Полиномиальный алгоритм построения конечного автомата для проверки равенства бесконечных итераций двух конечных языков

Б. Ф. Мельников, А. А. Мельникова

***Аннотация*—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. Мы сформулировали более простое бинарное отношение на множестве языков – отношение покрытия, двойное применение которого равносильно применению отношения эквивалентности в бесконечности. После этого мы рассмотрели алгоритм проверки выполнения отношения покрытия, определили вспомогательные объекты, используемые для доказательства корректности этого алгоритма.**

Одним из таких объектов бесконечное итерационное дерево. В рассмотренных бесконечных итерационных деревьях мы объединяем эквивалентные состояния – фактически получая при этом детерминированный конечный автомат. Мы определили конкретный такой автомат для двух заданных конечных языков – т. н. первичный автомат, PRI; он детерминированный, определён на множествах слов – и каждое из этих множеств является подмножеством множества префиксов второго из заданных языков.

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. упрощённый первичный автомат, NSPRI, – который также описывает построение итерационного дерева морфизма, но определён не на множествах слов, а на словах.

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 делает невозможным полиномиальность алгоритма. Поэтому мы строим недетерминированный автомат – определённый просто на множестве возможных префиксов; 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.

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

Статья получена 12 июля 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).

I. ВВЕДЕНИЕ

В настоящей статье мы продолжаем тематику статей [1], [2], [3], [4]. В них мы продолжили работу со специальным бинарным отношением на множестве формальных языков (рассматриваемом в первую очередь на множестве итераций непустых конечных языков) – т. н. отношением эквивалентности в бесконечности. Мы рассмотрели примеры применения этого отношения (причём как примеры необходимости его выполнения, так и примеры его использования) в различных областях теории формальных языков, дискретной математики и абстрактной алгебры.

Мы сформулировали более простое (чем упомянутая эквивалентность в бесконечности) бинарное отношение на множестве языков – т. н. отношение покрытия, двойное применение которого равносильно применению отношения эквивалентности в бесконечности. После этого мы рассмотрели алгоритм проверки выполнения отношения покрытия, определили вспомогательные объекты, используемые как для доказательства корректности этого алгоритма, так и для других задач теории формальных языков.

Одним из таких объектов является бесконечное дерево отношения покрытия – частный случай бесконечного итерационного дерева. С помощью таких деревьев мы доказали корректность алгоритма проверки выполнения отношения покрытия, а также оценили сложность этого алгоритма – в случае *формальной* его реализации.

В рассмотренных бесконечных итерационных деревьях мы объединили эквивалентные состояния – фактически получив при этом детерминированный конечный автомат; мы также описали некоторые свойства такого автомата. Мы описали один из *конкретных* подобных автоматов: для двух заданных конечных языков мы определили т. н. *первичный* автомат, PRI . Он, как мы уже отметили, детерминированный, определён на множествах слов – и каждое из этих множеств является подмножеством множества префиксов второго из заданных языков.

После этого мы определяем соответствующие ему несколько вариантов специальных *недетерминированных* автоматов, фактически описывающих построение этого итерационного дерева морфизма. Мы вводим совершенно иной объект – т. н. *упрощённый первичный* автомат, NSPRI , – который также описывает построение итерационного дерева морфизма, но определён не на множествах слов, а на словах. Отметим, что он «мало отличается» от автомата, который может быть получен из PRI

путём применения алгоритмов минимизации недетерминированных конечных автоматов (см. [5], [6], а также работы одного из авторов [7], [8], [9], [10] и др.) – однако подробности подобной минимизации мы в настоящей статье рассматривать не будем.

Основная идея построения полиномиального алгоритма проверки равенства бесконечных итераций двух конечных языков (это – фактически продолжение алгоритма, изложенного в [3], [4]) такова. Естественным является построение для этой задачи *детерминированного* автомата – каждое состояние которого описывает целое множество возможных префиксов («хвостов», оставшихся от различных разложений слова – морфизма итерации первого заданного конечного языка по словам второго заданного конечного языка). Однако при этом мы работаем с множеством множеств возможных префиксов – что делает невозможным полиномиальность алгоритма: просто потому, что такое множество множеств имеет, вообще говоря, экспоненциальный размер. Поэтому мы строим *недетерминированный* автомат (в [4] обозначавшийся нами NSPRI) – определённый просто на множестве возможных префиксов (а не на множестве таких множеств). Однако при этом возникает вопрос, связанный с тем, когда именно этот автомат даёт положительный результат – и подобной проблемы не было в случае детерминированного автомата PRI. Мы решаем эту проблему тем, что требуем, чтобы получающийся недетерминированный автомат являлся бы аналогом всюду определённого детерминированного автомата (“total automaton”).

Отметим сразу, что *мы не утверждаем, что саму работу полученного автомата можно описать с помощью полиномиального алгоритма* – однако не утверждаем и обратного:

- во-первых, может быть, удастся доказать, что у получаемого автомата NSPRI число состояний ограничено, причём это ограничение даст возможность организовать полиномиальные вычисления;
- во-вторых, могут существовать принципиально иные алгоритмы, связывающие конечный автомат с проверкой отношения $\leqslant$.

Предлагаемый в статье алгоритм важен и безотносительно к другим задачам – но есть и иные «пункты мотивации» для его рассмотрения, т.е. как раз возможное применение этого алгоритма как вспомогательного. В первую очередь мы имеем в виду его применение в двух схожих задачах теории формальных языков[1]:

- в задаче извлечения корня заданной (либо максимально возможной) степени из заданного языка;
- и в задаче построения оптимального инверсного морфизма.

Конечно, в обеих этих задачах важна возможность полиномиальности всех вспомогательных алгоритмов. Мы собираемся подробно рассмотреть эти задачи в следующей публикации.

Приведём содержание статьи по разделам. Раздел II (предварительные све́дения) – небольшой[2]: все необходимые обозначения были нами подробно рассмотрены ранее, и мы повторяем только самые важные из них – те, которые особенно важны для настоящей статьи. Также в этом разделе мы приводим комментарии, которые объясняют смысл состояний всех рассматривавшихся в [3], [4] конечных автоматов – как детерминированных, так и недетерминированных.

В разделе III приведено общее описание рассматриваемого в настоящей статье алгоритма построения искомого конечного автомата: указаны его вспомогательные алгоритмы (подалгоритмы) и показано, что именно нужно продемонстрировать для доказательства полиномиальности всего алгоритма построения автомата.

Далее мы переходим непосредственно к описаниям подалгоритмов – *а их полиномиальность становится понятной на основе этих описаний*. В разделе IV мы рассматриваем возможный алгоритм построения множества состояний недетерминированного автомата NSPRI. В разделе V – алгоритмы построения вспомогательных объектов, используемых нами при описании основного алгоритма. Этот основной алгоритм мы приводим в разделе VI – фактически это алгоритм построения множества дуг графа переходов автомата NSPRI (или, иными словами, возможные значения функции Ф).

В заключении (раздел VII) кратко сформулированы направления дальнейшей работы.

II. ПРЕДВАРИТЕЛЬНЫЕ СВЕ́ДЕНИЯ

Все используемые нами обозначения были подробно рассмотрены в предыдущих статьях [1], [2], [3], [4], и мы повторяем только самые важные из них, необходимые для настоящей статьи.

Если для конечных языков A и B выполнено условие

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

то мы пишем $A \leqslant B$ (либо $B \geqslant A$).

Если одновременно выполнены условия $A \leqslant B$ и $A \geqslant B$, пишем $A \lessgtr B$.

Полностью повторим также [3, опр. 7] – в нём мы определяем функцию Ф. А именно, для пары непустых конечных языков $A, B \subseteq \Sigma^*$, таких что $A \not\ni \varepsilon$ и $B \not\ni \varepsilon$, язык

$$\underset{A-B}{\Phi}(u) \subseteq \Sigma^*$$

строится для некоторого слова $u \in \Sigma^*$ по следующему алгоритму – в нём используется вспомогательный язык D' и формируется язык-ответ D.

1) Для начала работы алгоритма полагаем
$$D' = \{u\}A, \quad D = \varnothing.$$
2) Если $D' = \varnothing$, то выходим из алгоритма с ответом D.
3) Выбираем некоторое слово $v \in D'$, исключая его из D':
$$D' -: v.$$
4) Для выбранного v рассматриваем все варианты его представления в виде $v = uv'$, где $u \in B^+$,
$$D +: v'$$
(т. е. включаем каждое такое v' в формируемый язык D).

[1] Точные формулировки этих задач приведены в наших предыдущих статьях, процитированных выше.

[2] В отличие от предыдущих статей.

5) Если
$$(\exists w \in B)\,(v \in \mathrm{opref}\,(w)),$$
то $D +\!: v$ (т. е. включаем v в язык D).
6) Переходим на п. 2.

(Ещё раз отметим, что согласно приведённому алгоритму про итоговый «выходной» язык D можно утверждать, что $D \subseteq \mathrm{opref}\,(B)$. Также отметим, что в настоящей статье определение значения функции $\underset{A-B}{\Phi}(C)$ для языка $C \subseteq \Sigma^*$ не используется.)

Теперь очень кратко (и неформально) опишем возможный вариант доказательства возможности сведéния итераций: бесконечных к конечным и наоборот; этот вариант – альтернативный к приведённому в [2]. Любое ω-слово, являющееся ω-итерацией слов языка конечного A, мы выписываем как бесконечную итерацию слов из этого A. Оно (это ω-слово) в точности совпадает с ω-словом «над языком B»[3] – выпишем и его словами из B. Раз это удалось выполнить, то для любого префикса рассматриваемого ω-слова «над языком A» существует конечная итерация слов из B, для которой (итерации) это слово из A^* является собственным префиксом, – а это приводит к выполнению определения $A \trianglelefteqslant B$. Аналогично доказывается выполнение условия $B \trianglelefteqslant A$, а также обратный факт – т. е. возможность сведéния равенства бесконечных итераций к эквивалентности итераций конечных.

Теперь приведём несколько важных комментариев к нашим предыдущим статьям – в первую очередь к [3], [4]. Будем использовать терминологию и примеры из этих статей.

Во-первых, про наличие пустых слов ε во всех получающихся «множествах хвостов» – в тех случаях, когда языки A и B совпадают. Действительно, именно из-за совпадения A и B во всех формируемых множествах, описывающих эти «множества хвостов», обязательно имеется слово ε – поскольку всегда возможны одинаковые разложения и по словам языка A, и по словам языка B. Мы, конечно, при этом получаем положительный ответ на вопрос о выполнении условия $A \trianglelefteqslant B$ – однако кроме этого ответа нам часто бывает нужна и другая информация о работе алгоритма получения этого ответа: ведь вместе с этим словом ε получаются и другие слова, которые все вместе формируют *класс эквивалентности* (состояние получаемого конечного автомата). Таким образом, в нашей ситуации получение (положительного) ответа не влечёт прекращения работы алгоритма.

Во-вторых, мы всюду писали что это «множество хвостов» (т. е. также состояния автомата NSPRI, либо элементы состояний автомата NSPRI) принадлежит языку $\mathrm{opref}\,(B)$. Может показаться, что в этих названиях и комментариях содердится ошибка[4], однако на самом деле никакой ошибки нет: эти «хвосты» являются *началами новых слов* языка B, не входящих в формируемое разложение «длинного» слова – некоторого конкретного A-морфизма.

В-третьих, повторим, это действительно префиксы языка B – однако можно сказать ещё более точно: в каждом конкретном случае (т. е. для каждого состояния автомата NSPRI) мы получаем подмножество множества префиксов некоторого *одного* слова языка B. Однако этот факт вовсе «не отменяет» сделанного в двух последних разделах статьи [4] предположения о неполиномиальности: ведь в приведённом там *неформальном* объяснении этого предположения рассматривают именно одно слово языка B.

III. ОБЩЕЕ ОПИСАНИЕ АЛГОРИТМА ПОСТРОЕНИЯ ИСКОМОГО АВТОМАТА

Как мы уже отмечали во введении, алгоритм, предлагаемый в настоящей статье, является некоторой модификацией алгоритма, строящего конечные автоматы в статье [3], [4]. При этом понятно, что для проверки выполнения отношения $A \trianglelefteqslant\!\trianglerighteqslant B$ достаточно просто дважды проверить выполнение отношения $\trianglelefteqslant\!: A \trianglelefteqslant B$ и $B \trianglelefteqslant A$; таким образом, оба алгоритма либо одновременно являются, либо одновременно не являются полиномиальными. Поэтому важно отметить, что на основе формулировок алгоритмов, приведённых в наших предыдущих публикациях, про полиномиальность «говорить было рано».

На основе одного из таких алгоритмов мы в [3], [4] для заданных языков A и B определили конечный автомат $\mathrm{PRI}\,(A, B)$; однако очевидно, что любой возможный алгоритм его построения – экспоненциальный[5]. Поэтому мы переходим к недетерминированному конечному автомату NSPRI, определённому на словах (префиксах языка B) – а не на множествах таких слов. При этом возникает проблема, связанная тем, когда именно этот автомат даёт положительный результат. Мы решаем эту проблему тем, что требуем, чтобы получающийся недетерминированный автомат являлся бы аналогом всюду определённого детерминированного конечного автомата[6] – т. е. чтобы из любого его состояния для любой буквы рассматриваемого алфавита вела бы дуга (в некоторое состояние, не являющееся бесполезным). И, понятно, выполнение этого условия *можно проверить полиномиально*. Поэтому надо просто полиномиально построить сам автомат NSPRI (полиномиально относительно длин заданных языков A и B) – что мы и делаем в описываемом алгоритме.

И, конечно, для доказательства полиномиальности алгоритма построения множества состояний автомата вряд ли есть смысл пользоваться формальным описанием некоторой машины Тьюринга, решающей эту задачу, – равно как и соответствующими нормальным алгоритмом Маркова [12], машиной с неограниченными регистрами [13] и прочими возможными для этого формализмами; по-видимому, достаточно привести программу на какомлибо известном языке программирования.

При этом в принципе можно доказать полиномиальность *и на основе компьютерной программы, приве-*

[3] Ставим здесь (и далее в подобных случаях) кавычки – поскольку это понятие строго не определено, причём вряд ли есть смысл его определять. Смысл, по-видимому, хорошо ясен.

[4] «Хвосты» – но префиксы? более «логичными» были бы суффиксы?

[5] В одной из процитированных статей была приведена схема получения примера, в которой у получаемого автомата число состояний будет таким:
$$\mathrm{C}_N^{N/2};$$
здесь N – длина одного из слов заданного языка, который в этом примере совпадает как с A, так и с B.

[6] Чисто формально в литературе по теории формальных языков всюду определённый (“total”) конечный автомат обычно определяется только для детерминированного случая – однако в нашем случае, для недетерминированных автоматов, это не мешает пользоваться практически такими же определениями и алгоритмами.

дённой нами ранее, в [1], [2][7]: она фактически решала ту же самую задачу, хотя и названную совершенно иначе. Использованное в той программе объектно-ориентированное программирование позволяло, во-первых, сделать качественную программу, во-вторых, использовать её в качестве комментариев к приведённым алгоритмам … однако эту программу достаточно сложно использовать для доказательства полиномиальности реализуемого ею алгоритма.

Поэтому мы «специально ухудшаем» эту программу, не оставляя практически ничего из элементов Си++, – и получаем приведённый далее текст; практически все его конструкции транслируются и на «неоплюсевшем» Си – однако для удобства изложения формально выбран язык Си++.

Однако само окончательное построение основной требуемой функции Ф мы всё-таки опишем не в виде программы (она получается слишком громоздкой) – а в виде подробного описания алгоритма, по которому эту программу легко построить[8].

И, конечно, доказательство существования полиномиального алгоритма построения функции Ф достаточно для того, чтобы дать положительный ответ на основной вопрос статьи – о существовании полиномиального же алгоритма построения недетерминированного конечного автомата, предназначенного для проверки равенства бесконечных итераций двух конечных языков[9]. Действительно, мы показываем, что все состояния автомата строятся полиномиальным алгоритмом; мы также показываем, что число таких состояний у автомата пропорционально N^2 – для размера входных данных N. Поэтому число пар таких состояний $\sim N^4$, и наличие полиномиального алгоритма построения множество дуг между каждой такой парой (мы приводим описания алгоритма, имеющего сложность $\sim N^8$) говорит о возможности построения автомата за полиномиальное время $\sim N^{12}$. Проверка же всюдуопределённости построенного автомата пропорциональна N^2.

IV. ПОЛИНОМИАЛЬНОСТЬ АЛГОРИТМА ПОСТРОЕНИЯ МНОЖЕСТВА СОСТОЯНИЙ НЕДЕТЕРМИНИРОВАННОГО АВТОМАТА

Повторим ещё раз, что нам очень важна недетерминированность строимого автомата NSPRI: для детерминированного автомата PRI алгоритмы построения существенно проще[10] – однако с формальной точки зрения они неполиномиальны.

Итак, сначала рассмотрим построение множества состояний автомата NSPRI. Строки используем только в качестве удобной записи массива символов – не применяя функций работы со строками. Однако «совсем без процедурного программирования» (т. е. без использования функций) обойтись вряд ли получится – хотя «чистые математики», по-видимому, такое доказательство могут и раскритиковать[11].

Приведём описание данных:

```
const int N = 200;
const int NN = N*N;
const string A = "aaa|aabba|abba|bb|*";
const string B = "aaaa|abb|abba|bbb|*";
string Bpref;
int Begins[NN];
```

Некоторые комментарии про представление данных. Строки-константы `A` и `B` описывают заданные языки[12]; в них, аналогично выдаче на печать языков `lA` и `lB` в программе из [1], [2], каждое слова языка заканчивается символом `|`, а вся строка – символом `*` (сами эти символы в рассматриваемые слова не входят). В примере программы их значения – такие же, как у аналогичных строк в примерах, приведённых в [1], [2]. Будем считать, что максимально возможная длина каждой из строк `A` и `B` (включая эти специальные символы) ограничена значением заданной константы `N`.

Результат работы (всё множество собственных префиксов строки́ `B`) мы будем записывать в выходную строку `Bpref`, которая организована по тем же правилам, что и строки `A` и `B`. Важно отметить, что длина этой выходной строки заведомо не превышает N^2 – это несложно доказывается методами элементарной математики[13].

`Begins` – формируемый массив[14], куда мы будем записывать номера байтов, каждый из который является началом очередного слова в строке (массиве) `Bpref`; понятно, что имеется не более N^2 слов. После начала последнего слова запишем значение -1.

Теперь перейдём непосредственно к алгоритмам. К формируемому массиву (строке `Bpref`) следующая функция `AddStrings()` добавляет все подстроки – собственные суффиксы строки, которая неявно передаётся этой функции с помощью параметров:

```
void AddStrings(int Beg, int End) {
  for (int i=Beg; i<=End; i++) {
    for (int k=Beg; k<=i; k++)
      Bpref += B[k];
    Bpref += '|';
  }
}
```

[7] Т. е., видимо, можно сказать, что мы *уже* доказали требуемую полиномиальность алгоритма проверки равенства бесконечных итераций конечных языков – в предыдущих публикациях.

[8] И противоречия здесь нет. Действительно, по нашему мнению, хорошая объектно-ориентированная программа – это лучший комментарий к алгоритму. Однако программа, написанная *специально без использования* объектно-ориентированной технологии (написанная так для того, чтобы можно было формально оценить её сложность), – наоборот, нуждается в больших комментариях.

[9] В очередной раз отметим, что речь идёт *только о построении* этого автомата.

[10] Более того – для реальных автоматов они, по-видимому, всегда выполняются быстрее!

[11] Некоторую аналогию можно видеть в доказательстве теоремы о четырёх красках, выполненном в 1976 г.: в этом доказательстве (также до сих пор признаваемом не всеми «чистыми математиками») осуществляется компьютерный перебор 1936 карт некоторого специального множества и посредством этого перебора доказывается, что ни одна из них не содержит карту меньшего размера, которая опровергала бы сформулированное условие теоремы.

[12] В этом разделе строка `A` не используется.

[13] А при применении тривиальных переборных алгоритмов – мы про них писали во всех предыдущих публикациях, связанных с итерациями языков и / или задачей извлечения корня из языка – возникают массивы данных размерностью $\sim 2^N$. Это заведомо приводит к не меньшей временно́й сложности тех алгоритмов проверки равенства бесконечных итераций двух конечных языков, которые используют подобные массивы строк. Поэтому предмет настоящей статьи может быть кратко назван так: *не* использование подобных массивов.

[14] Подчеркнём ещё раз, что он здесь формируется – но после этого формирования используется только в следующем разделе.

(Параметр внешнего цикла `i` – начало рассматриваемой подстроки.)

Мы *не следим* за возможными повторениями включаемых слов: так алгоритм получается проще (а полиномиальным он быть не перестаёт). Нам важно, что выполнение обоих вложенных циклов состоит из не более чем N итераций – поэтому *сложность приведённого алгоритма* $\sim N^2$.

Далее, в основной функции `main()`, мы ищем слова во входной строке и применяем описанную выше функцию `AddStrings()`:

```
Bpref = "";
for (int iBeg=0; iBeg<B.size(); iBeg++) {
  for (int iEnd=iBeg+1;
       iEnd<B.size(); iEnd++) {
    if (B[iEnd]!='|') continue;
    AddStrings(iBeg,iEnd-1);
    iBeg = iEnd;
    break;
  }
}
Bpref += '*';
cout << "Answer: " << Bpref << endl;
```

Из текста функции видно, что каждый из циклов выполняется за не более чем N итераций – т. е. рассматриваемые нами алгоритмы полиномиальны.

V. ПОЛИНОМИАЛЬНОСТЬ АЛГОРИТМОВ ПОСТРОЕНИЯ ВСПОМОГАТЕЛЬНЫХ ОБЪЕКТОВ

Конечно, полиномиальным является и алгоритм формирования массива `Begins`:

```
Begins[0] = 0;
int i = 1; // новый индекс в Begins
for (int j=1; j<NN; j++) {
  if (Bpref[j]=='*') break;
  if (Bpref[j]!='|') continue;
  Begins[i++] = j+1;
}
Begins[--i] = -1;
```

А далее мы начинаем описывать построение основной функции – ранее обозначавшейся нами либо F, либо Ф, либо, согласно [3], [4], пометке дуги графа переходов автомата NSPRI. В первую очередь отметим, что аргументом этой функции (или, если использовать обозначения из [3], [4], состоянием автомата NSPRI) можно считать не только слово над вспомогательным алфавитом Δ (иногда в процитированных статьях обозначавшемся Δ_A) – но и непосредственно слово языка A (а именно – $h_A(d)$ для некоторой буквы $d \in \Delta$).

К описанным в предыдущем разделе структурам данным добавим следующие:

```
string FiTmp;
bool Nac[NN];
bool Otv[NN];
```

Небольшие комментарии:

- `FiTmp` – промежуточное представление для формирования функции F; чисто формально длина этой строки ограничена значением N^2+N (чего, конечно, достаточно для доказательства полиномиальности алгоритма) – однако средствами элементарной математики можно показать, что достаточно рассматривать значение N^2, что несколько упрощает запись алгоритмов;
- `Nac` – массив начал слов разложения рассматриваемого слова `FiTmp` по словам языка `B`; эти начала слов обозначены на приведённых ранее рисунках зелёными кружка́ми ([1, рис. 2, 3], рисунки повторены ниже);
- `Otv` – массив начал тех слов, которые пойдут в ответ (в сформированное значение функции F); концы таких слов обозначены на тех же рисунках красными кружка́ми.

Итак, повторим рисунки в малом масштабе – в новой нумерации это рис. 1 (выбор слова u_Δ на шаге 1 алгоритма, приведённого в [1], а также модификация соответствующего языка $p(u_\Delta)$ для дальнейших действий) и рис. 2 (варианты продолжений слов на шаге 2 этого алгоритма).

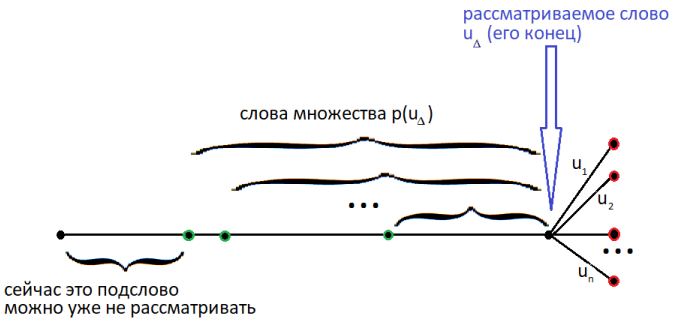

Рис. 1. Выбор слова u_Δ …

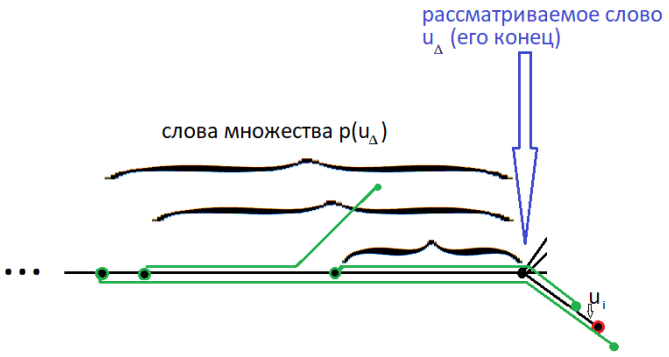

Рис. 2. Варианты продолжений слов на шаге 2 алгоритма.

Следующая функция формирует слово `FiTmp` – путём дописывания к каждому слову, входящему в `Bpref`, некоторого (одного) слова языка A, заданного своими началом и концом в строке `A`.

```
void AddToBpref(int Beg, int End) {
  FiTmp = "";
  for (int i=0; ; i++) {
    char ch = Bpref[i];
    if (ch=='*') break;
    if (ch=='|')
      for (int j=Beg; j<=End; j++)
        FiTmp += A[j];
    FiTmp += ch;
  }
  FiTmp += '*';
```

```
}
```

В приведённой функции `i` – номер рассматриваемой позиции в строке `Bpref`, `j` – номер рассматриваемой позиции в строке `A`.

Кроме того, нужно выполнить следующие действия:

```
Nac[0] = true;
for (int i=1; i<NN; i++)
  Nac[i] = false;
for (int i=0; i<NN; i++)
  Otv[i] = false;
```

Полиномиальность всех описанных вычислений (относительно длины входных данных) очевидна.

VI. ПОЛИНОМИАЛЬНОСТЬ ОСНОВНОЙ ЧАСТИ АЛГОРИТМА – ПОДАЛГОРИТМА ПОСТРОЕНИЯ МНОЖЕСТВА ДУГ ГРАФА ПЕРЕХОДОВ АВТОМАТА

Продолжим описывать построение основной функции Ф (она же F). Как мы уже отмечали, это построение основной функции мы приведём не в виде программы (она получается слишком громоздкой) – а в виде подробного описания алгоритма, по которому эту программу легко построить [15].

Алгоритм построения одного значения функции Ф (в общих обозначениях – значения $\underset{A-B}{\Phi}(u)$.)

Вход [16]:

- строка `FiTmp` (длиной не более N^2),
- строка `B` (длиной не более N).

Также входными данными можно считать:

- булевы массивы `Nac` и `Otv` (каждый размерностью N^2) [17].

Выход: строка `Fi` (длиной не более N^2) – соответствующее значение [18] функции Ф.

Метод.

(0) Формируемую строку-ответ `Fi` полагаем равной пустой строке.

(1) Используя массив `Begins`, выбираем очередное слово из строки `FiTmp` (назовём это слово `Tmp`) и выполняем для каждого выбранного слова следующие действия (шаги (2)–(10), число повторений этой группы шагов ограничено значением N^2).

(2) Рассматриваем очередной (`i`-й) символ строки `Tmp` начиная с позиции `i=0`. Если `Nac[i]==true`, то выполняем шаги (3)–(6), число их повторений также ограничено значением N^2. (Иначе для этого `i`-го символа не делаем ничего.)

(3) Рассматриваем все слова из `B`. Шаги (4)–(6) выполняем для каждого такого слова (в описании этих шагов слово обозначено u), число этих слов ограничено значением N^2.

(4) Проверяем, верно ли, что длина слова u больше ещё не рассмотренной части слова (строки) `Tmp` – т. е. суффикса этой строки, начинающегося с `i`-го символа. Если это верно, то для слова u выполняем шаг (5), иначе – шаг (6).

(5) Сравниваем последовательно все буквы рассматриваемых подслов:

- суффикса строки `Tmp`, начинающегося с `i`-го символа;
- и префикса слова u той же самой длины.

Число сравнений символов ограничено значением N^2. Если все символы совпадают, то производим присваивание `Otv[i] = true` [19].

(6) Аналогично (альтернативный случай) – сравниваем последовательно все буквы двух других рассматриваемых подслов:

- сло́ва u;
- и подсло́ва строки `Tmp` той же самой длины, начинающегося с её `i`-го символа.

Число сравнений символов также ограничено значением N^2. Если все символы совпадают, то производим присваивание `Nac[i+u.size()] = true`.

(7) Если строка `Tmp` ещё не кончилась, то выполняем `i++`, после чего переходим на шаг (2). (Иначе переходим на шаг (8).)

(8) Ещё один раз рассматриваем строку `Tmp`: её очередной (`i`-й) символ, начиная с позиции `i=0`. Если `Otv[i]==true`, то выполняем шаг (9), число повторений этого шага также ограничено значением N^2. (Иначе для этого `i`-го символа не делаем ничего.)

(9) Для тех слов языка `B` (пусть v), для которых префиксы нужной длины совпадают с суффиксом строки `Tmp`, начинающемся с позиции `i`, добавляем в искомую строку-ответ `Fi` соответствующий (оставшийся) суффикс слова v.

(10) Если строка `Tmp` ещё не кончилась, то выполняем `i++`, после чего переходим на шаг (8). (Иначе выход из алгоритма.)

□

Согласно приведённым в [1], [3], [4] эквивалентным описаниям функции Ф очевидно, что в описанном алгоритме строится именно значение этой функции.

Теперь перейдём к оценке сложности алгоритма. При описании его шагов мы приводили максимально возможное число их повторений – это делалось для оценки сложности всего алгоритма. Также очевидно, что эта сложность полиномиальна – причём рассматривая 4 вложенных цикла, каждый из которых состоит из $\sim N^2$ повторений (N – размер входных данных) [20], несложно убедиться, что общая сложность, как уже было сказано

[15] Явное или неявное использование в нём некоторых стандартных функций Си++ – например, метода `size()` – проблем, по-видимому, не вызовет.

[16] Содержательный смысл всех входных данных совпадает с тем, который у них был в предыдущих разделах настоящей статьи. Размерность этих данных – тоже такая, как выше.

[17] Установка их начальных значений описана в предыдущем разделе. Однако их можно сформировать и в процессе работы описываемого алгоритма.

В массиве `Otv` в процессе работы алгоритма формируется промежуточный ответ – с этим связано его название.

[18] Представление строки имеет тот же самый формат.

[19] Отметим, что запоминать соответствующее этой позиции `i` слово u вряд ли целесообразно: этой позиции могут соответствовать несеольо разных слов, и поэтому придётся использовать сложную структуру данных. Проще лишний раз просмотреть всю строку `Tmp`, это не сделает алгоритм «более чем полиномиальным».

[20] Мы имеем в виду вложенные циклы, стартующие шагами (1), (2), (3) и (5).

выше, $\sim N^8$. Поэтому общая сложность всего получаемого алгоритма построения автомата NSPRI $\sim N^{12}$.

Таким образом, итоговая программа (итоговый алгоритм):

- не только сложнее объектно-ориентированного варианта из [1], [2],
- не только выполняется дольше него,
- но ещё и формально имеет очень большую (пусть и полиномиальную) степень.

Впрочем, мы и не стремились её (эту степень) минимизировать: по-видимому, такая минимизация вряд ли имеет практический смысл, поскольку, повторим, для практики объектно-ориентированноый вариант существенно лучше.

VII. ЗАКЛЮЧЕНИЕ

Кратко сформулируем направления дальнейшей работы – возможные публикации, связанные с рассмотренной в настоящей статье тематикой.

Ближайшая возможная тема публикации – про полурешётки потенциальных корней. Рассмотрим её немного более подробно. В предыдущих публикациях мы не раз отмечали похожесть двух задач для некоторого заданного конечного языка:

- задачи построения оптимального инверсного морфизма [21]
- и задачи извлечения корня заданной [22] степени.

Проблема здесь состоит в возможном описании *полиномиальных* алгоритмов для этих задач.

В обоих случаях необходимо построить множество т. н. потенциальных корней [23], обычно для заданного языка A обозначаемое нами

$$\overline{\sqrt[n]{A}},\ -$$

поэтому экспоненциальные алгоритмы для обеих этих задач очевидны: надо просто перебрать все подмножества множества этих потенциальных корней.

Для обеих подзадач вызывает интерес возможное существование двух полурешёток [14], построенных на множестве подмножеств потенциальных корней, – полурешётки по пересечению и полурешётки по объединению. При этом для второй из этих задач [24], вообще говоря, не имеется ни одной из этих полурешёток (примеры были приведены в [3], [4]) – возникающие структуры имеют более сложную структуру, также интересную для возможного изучения. А для первой из этих задач всегда (т. е. для любого заданного языка) получаются обе полурешётки – об этом и предполагается следующая публикация.

На основе второй из этих задач [25] можно сформулировать ещё одну – по-видимому, ещё более интересную: описать *полиномиальный* алгоритм построения оптимального инверсного морфизма. Причём согласно [15], решение этой задачи – часть *возможного плана* для доказательства равенства P=NP.

[21] Мы также отмечали, что эту оптимальность можно определять различными (но «естественными») способами.

[22] Либо максимально возможной.

[23] Это легко делается полиномиальным алгоритмом.

[24] По-видимому, более сложной.

[25] А также, конечно, на основе доказанного в настоящей статье.

А для первой из вышеуказанных двух задач (извлечения корня из языка) перед созданием полиномиального алгоритма её решения необходимо провести исследование возникающих структур – не являющихся, вообще говоря, полурешётками; однако они могут быть рассмотрены как специальный вариант «объединения» нескольких полурешёток. В связи с этим возникает несколько схожих задач (некоторые из них были упомянуты в [2]), связанных с извлечением корня из языка: например – построение любого корня и построение корня, содержащего минимально возможное число слов [26].

По поводу отсутствия полурешёток см. примеры в заключении статьи [4]. Поэтому, видимо, наиболее интересна задача построения *хотя бы одного* корня из заданного языка. Одной из вспомогательных для неё задач может быть такая. Для всей задачи – построения хотя бы одного корня – конечно же, надо рассматривать ситуацию когда нужная степень (или, в частном случае, квадрат) языка, состоящего из всех потенциальных корней, включает в себя исходное множество слов как *собственное* подмножество (такой пример и был приведён в [4]); назовём любое такое подмножество потенциальных корней [27] *«бóльшим»* множеством. И, понятно существуют *«меньшие»* подмножества потенциальных корней (в частности – пустое). Постепенно удаляя по одному потенциальному корня из максимально возможного большего множества, мы в конце концов получаем меньшее множество; для общего числа потенциальных корней M число таких возможных последовательностей удаления потенциальных корней равно $M!$. Вот две задачи про такие последовательности.

- Возможна ли ситуация, когда *за одни шаг* «большее» множество становится «меньшим»?
- Если на предыдущий вопрос ответ положителен – то назовём удаляемый для такого шага потенциальный корень «важным». Верно ли, что объединение всех «важных» корней даёт корень требуемой степени из заданного языка?

И ещё одна возможная задача – связанная с последними двумя. Перед её формулировкой повторим, что множество подмножеств потенциальных корней не образует решётку. Для конкретных решёток (по пересечению и объединению) все обладающие специальными ограничениями *свойства* этой решётки сохраняются при одновременной замене пересечения на объединение и наоборот – см. [14]. А в нашем случае – который *не* описывает решётку – выполняются ли те же самые факты при выполнении подобной одновременной замены? (Для произвольной алгебраической структуры с двумя операциями, аналогичными пересечению и объединению, ответ на этот вопрос, конечно, отрицателен – но ведь структура, состоящая из множества корней из заданного

[26] Такие же структуры возникают, например, в задачах минимизации недетерминированных конечных автоматов и дизъюнктивных нормальных форм. При этом построение любого корня – в качестве аналога в задаче минимизации ДНФ имеет построение (какой-нибудь) тупиковой ДНФ для заданной функции, а аналог построения «минимального» корня – построение ДНФ, содержащей минимальное число плоскостей. (Термин «минимальная ДНФ» мы стараемся не употреблять: в литературе он используется в разных смыслах.)

[27] Не обязательно включающим все потенциальные корни. Обязательно – включающим в себя исходное множество слов как собственное подмножество.

языка, обладает специальными свойствами – поэтому, возможно, в нашем случае ответ положителен.)

Все последние задачи можно рассматривать как *гипотезы*: для них у авторов нет ни доказательств, ни опровергающих примеров.

Практически все сформулированные сдесь задачи можно связать с развитием т. н. «задачи Пина» [16] – т. е. с тем фактом, что инверсный морфизм регулярного языка регулярен. Подобное изменение исходных условий задачи можно рассматривать при применении к исходным языкам любого инверсного морфизма – для всех задач, сформулированных в этом разделе.

А в качестве последней из задач, связанных с тематикой настоящей статьи, назовём возможное понижение 12-й степени при описании полиномиального алгоритма построения автомата для проверки равенства бесконечных итераций. Здесь нет противоречия с тем, что было сказано в конце предыдущего раздела: такое понижение степени вряд ли интересно для описания практических алгоритмов, но это – отдельно стоящая задача, и чисто теоретический интерес она, конечно, представляет.

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] Kameda T., Weiner P. *On the state minimization of nondeterministic finite automata* // IEEE Transactions on Computers. – 1970. – Vol. C-19. No. 7. – P. 617–627.

[6] Polák L. *Minimizations of NFA using the universal automaton* // International Journal of Foundation of Computer Sciences. – 2005. – Vol. 16. No. 5. – P. 999–1010.

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

[8] Melnikov B., Tsyganov A. *The state minimization problem for nondeterministic finite automata: the parallel implementation of the truncated branch and bound method* // In: Proceedings – 5th International Symposium on Parallel Architectures, Algorithms and Programming, PAAP–2012. – 2012. – P. 194–201.

[9] Долгов В., Мельников Б. *Построение универсального конечного автомата. I. От теории к практическим алгоритмам* // Вестник Воронежского государственного университета. Серия: Физика. Математика. – 2013. – № 2. – С. 173–181.

[10] Долгов В., Мельников Б. *Построение универсального конечного автомата. II. Примеры работы алгоритмов* // Вестник Воронежского государственного университета. Серия: Физика. Математика. – 2014. – № 1. – С. 78–85.

[11] Abramyan M., Melnikov B. *An approach to algorithmizing the problem of vertex minimization of nondeterministic automata. Part I. Problem statement and the brief description of the basis methods* // In: IOP Conference Series: Materials Science and Engineering. Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Associations. – 2020. – С. 52055.

[12] Марков А. *Теория алгорифмов (Труды Математического ин-та им. В. Стеклова. Т. 42)*. – М., Л.: Изд-во АН СССР. – 1954. – 376 с.

[13] Игошин В. *Математическая логика и теория алгоритмов*. – М.: Академия. – 2004. – 447 с.

[14] Скорняков Л. (ред.) *Общая алгебра. Том 2*. – М., Наука. – 1991. – 480 с.

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

[16] Pin J.-E. *Mathematical Foundations of Automata Theory*. – Berlin, Springer-Verlag. – 2012. – 310 p.

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.

A polynomial algorithm for constructing a finite automaton for determining the equality of infinite iterations of two finite languages

Boris Melnikov, Aleksandra Melnikova

***Abstract*—In this paper, we continue the topic related to the special binary relation on the set of formal languages (considered primarily on the set of iterations of non-empty finite languages); this is so called equivalence relation at infinity. We have formulated a simpler binary relation in a variety of languages (i.e. the covering relation), the double use of which is equivalent to the use of the equivalence relation at infinity. After that, we considered the algorithm for verifying the implementation of the coverage relation, and identified the auxiliary objects used to prove the correctness of this algorithm.**

One such object is an infinite iterative tree. In the infinite iterative trees considered, we combine the equivalent states, obtaining in fact a deterministic finite automaton. We have defined a specific such automaton for two given finite languages; this is so called primary automaton, PRI. It is deterministic, defined on sets of words, and each of these sets is a subset of the set of prefixes of the second of the given languages.

After that, we define some variants of special nondeterministic automata corresponding to it, describing in fact the construction of this iterative morphism tree. We introduce a completely different object (i.e., a simplified primary automaton, NSPRI), which also describes the construction of an iterative morphism tree, but is defined not on sets of words, but on words.

The main idea of the proof of the fact, that the algorithm for constructing a finite automaton for checking the equality of infinite iterations of two finite languages is polynomial is as follows. It is natural to construct a deterministic automaton for this problem, and each state of such deterministic automaton describes the whole set of possible prefixes remaining from various decompositions of the word (this is decompositions of the morphism of the iteration of the first given finite language according to the words of the second given finite language). However, in doing so, we are working with many sets of possible prefixes, which makes it impossible for the algorithm to be polynomial. Therefore, we construct a nondeterministic automaton defined simply on the set of possible prefixes; at the same time, there is some question related to when exactly this automaton gives a positive result. We solve this problem by requiring that the resulting nondeterministic automaton be an analog of the total deterministic automaton.

***Keywords*—formal languages, iterations of languages, morphisms, binary relations, infinite trees, 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] Kameda T., Weiner P. *On the state minimization of nondeterministic finite automata* // IEEE Transactions on Computers. – 1970. – Vol. C-19. No. 7. – P. 617–627.

[6] Polák L. *Minimizations of NFA using the universal automaton* // International Journal of Foundation of Computer Sciences. – 2005. – Vol. 16. No. 5. – P. 999–1010.

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

[8] Melnikov B., Tsyganov A. *The state minimization problem for nondeterministic finite automata: the parallel implementation of the truncated branch and bound method* // In: Proceedings – 5th International Symposium on Parallel Architectures, Algorithms and Programming, PAAP–2012. – 2012. – P. 194–201.

[9] Dolgov V., Melnikov B. *Building a universal finite automaton. I. From the theory to the practical algorithms* // Bulletin of the Voronezh State University. Series: Physics. Mathematics. – 2013. – No. 2. – P. 173–181 (in Russian).

[10] Dolgov V., Melnikov B. *Building a universal finite automaton. II. Examples of how algorithms work* // Bulletin of the Voronezh State University. Series: Physics. Mathematics. – 2014. – No. 1. – P. 78–85 (in Russian).

[11] Abramyan M., Melnikov B. *An approach to algorithmizing the problem of vertex minimization of nondeterministic automata. Part I. Problem statement and the brief description of the basis methods* // In: IOP Conference Series: Materials Science and Engineering. Krasnoyarsk Science and Technology City Hall of the Russian Union of Scientific and Engineering Associations. – 2020. – C. 52055.

[12] Markov A. *Theory of algorithms (Proceedings of the V. Steklov Mathematical Institute. Vol. 42)*. – Moscow, Leningrad: USSR Academy of Sciences Ed. – 1954. – 376 p. (in Russian).

[13] Igoshin V. *Mathematical logic and theory of algorithms*. – Moscow: Akademia. – 2004. – 447 p. (in Russian).

[14] Skornyakov L. (Ed.) *General Algebra. Vol. 2*. – Moscow, Nauka. – 1991. – 480 p. (in Russian).

[15] 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).

[16] Pin J.-E. *Mathematical Foundations of Automata Theory*. – Berlin, Springer-Verlag. – 2012. – 310 p.

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.