
    friend ostream& operator<<(ostream& os, Tails& t);
};
```

Рис. 8. Класс для работы с конечным множеством слов, «предваряющими» морфизм заданного слова над алфавитом Δ («хвостов»).

```
class TailsArr {
private:
    int nTails;
    Tails tSets[maxTails];
    friend ostream& operator<<(ostream& os, TailsArr& ta);
public:
    TailsArr() { nTails = 1; tSets[0].InitEmpty(); }
    void ConcatSimple(Language& lA, Language& lB);
};
```

Рис. 9. Класс – массив объектов предыдущего класса (множество «множеств хвостов», рассматриваемое в даный момент).

```
void TailsArr::ConcatSimple(Language& lA, Language& lB) {
    if (nTails<=0) return;
    Tails tExtract = tSets[0];
    for (int i=0; i<=nTails-2; i++) tSets[i] = tSets[i+1];
    int nTailsOld = --nTails;
    if (tExtract.KolWords()<=0) return;
    for (int i=0; i<lA.KolWords(); i++) {
        tSets[nTails] = tExtract;
        tSets[nTails].ConcatLetterByInt(i);
        tSets[nTails].ConcatWord(lA[i]);
        tSets[nTails].EraseSleva(lB);
        nTails++;
    }
    for (int i=nTailsOld; i<nTails; i++) tSets[i].DeleteDaleko(lB);
}
```

Рис. 10. Метод для изъятия элемента («множества хвостов») из начала массива и добавления в конец слова `sDelta`, соответствующего этому элементу, всех вариантов букв из Δ – со всеми необходимыми действиями для соответствующих полученным словам «множествам хвостов».

```
    TailsArr tata; cout << tata << endl;
    Language lA; lA.AddWord("aaa"); lA.AddWord("aabba"); lA.AddWord("abba");
        lA.AddWord("bb"); cout << lA << endl;
    Language lB; lB.AddWord("aaaa"); lB.AddWord("abb"); lB.AddWord("abba");
        lB.AddWord("bbb"); cout << lB << endl;
    tata.ConcatSimple(lA,lB); cout << tata << endl;
    tata.ConcatSimple(lA,lB); cout << tata << endl;
```

Рис. 11. Часть текста функции `main()`, в которой задаются данные и вызываются основные методы.

```
1  :|*
2  aaa|aabba|abba|bb|
3  aaaa|abb|abba|bbb|
4  0:aaa|* 1:* 2:a||* 3:bb|*
5  1:* 2:a||* 3:bb|* 00:aa|* 01:a||* 02:* 03:*
6  2:a||* 3:bb|* 00:aa|* 01:a||* 02:* 03:* 10:* 11:* 12:* 13:*
7  3:bb|* 00:aa|* 01:a||* 02:* 03:* 10:* 11:* 12:* 13:* 20:aaa||* 21:* 22:a||* 23:abb|bb||*
8  00:aa|* 01:a||* 02:* 03:* 10:* 11:* 12:* 13:* 20:aaa||* 21:* 22:a||* 23:abb|bb||* 30:* 31:* 32:* 33:b|*
```

Рис. 12. Результаты работы алгоритма (начало выдачи программы). $\Sigma = \{a, b\}$, $\Delta = \{0, 1, 2, 3\}$. Ниже – комментарии (с номерами строк, относящихся к результатам работы программы):

1) множество $\{e\}$, самое начало работы, «множество хвостов» для пустого слова (специально отметим следующее: во-первых, это – «пустое слово над алфавитом Δ»; во-вторых, представления пустого множества $\varnothing$ и множества $\{e\}$ различны); здесь и далее (начиная с 4-й строки) выдаётся не язык, а наследник этого класса, «множество хвостов» – и в нашей выдаче множество кончается символом * (а каждое слово, в том числе e, кончается символом |);
2) исходное множество слов A (как язык, а не как «множество хвостов», его наследник); каждое слово кончается символом |;
3) аналогично для языка B;
4) «множество хвостов», получающееся после перехода от пустого (самого первого рассматриваемого) слова по первой букве; слову 1 (как элементу Δ^*) соответствует пустое множество – что видно из того, что в выдаче символов | нет; кстати наличие этого пустого множества (хотя бы одного) уже говорит о том, что алгоритм на выходе даст 0 («ложь») – но в настоящей статье это не самое интересное, нам важна работа алгоритма; именно поэтому мы считам, что перебор слов языка Δ^* происходит в том порядке, который удобен для изложения;
5) идёт обычный поиск в глубину: мы удаляем первое слово – и вместо него в конец рассматриваемого списка заносим его же с приписанными буквами (точнее, с морфизмами этих букв);
6) аналогично;
7) аналогично ещё раз; отметим, что для слова 23 (над Δ^*) первый раз появляется такое множество из по крайней мере 2 слов, в котором оба слова непустые;
8) и ещё одна итерация; отметим заранее (рассмотрим подробнее в части II), что для того, чтобы сделать такое количество итераций, которое «дойдёт» *до обработки* такого множества из 2 слов, нам нужно ещё 12 таких шагов; несмотря на то, что больше половины из таких шагов работают с пустыми множествами (и, следовательно, являются вырожденными) – мы их приводить не будем: как мы уже отмечали, уже сейчас видно, что алгоритм даёт ответ 0 («ложь», см. выше) – и, главное, понятна его дальнейшая работа.

- Повторим ещё раз, что мы рассматриваем программу как «большой комментарий» к приведённому выше алгоритму.
- Небольшое (но очень важное) *отличие* описываемой программы от приведённого выше алгоритма заключается в следующем. На шаге 2.3 алгоритма мы поступаем немного по-другому: мы *не выходим* с ответом 0 (сразу не выходим), а *запоминаем*, что будет ответ 0 (а также, конечно, запоминаем в качестве ответа ещё и само рассматриваемое слово над алфавитом Δ) – но работу алгоритма при этом не останавливаем, продолжаем его выполнять. Конечно же, зацикливания при этом не будет, поскольку все возможные «множества хвостов» заведомо представляют собой конечное множество.
  Мы вернёмся к этому вопросу в части II настоящей статьи, а пока рассмотрим только начало одного из примеров – см. рис. 4 и некоторые комментарии далее. На рисунке кружка́ми-листьями изображены слова над алфавитом Δ, соответствующие ответу 0 (ложь); понятно, что появление самого первого такого слова (слова 1) даёт возможность прервать работу алгоритма – в том случае, если нам нужен только ответ (да/нет), а не все возможные получаемые в процессе работы алгоритма «множества хвостов».
- В описании алгоритма мы *заранее ограничиваем длину* рассматриваемых слов – и «отрезаем» от них по одной букве справа. В программе же мы работаем со словами, прибавляя, наоборот, по одной букве слева. Аналогично предыдущему замечанию, мы пользуемся тем, что зацикливания не будет: может существовать не более чем конечное число подмножеств «множества хвостов».
- Для обоих описанных отличий мы фактически рассматриваем *классы эквалентности* слов множества Δ^*. Мы также вернёмся к этому вопросу в части II.
- На рисунках выше описания классов и их методов приведены в том порядке, в которых их естественно располагать в едином `cpp`-файле программы, – однако комментарии начнём приводить не «с конца» (можно сказать, что это соответствовало бы описанию *принципов программирования «сверху вниз»*) и не «с начала» (это соответствовало бы описанию *принципов объектно-ориентированных технологий*) – а «с середины», а именно – начиная *с самых важных* классов.
- Многочисленные операторы `continue` – наш стиль написания программ (à la Фортран). По нашему мнению, так значительно нагляднее, чем при применении столь же многочисленных вложенных условных операторов, которые часто не позволяют видеть всю программу одновременно. Применяемый нами стиль, как правило, даёт возможность располагать текст функций на 1–2 уровнях, а не на 4–5, как «следуя заветам Н. Вирта».

Итак, ниже мы приводим дополнительные комментарии к описаниям класов и их методов.

На рис. 8 приведено описание класса для языка $p(u_{\Delta})$ («множества хвостов», оно показано «лежащими» фигурными скобками на рис. 2). Про класс-родитель `Language` будут комментарии ниже – но ведь «множество хвостов» – это не только сам язык, но и слово языка Δ^*, которому оно (множество) соответствует; это слово – поле класса, `sDelta`. Мы всюду считаем, что буквы алфавита Δ – это цифры, начиная с нуля; в нашем примере

$$\Delta = \{\, 0, 1, 2, 3 \,\}, \quad \Sigma = \{\, a, b \,\}.$$

При печати сначала выводится это слово, а затем, после знака `:`, выводится сам язык; в примерах после вывода языка мы всегда переводим строку (это важно для понимания результатов выдачи). Забегая вперёд отметим, что на рис. 12 в строке 1 – печать простейшего «множества хвостов» (соответствующего пустому слову над алфавитом Δ), а начиная со строки 4 – печать более сложных множеств.

Конструктор без параметров можно было бы не переописывать (это сделано «на всякий случай») – он нужен при заведении массива объектов этого класса. Реальная начальная инициализация проводится в методе `InitEmpty()`, его смысл, видимо, понятен: генерируется то самое простейшее «множество хвостов», которое описано в прошлом абзаце.

Метод `ConcatLetterByInt()` – самый важный в классе. Однако он очень прост[15]: мы в нём всего лишь добавляем (приписываем) букву из Δ в конец слова (буква задаётся своим номером начиная с 0). Понятно, что такие действия должны сопровождаться:

- во-первых, приписыванием слова – морфизма этой буквы в конец каждого слова «множества хвостов»;
- во-вторых, изменением этого множества: ведь некоторые слова в нём «становятся слишком длинными».

Однако эти действия мы реализовали в методах класса-родителя: мы считаем, что они относятся к языку, а не к «множеству хвостов».

Класс-родитель `Language` описан на рис. 5. Для упрощения действий, не связанных с «алгоритмической составляющей», мы в этом и ещё в одном случае рассматриваем множество как массив. По-видимому, почти всё (кроме самого сложного метода `DeleteDaleko()`) понятно из названий методов и приведённых на рисунке комментариев к ним – добавим лишь небольшие дополнительные комментарии для двух самых важных методов, которые были «анонсированы» в предыдущем абзаце.

- Метод `ConcatWord()` добавляет строку (морфизм требуемой буквы, добавляемая строка передаётся как параметр) в конец каждого слова языка.
- А метод `EraseSleva()` (рис. 6) пытается всеми возможными способами «отрезать начало» от слов языка[16]; в описании алгоритма такого действия нет, поскольку, как мы уже отмечали, мы постепенно прибавляем к рассматриваемым словам над алфавитом Δ по одной букве справа (и производим соответствующие добавления к словам рассматриваемого языка над алфавитом Σ) – в результате чего слова становятся «слишком длинными». Эти «отрезаемые начала» являются словами второго языка (в описании алгоритма – B). После такого «обрезания»

[15] И мы в таких случаях текст реализации приводить не будем.

[16] Это действие выполняется простейшей одноимённой внешней функцией, `::EraseSleva()`.

полученные слова добавляются в рассматриваемый язык.

Важно отметить, что как последний метод, так и «внешние алгоритмы» используют такой вариант добавления слова в язык (метод `AddWord()`), который не допускает дублирования слов в языке.

Перейдём к самому сложному метода класса, методу `DeleteDaleko()`, рис. 7. Для каждого рассматриваемого слова языка[17] мы решаем, возможна ли одна из тех двух ситуаций на рис. 3, в которых конец «зелёного слова» находится либо до конца «красного», либо после его конца – т. е. выполняется ли одна из двух ситуаций, в которых помеченные зелёными кружка́ми концы слов лежат внизу. А если это не так (т. е. если конец слова можно изобразить как зелёный кружок на отрезке, направленном на рисунке наверх), то мы это слово готовим к удалению из рассматриваемого «множества хвостов». Само удаление также производится в этом методе.

Рис. 9 – класс для работы с множеством «множеств хвостов»; как и ранее, для упрощения записи, не являющемся упрощением алгоритма, мы множество представляем в виде массива. Про единственный важный метод этого класса `ConcatSimple()`[18] сказано в подробной подписи к соответствующему рис. 10.

На рис. 11 приведена часть текста функции `main()`, в которой задаются данные и вызываются основные методы: мы по одной букве увеличиваем слова над алфавитом Δ и производим требуемые изменения в соответствующих им «множествах хвостов». В части II мы рассмотрим некоторое усложнение алгоритма, дающее возможность выполнять эти действия автоматически – до тех пор, пока множество «множеств хвостов» не перестанет пополняться.

Краткие результаты *начала* выполнения программы для двух заданных языков A и B приведены на рис. 12; соответствующее дерево выполнения приведено ранее, на рис. 4. Подпись к рис. 12 очень подробная, она, в частности, включает и конкретные слова исходных языков. Специально ещё раз отметим следующее:

- каждое слово заканчивается символом `|`, а каждое множество слов – символом `*` (плюс, в нашем случае, переводом строки);
- мы специально включили в оба исходных множества по одинаковому слову *abba*;
- кружка́ми-листьями изображены слова над алфавитом Δ, соответствующие ответу 0 (ложь).

Список литературы

[1] Мельников Б. *Некоторые следствия условия эквивалентности однозначных скобочных грамматик* // Вестник Московского университета, серия 15 («Вычислительная математика и кибернетика»). – 1991. – № 10. – С. 51–53.

[2] 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.

[3] Melnikov B. *Some equivalence problems for free monoids and for subclasses of the CF-grammars class* // Number theoretic and algebraic methods in computer science, World Sci Publ. – 1995. – P. 125–137.

[4] Дубасова О., Мельников Б. *Об одном расширении класса контекстно-свободных языков* // Программирование (РАН). – 1995. – № 6. – С. 46–58.

[5] Мельников Б. *Алгоритм проверки равенства бесконечных итераций конечных языков* // Вестник Московского университета, серия 15 («Вычислительная математика и кибернетика»). – 1996. – № 4. – С. 49–54.

[6] Brosalina A., Melnikov B. *Commutation in global supermonoid of free monoids* // Informatica (Lithuanian Academy of Sciences). – 2000. – Vol. 11. No. 4. – P. 353–370.

[7] Melnikov B., Kashlakova E. *Some grammatical structures of programming languages as simple bracketed languages* // Informatica (Lithuanian Academy of Sciences). – 2000. – Vol. 11. No. 4. – P. 441–454.

[8] Мельников Б. *Описание специальных подмоноидов глобального надмоноида свободного моноида* // Известия высших учебных заведений. Математика. – 2004. – № 3. – С. 46–56.

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

[10] 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.

[11] Melnikov B., Melnikova A. *The problem of equality in the class of bracketed languages and its use in automation systems for building compilers* // IOP Conference Series: Materials Science and Engineering. – 2020. – No. 919 (5). – 052061.

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

[13] Ахо А., Ульман Дж. *Теория синтаксического анализа, перевода и компиляции. Том 1. Синтаксический анализ*. М., Мир. – 1978. – 612 с.

[14] Sénizergues G. *L(A) = L(B)? decidability results from complete formal systems* // Theoretical Computer Science. – 2001. – Vol. 251. No. 1–2. – P. 1–166.

[15] Станевичене Л. *Об одном средстве исследования бесконтекстных языков* // Кибернетика. – 1989. – № 4. – С. 135–136.

[16] Stanevicienė L. *D-graphs in context-free language theory* // Informatica (Lithuanian Academy of Science Ed.) – 1997. – Vol. 8. No. 1. – P. 43–56.

[17] Мейтус В. *Разрешимость проблемы эквивалентности детерминированных магазинных автоматов* // Кибернетика и системный анализ. – 1992. – № 5. – С. 20–45.

[18] Марченков С. *Об алфавитном кодировании сверхслов* // Проблемы передачи информации. – 2019. – Т. 55. Вып. 3. – С. 83–92.

[19] Horváth S., Leupold P., Lischke G. *Roots and Powers of Regular Languages* // 6th International Conference “Developments in Language Theory”, Kyoto, Japan, 2002. – Lecture Notes in Computer Science, vol. 2450, pp. 220–230. Springer, Heidelberg. – 2003. DOI: 10.1007/3-540-45005-X_19.

[20] Frei F., Hromkovic J., Karhumäki J. *Roots and Powers in Regular Languages: Recognizing Nonregular Properties by Finite Automata* // In book: A Mosaic of Computational Topics: from Classical to Novel. – 2020. – DOI: 10.3233/STAL200010.

[21] Саломаа А. *Жемчужины теории формальных языков*. – М., Мир. – 1986. – 159 с.

[22] Melnikov B., Melnikova A. *Some more on omega-finite automata and omega-regular languages. Part I: The main definitions and properties* // International Journal of Open Information Technologies. – 2020. – Vol. 8. No. 8. – P. 1–7.

Борис Феликсович МЕЛЬНИКОВ,
профессор Университета МГУ – ППИ в Шэньчжэне
(`http://szmsubit.ru/`),
email: `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`.

[17] Слова рассматриваются в цикле, при этом конец некоторого рассматриваемого слова помечен на рис. 3 красным кружком.

[18] Отметим, что только в этом методе в качестве параметров используются *оба* языка – A и B в терминологии описания алгоритма.

Infinite trees in the algorithm for checking the equivalence condition of iterations of finite languages. Part I

Boris Melnikov, Aleksandra Melnikova

***Abstract*—In this paper, we return to the topic related to one important binary relation on the set of formal languages (considered primarily on the set of iterations of nonempty finite languages), i.e. the equivalence relation at infinity. First of all, we consider examples of the application of this relation (both examples of the need for its implementation, and examples of its use) in various fields of the theory of formal languages, discrete mathematics, and abstract algebra. To simplify the consideration of equivalence at infinity we formulate a simpler binary relation over the set of languages, i.e. the covering relation, the double application of which is equivalent to the application of the equivalence relation at infinity. Next, we consider an algorithm for verifying the fulfillment of the coverage relation, and then we define auxiliary objects used both for proving the correctness of this algorithm and for other problems in the theory of formal languages. As one of the comments on the algorithm, we give the corresponding computer program, consider examples of its operation for specific input languages, after that, we formulate the definitions of the objects associated with them, in particular, the definition of infinite trees of the coverage relation. With their help we prove the correctness of the algorithm for checking the fulfillment of the coverage relation, and also estimate the complexity of this algorithm.**

***Keywords*—formal languages, iterations of languages, binary relations, infinite trees, algorithms.**

References

[1] Melnikov B. *Some consequences of the equivalence condition of unambiguous bracketed grammars* // Bulletin of the Moscow University, Series 15 ("Computational Mathematics and Cybernetics"). – 1991. – No. 10. – P. 51–53 (in Russian).

[2] 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.

[3] Melnikov B. *Some equivalence problems for free monoids and for subclasses of the CF-grammars class* // Number theoretic and algebraic methods in computer science, World Sci Publ. – 1995. – P. 125–137.

[4] Dubasova O., Melnikov B. *On an extension of the context-free language class* // Programming (Russian Academy of Sciences). – 1995. – No. 6. – P. 46–58 (in Russian).

[5] Melnikov B. *Algorithm for checking the equality of infinite iterations of finite languages* // Bulletin of the Moscow University, Series 15 ("Computational Mathematics and Cybernetics"). – 1996. – No. 4. – P. 49–54 (in Russian).

[6] Brosalina A., Melnikov B. *Commutation in global supermonoid of free monoids* // Informatica (Lithuanian Academy of Sciences). – 2000. – Vol. 11. No. 4. – P. 353–370.

[7] Melnikov B., Kashlakova E. *Some grammatical structures of programming languages as simple bracketed languages* // Informatica (Lithuanian Academy of Sciences). – 2000. – Vol. 11. No. 4. – P. 441–454.

[8] Melnikov B. *The description of special submonoids of the global supermonoid of the free monoid* // News of Higher Educational Institutions. Mathematics. – 2004. – No. 3. – P. 46–56 (in Russian).

[9] 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).

[10] 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.

[11] Melnikov B., Melnikova A. *The problem of equality in the class of bracketed languages and its use in automation systems for building compilers* // IOP Conference Series: Materials Science and Engineering. – 2020. – No. 919 (5). – 052061.

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

[13] Aho A., Ullman J. *The Theory of Parsing, Translation and Compiling. Vol. 1. Parsing*. NJ, Prentice Hall. – 1972. – 2051 p.

[14] Sénizergues G. *L(A) = L(B)? decidability results from complete formal systems* // Theoretical Computer Science. – 2001. – Vol. 251. No. 1–2. – P. 1–166.

[15] Stanevicienė L. *On a research facility without contextual languages* // Cybernetics. – 1989. – No. 4. – P. 135–136 (in Russian).

[16] Stanevicienė L. *D-graphs in context-free language theory* // Informatica (Lithuanian Academy of Science Ed.) – 1997. – Vol. 8. No. 1. – P. 43–56.

[17] Meytus V. *The solvability of the equivalence problem of deterministic pushdown automata* // Cybernetics and systems analysis. – 1992. – No. 5. – P. 20–45 (in Russian).

[18] Marchenkov S. *About alphabetic coding for superwords* // Problems of information transmission. – 2019. – Vol. 55. No. 3. – P. 275–282.

[19] Horváth S., Leupold P., Lischke G. *Roots and Powers of Regular Languages* // 6th International Conference "Developments in Language Theory", Kyoto, Japan, 2002. – Lecture Notes in Computer Science, vol. 2450, pp. 220–230. Springer, Heidelberg. – 2003. DOI: 10.1007/3-540-45005-X_19.

[20] Frei F., Hromkovic J., Karhumäki J. *Roots and Powers in Regular Languages: Recognizing Nonregular Properties by Finite Automata* // In book: A Mosaic of Computational Topics: from Classical to Novel. – 2020. – DOI: 10.3233/STAL200010.

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

[22] Melnikov B., Melnikova A. *Some more on omega-finite automata and omega-regular languages. Part I: The main definitions and properties* // International Journal of Open Information Technologies. – 2020. – Vol. 8. No. 8. – P. 1–7.

Boris MELNIKOV,
Professor of Shenzhen MSU – BIT University, China
(http://szmsubit.ru/),
email: 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.