Полурешётки подмножеств потенциальных корней в задачах теории формальных языков. Часть II. Построение инверсного морфизма

Б. Ф. Мельников

***Аннотация*—В статье мы рассматриваем все возможные подмножества множества потенциальных корней, образующие в некоторых ситуациях полурешётки – по пересечению и / или по объединению. Такие структуры возникают в двух похожих задачах теории формальных языков. Конкретно, для некоторого заданного конечного языка мы рассматриваем задачу извлечения корня заданной степени, а также задачу построения оптимального инверсного морфизма – где оптимальность можно определить, например, как длину максимального слова языка, являющегося инверсным морфическим образом.**

В обоих приведённых случаях необходимо построить множество так называемых потенциальных корней – т. е. таких слов рассматриваемого алфавита, для каждого из которых некоторая его степень входит в исходный язык. При этом важно отметить, что построение множества всех потенциальных корней может быть выполнено с помощью несложного полиномиального алгоритма. Экспоненциальные алгоритмы для обеих задач очевидны: надо просто перебрать все подмножества множества этих потенциальных корней, и среди этих подмножеств выбрать подходящее. Поэтому проблема состоит в описании возможных полиномиальных алгоритмов для этих задач.

Для обеих указанных задач вызывает интерес возможное существование двух полурешёток, имеющихся на предварительно построенном множестве подмножеств потенциальных корней.

Среди прочего, мы в статье приводим формулировку одной важной гипотезы теории формальных языков – при выполнении которой мы можем утверждать, что специальное подмножество множества языков, каждый из которых является инверсным морфическим образом заданного языка, образует не только полурешётку по объединению, но и полурешётку по пересечению (а, следовательно, решётку).

Предлагаемая вторая часть статьи в большей степени посвящена второй из рассматриваемых задач – задаче построения оптимального инверсного морфизма.

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

Предлагаемая часть II статьи является продолжением [1]. Здесь мы продолжаем нумерацию разделов, а нумерация ссылок на использованную литературу – новая [1].

VII. ПОСЛЕДОВАТЕЛЬНОЕ УДАЛЕНИЕ ПОТЕНЦИАЛЬНЫХ КОРНЕЙ

Сформулируем несколько проблем про последовательности, фактически определённые в конце предыдущего раздела. По этому поводу нужно отметить, что почти все подобные проблемы нельзя назвать относящимися только к задаче извлечения корня: те же самые проблемы (практически без изменений условия) можно сформулировать и для второй задачи – задачи построения (оптимального) инверсного морфизма. Например, «большее» и «меньшее» подмножества множества потенциальных корней для второй задачи (с заданным языком A) естественно определить так:

- D – «большее» подмножество множества потенциальных корней в том случае, когда $D \rhd A$;
- D – «меньшее» подмножество множества потенциальных корней в том случае, когда $D \lhd A$.

Итак, приведём формулировки самих проблем.

- Возможна ли такая ситуация, когда *ровно за один шаг* «большее» множество становится «меньшим»? (При этом мы всё-таки не имеем в виду тривиальные примеры, а именно – не рассматриваем те потенциальные корни, которые имеют наименьшую либо наибольшую длину.) [2]
- Если на предыдущий вопрос ответ положителен – то назовём удаляемый для такого шага потенциальный корень *«важным»*. Верно ли, что объединение всех «важных» корней даёт:
  - в первой задаче – корень требуемой степени из заданного языка? [3,4]
  - а во второй задаче – множество, эквивалентное исходному?

  (Иными словами, описание алгоритма построения такого объединения всех «важных» корней может рассматриваться как решение любой из двух наших задач.)

Ещё одна возможная проблема связана с последними двумя, т. е. с множеством потенциальных корней, – однако не связана с их *последовательностью*. Перед формулировкой этой проблемы повторим, что множество подмножеств потенциальных корней в задаче извлечения

Статья получена 15 февраля 2022 г.

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

[1] Небольшая «алгебраическая» неточность в части I: дважды употреблены словá «частичный линейный порядок» В обоих случаях имелся в виду частичный порядок.

[2] Отметим, что такой выбор корней «работает» и в случае 2- (и более) буквенного алфавита.

[3] Напомним, что *пока* мы ищем *хоть один* подобный корень.

[4] Отметим, что задача, обратная для сформулированной, также является далеко не тривиальной. Мы действительно можем – например, выполнив вычисления с помощью специальной переборной компьютерной программы – получить разные примеры языков, каждый из которых имеет несколько корней (имеются в виду корни-языки). Но алгоритма, отвечающего на вопрос о «важности» какого-либо конкретного потенциального корня, такие примеры не дают – и здесь также возникает задача описать для ответа на этот вопрос алгоритм, отличный от переборного.

корня, вообще говоря, не образует решётку. Вспомним также, что для конкретных решёток (по пересечению и объединению) обладающие специальными ограничениями *свойства* каждой из них сохраняются при *одновременной* замене пересечения на объединение и наоборот – см. [2].

- Вопрос такой: в нашей ситуации – которая *не* описывает решётку – могут ли в каких-либо частных случаях выполняться те же самые факты при выполнении подобной одновременной замены? [5]

Для всех этих проблем у автора пока нет ответов – ни примеров, когда это возможно, ни доказательств невозможности [6].

А для тех из перечисленных задач, в которых требуется построить алгоритмы, возникают задачи дополнительные: построить соответствующие *полиномиальные* алгоритмы. Более того, положительный ответ на вторую из сформулированных выше задач *даст возможность получить полиномиальный алгоритм построения языка-корня* – т. е. полиномиально решить одну из двух основных задач настоящей статьи; однако бóльшие подробности (связанные с конкретным описанием построения такого полиномиального алгоритма) выходят за рамки рассматриваемых здесь вопросов.

В заключение раздела отметим, что из условия $B^n = A$ легко можно получить другое условие: $A \Leftrightarrow B$, также рассматриваемое в настоящей статье. Поэтому в некоторых ситуациях для задачи извлечения корня можно применять результаты, которые могут быть получены для второй задачи, – т. е. для задачи построения оптимального инверсного морфизма.

VIII. НЕСЛОЖНЫЙ ПРИМЕР

В этом разделе рассматривается несложный пример, который можно отнести к обеим рассматриваемым в статье задачам. Сразу заметим, что мы могли бы привести и альтернативное название раздела, то есть сказать во множественном числе, «примеры». Действительно, мы для одного и того же исходного языка начинаем рассматривать его «применение» как в задаче извлечения квадратного корня, так и в задаче построения оптимального инверсного морфизма [7]. В разделе VIII показывается, что множество потенциальных корней может включать в себя «лишние» – т. е. те корни, которые *не* были заданы при определении исходного языка с помощью некоторого морфизма. Также в настоящем разделе кратко описывается связь множества потенциальных корней с определёнными и кратко исследованными в [5], [6] автоматами PRI и NSPRI.

Итак, сам пример. Для алфавита

$$\Sigma = \{a, b\}$$

и языка над этим алфавитом

$$A = \{a, bab\} \tag{1}$$

(мы его уже применяли в примерах из наших предыдущих статей) рассмотрим язык

$$B = A^2 = \{aa, abab, baba, babbab\}.$$

Мы получаем такое множество потенциальных квадратных корней:

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

Далее. Для того же самого языка A будем рассматривать эквивалентный ему [8] язык

$$C = \{aa, abab, babababa, bab\}. \tag{2}$$

Действительно, эквивалентность $A \Leftrightarrow C$ следует, например, из того, что для некоторого нового алфавита

$$\Delta = \{0, 1\}$$

и соответствующего морфизма

$$h_A(0) = a, \quad h_A(1) = bab$$

выполнено следующее [9]:

$$h_A^{-1}(C) = \{00, 01, 1010, 1\};$$

а последний язык входит в $mp^+(\Delta)$ – поскольку язык

$$\{00, 01, 1\}$$

является, с одной стороны, его (собственным) подмножеством, а с другой стороны, некоторым максимальным префиксным кодом.

Множество потенциальных корней здесь таково:

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

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

В заключении раздела ещё раз отметим, что в обоих примерах при построении $h_A^{-1}(B)$ и $h_A^{-1}(C)$ декодирование было однозначным; этот факт в наших примерах можно считать следствием критерия однозначного декодирования для рассматриваемого в этом разделе языка A, см. подробнее [3], [4] и мн. др. Однако, как следует из предыдущего изложения, последнее условие (однозначность декодирования) не является обязательным –

[5] 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 языка (а во второй задаче – из множеств, эквивалентных заданному языку по отношению $\Leftrightarrow$), обладает какими-то специальными свойствами – которые мы ещё до конца не знаем… Поэтому, возможно, в каких-то частных случаях ответ на такой вопрос может быть положительным.

[6] В одной из ближайших статей автор предполагает привести *компьютерные программы* (и результаты их работы) для проверки этих утверждений «в самых простых ситуациях» – т. е. *для 1-буквенного алфавита* и ограниченности на максимальную длину слова. (Стóит отметить, что даже для таких простых ситуаций алгоритмы проверки нетривиальны: ведь при ограничении на максимальную длину слова N может быть до $N/2$ потенциальных корней степени 2, и поэтому, как мы уже отмечали, может быть до $(N/2)!$ возможных последовательностей их удаления.)

Однако, для общего случая (т. е. для степени, превышающей 2, и, главное, для алфавита, мощность которого превышает 1) все поставленные вопросы останутся открытыми и после публикации результатов работы такой компьютерной программы.

[7] При этом может показаться, что разницы в названиях практически нет. Однако рассмотрение примеров в каждой из этих задач ведётся по-разному, поскольку рассматривается либо равенство специально построенных языков, либо эквивалентность языков, заданных заранее.

[8] С точки зрения бинарного отношения $\Leftrightarrow$.

[9] Здесь всё очень просто: для рассматриваемого языка A выполняется критерий однозначности декодирования – см. некоторые подробности далее.

и более сложные примеры мы также предполагаем рассмотреть в статье-продолжении.

Можно также считать, что некоторые примеры, связанные с *не* выполнением этого критерия, мы *уже* рассмотрели в наших предыдущих публикациях при формальном определении автоматов PRI и NSPRI, в частности – в конкретных примерах этих автоматов. А именно, в тех примерах было важно то, что при построении автоматов $\mathrm{PRI}(A, B)$ (а также $\mathrm{NSPRI}(A, B)$) критерий однозначности декодирования *не* выполнялся для рассматривавшихся в примерах языков B.

По поводу упомянутых в предыдущем абзаце автоматов PRI и NSPRI отметим также следующее. Они оба определены для двух заданных конечных языков A и B и предназначены для удобной записи алгоритма проверки выполнения условия $A \trianglelefteqslant B$. *Детерминированный* автомат $\mathrm{PRI}(A, B)$ определён на *подмножествах множества* собственных суффиксов; он был подробно рассмотрен в [5]. А *недетерминированный* автомат $\mathrm{NSPRI}(A, B)$ эквивалентен последнему [10] – но определён на *множествах* собственных суффиксов; он был подробно рассмотрен в [6]. Кроме того, элементы компьютерной программы и примеры построения автоматов PRI и NSPRI для «сложных» языков были рассмотрены в [7].

IX. ПРИНАДЛЕЖНОСТЬ ЯЗЫКА МНОЖЕСТВУ МОРФИЗМОВ РАСШИРЕННЫХ МАКСИМАЛЬНЫХ ПРЕФИКСНЫХ КОДОВ

В этом разделе рассматривается один из возможных алгоритмов, определяющих принадлежность некоторого заданного языка множеству морфизмов расширенных максимальных префиксных кодов; при этом сам язык-морфизм также заранее задан. Отметим, что предлагаемый здесь алгоритм, конечно же, *не является* решением второй рассматриваемой в статье задачи: ведь оба языка (A и B в приведённых ранее обозначениях, а здесь и обычно далее – D вместо B [11]) заранее заданы.

Итак, для заданных конечных языков $A, B \subseteq \Sigma^*$ необходимо ответить на вопрос о выполнении условия $A \in \mathrm{mp}^+(B)$.

Алгоритм IX.1: проверка выполнения условия расширенного максимального префиксного кода.

Вход: конечные языки $A, D \subseteq \Sigma^*$; пусть

$$D = \{v_1, \ldots, v_n\}.$$

Выход:

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

Метод.

1. Выбрать *произвольную* «нумерацию» для D-морфизма, а именно – некоторый конкретный варант отображения

$$h_D(d_1) = v_1, \ \ldots, \ h_D(d_n) = v_n$$

где $\Delta = \{d_1, \ldots, d_n\}$ – некоторый новый алфавит.

2. Для каждого слова $u \in A$ рассмотреть все возможные варианты представления

$$u = v_{i_1} \ldots v_{i_k}, \quad \text{где} \quad i_1, \ldots, i_k \in \{1, \ldots, n\}. \tag{3}$$

(Отметим, что для таких действий *существует детерминированный полиномиальный алгоритм*. Подобные алгоритмы мы несколько раз применяли в [8] – причём каждый раз также в качестве вспомогательных [12]. Однако *пока* эта «детерминированная полиномиальность» нам не нужна [13].)

3. Для каждого из полученных вариантов представления (3) рассмотреть соответствующее ему слово

$$d_{i_1} \ldots d_{i_k} \in \Delta^*. \tag{4}$$

По определению, объединяя *все* слова вида (4), соответствующие рассматриваемому слову u, мы таким образом получаем язык $h_D^{-1}(u)$.

4. Также по определению

$$h_D^{-1}(A) = \bigcup_{u \in A} h_D^{-1}(u).$$

Обозначим полученный язык $A_\Delta = h_D^{-1}(A)$.

5. Рассмотрим язык [14]

$$A'_\Delta = \{u_\Delta \in A_\Delta \mid (\nexists v_\Delta \in A_\Delta)(v_\Delta \in \mathrm{opref}(u_\Delta))\}.$$

6. Выйти из алгоритма с ответом $A'_\Delta \in \mathrm{mp}(\Delta)$ [15]. Такой ответ можно получить, например, путём применения алгоритма III.1 (часть I настоящей статьи) [16].

Конец описания алгоритма [17].

Корректность алгоритма IX.1 следует из приведённых выше определений.

В заключение раздела отметим, что мы *уже сейчас* могли бы говорить про оптимальность построенного

[10] На самом деле всё немного сложнее. Автоматы фактически эквивалентны *с точки зрения работы* – но *не* с точки зрения принятия некоторого слова. А именно принятие слова (точнее, *всех возможных слов*) является целью построения автомата $\mathrm{NSPRI}(A, B)$: он даёт положительный ответ в том и только том случае, когда допускает любое слово над получающимся алфавитом – алфавитом инверсного морфического образа. Мы *не можем* проверить это условие (допущение автоматом любого слова) *полиномиально* – и это даёт основания считать, что алгоритмы ресшения рассматриваемых нами задач также неполиномиальны.

[11] Логика в такой «замене» следующая. Языком D мы во второй задаче обычно обозначаем *возможно существующий общий* язык (при этом часто – один из минимально возможных согласно одному из рассматриваемых далее отношений частичного порядка), эквивалентный по отношению $\trianglelefteqslant$ обоим *заданным* языкам – которые логично обозначать A и B.

[12] Что подчёркивается названием той статьи – «Полиномиальный алгоритм построения конечного автомата …».

[13] И это можно объяснить следующим образом:

- во-первых, далее в описываемом алгоритме она «нарушается»;
- и, во-вторых, ниже в настоящей статье её применять не нужно: для возможной полурешётки нам важно *только существование* какого-либо подобного вспомогательного алгоритма.

Однако в одной из следующих публикаций мы предполагаем вернуться к детерминированным полиномиальным алгоритмам решения этой задачи.

[14] Он легко получается с помощью полиномиального (конкретно – квадратичного) алгоритма.

[15] То есть `true` в случае $A'_\Delta \in \mathrm{mp}(\Delta)$, иначе – `false`.

[16] Конечно, в нашем случае возможны и значительно более простые алгоритмы.

[17] В связи с этим алгоритмом приведём такую аналогию – «со студенческим материалом». Для каждого из 4 классов иерархии Хомского существует описание каждого из его языков:

- либо с помощью грамматики своего типа – работа которой «чем-то похожа» не недетерминированный алгоритм (приведённый нами, например, в разделе V части I);
- либо с помощью соответствующего автомата-распознавателя – который «похож» на детерминированный алгоритм (даже несмотря на то, что сам автомат может быть недетерминированным).

Впрочем, такая возможная аналогия не влияет на дальнейшее изложение.

морфизма – даже до рассмотрения в разделе XI возможных вариантов частичных порядков на множестве языков: оптимальность морфизма можно определить на основе «абстрактного» (заранее неизвестного) частичного порядка. Однако в этом разделе *не стояла* цель построить оптимальный морфизм [18].

X. ПРИМЕРЫ – НОВЫЙ И ПРОДОЛЖЕНИЕ СТАРОГО

Приводимые в этом разделе примеры (новый – а также, что важнее, продолжение рассмотренного в разделе VIII) должны не только пояснить алгоритм раздела предыдущего, но и показать возможные варианты применения этого алгоритма.

Во-первых – новый пример. Рассмотрим алфавит из одной буквы (a), а над этим алфавитом рассмотрим язык из одного сло́ва –

$$\{a^{6k}\} \quad \text{для некоторого} \quad k \geqslant 1.$$

Важно отметить, что формально для разложения слова заданного языка по потенциальным корням мы должны рассматривать возможность *любого порядка* этих потенциальных корней. В нашем случае существуют потенциальные корни a и aa; остальные, в частности обязательно имеющиеся корни a^3 и a^6 (а также a^k, возможно не равный ни одному из них), мы даже не станем рассматривать. Объяснить это нерассмотрение можно так: мы хотим получить оценку *снизу*, существование которой покажет неполиномиальность простейшего возможного алгоритма. Для получения одной из таковых – оценим снизу количество представлений слова a^{6k} в виде произведения выбранных нами потенциальных корней a и aa.

Сколькими способами представляется число 6k в виде суммы чисел 1 и 2? [19] Применим «очередное снизу»: будем рассматривать только те суммы, которые содержат ровно 2k значений 1 и ровно 2k значений 2. Таковых сумм имеется

$$\binom{4k}{2k} \sim \frac{16^k}{\sqrt{2\pi k}}$$

(мы применяем нужную формулу из [9]). Последняя оценка поясняет неполиномиальность простейшего возможного алгоритма проверки выполнения условия расширенного максимального префиксного кода.

Теперь перейдём к продолжению рассмотрения примера из раздела VIII – а именно, рассмотрим язык C (2). Перепишем *явно* все потенциальные корни:

$$\overline{\sqrt[*]{C}} = \{a, aa, ab, abab, ba, bab, baba, bababa\}$$

(для удобства записываем их в алфавитном порядке).

Далее рассмотрим два «интересных» варианта выбора подмножеств этого множества – этот выбор нужен для формирования D-морфизма.

Во-первых – «слишком большой» вариант. Пусть

$$D = \overline{\sqrt[*]{C}},$$

и для него рассмотрим такой («алфавитный») D-морфизм:

$$h_D(1) = a,\ h_D(2) = aa,\ h_D(3) = ab,$$
$$h_D(4) = abab,\ h_D(5) = ba,\ h_D(6) = bab,$$
$$h_D(7) = baba,\ h_D(8) = bababab$$

(далее нижний индекс D будем опускать). При этом, конечно,

$$\Delta = \{1, 2, 3, 4, 5, 6, 7, 8\}.$$

Для слов языка C (2) получаем:

$$aa = h(11) = h(2),$$
$$abab = h(33) = h(4),$$
$$babababa = h(5555) = h(5561) = h(557) =$$
$$h(5615) = h(5631) = h(575) =$$
$$h(6155) = h(6161) = h(617) =$$
$$h(6315) = h(6331) = h(641) =$$
$$h(755) = h(761) = h(77) = h(8),$$
$$bab = h(6)$$

(и других вариантов представления нет).

Из последнего получаем, что

$$C_\Delta = h^{-1}(C) = \{11, 2, 33, 4,$$
$$5555, 5561, 557, 5651, 5631, 575,$$
$$6, 6155, 6161, 617, 6315, 6331, 641,$$
$$755, 761, 77, 8\}$$

(мы выписали слова множества C_Δ в алфавитном порядке).

Удаляя из последнего языка слова, у которых существуют собственные префиксы, входящие в этот же язык, получаем следующее:

$$C'_\Delta = \{11, 2, 33, 4,$$
$$5555, 5561, 557, 5651, 5631, 575,$$
$$6, 755, 761, 77, 8\}.$$

Мы видим, что язык C'_Δ *не является* максимальным префиксным кодом над алфавитом Δ [20].

Во-вторых – «хороший» вариант [21]. Пусть

$$h(0) = a,\ h(1) = bab \quad \text{для} \quad \Delta = \{0, 1\}.$$

Для слов языка C (2) получаем:

$$aa = h(00), \quad abab = h(01),$$
$$babababa = h(1010), \quad bab = h(1)$$

[18] Вообще, мы предполагаем вернуться к этому вопросу в одной из следующих публикаций – причём «объединив» оптимальность с полиномиальностью.

[19] Повторим, что порядок важен – в такой формулировке это порядок слагаемых. И отметим, что каждому из таких порядков соответствует некоторое слово над алфавитом, который в разделе IX был обозначен как Δ, – причём все получающиеся подобные Δ^*-слова различны.

[20] Отметим следующее. Отсутствие языка – подмножества C_Δ, не являющегося максимальным префиксным кодом, можно получить на основе рассмотрения только этого языка (C_Δ) – т. е. без построения C'_Δ. *Возможно*, ответ удастся получить даже без построения C_Δ (т. е. на основе какого-то более простого варианта рассмотрения языков A и C). *Возможно*, всё это даст более простой вариант основного алгоритма – алгоритма построения оптимального инверсного морфизма.

[21] Мы – как homines sapientes, причём даже «авторы» этого примера – такой вариант *знаем заранее*: фактически он уже был почти полностью рассмотрен в разделе VIII. И не будет большим преувеличением сказать, что основная цель заключается в том, чтобы такой («хороший») вариант компьютер находил бы сам, причём *применяя полиномиальный алгоритм*.

(и других вариантов представления нет [22]). Из последнего получаем, что

$$C_\Delta = h^{-1}(C) = \{00, 01, 1010, 1\}.$$

Удаляя из последнего языка единственное слово, у которого префикс входят в этот же язык, получаем следующее:

$$C'_\Delta = \{00, 01, 1\}.$$

Мы видим, что язык C'_Δ *является* максимальным префиксным кодом над алфавитом Δ.

Отметим, что в рассмотренном примере *любая* последовательность удаления потенциальных корней, в которой первыми 6 удаляемыми элементами являются «ненужные» потенциальные корни

$$aa, ab, abab, ba, baba, babababa,$$

обладает следующими свойствами:

- до удаления всех этих 6 потенциальных корней – вариант «слишком большой»;
- после удаления их (и только их) – вариант «хороший»;
- а после удаления их плюс ещё какого-либо потенциального корня [23] – мы не можем представить все слова исходного языка C (2) в виде конкатенации оставшихся потенциальных корней.

Доказать эти свойства несложно.

Итак, это были примеры «слишком большого» и «хорошего» подмножеств множества потенциальных корней. Однако мы не рассмотрели *интересного* примера для «слишком маленького» подмножества множества потенциальных корней. При этом недостаточное число потенциальных корней (например – только множество $\{bab\}$ для рассматриваемого языка), конечно, таким интересным примером не является, поскольку мы просто не для всех слов исходного языка C (2) можем получить их представление как конкатенацию множества потенциальных корней.

Однако пример «слишком маленького» подмножества множества потенциальных корней получить можно – но полный пример (а именно – если начать его рассмотрение с соответствующего ему варианта, названного выше «слишком большим») получился бы очень громоздким. Для получения такого примера [24] добавим в исходный язык C (2) ещё одно слово – например, bb; но, конечно, при этом должен измениться и исходный язык A (1), и новый его вариант также предусматривает добавление слова bb:

$$A = \{a, bab, bb\}.$$

При этом некоторые варианты подмножества множества потенциальных корней, включающие b, но не включающий bb, как раз и являются такими «слишком маленькими» подмножествами, в которых:

- все слова исходного языка C (2) можно получить в виде их представления конкатенацией множества потенциальных корней,
- но при этом построенный согласно приведённому выше алгоритму язык C'_Δ; *не является* максимальным префиксным кодом над соответствующим алфавитом Δ.

В заключение раздела отметим, что

мы пока не утверждаем, что вариант подмножества множества потенциальных корней, названный выше «хорошим», может быть только один.

(Повторим другими словами, что «хороший» вариант означает, что в результате применения к нему алгоритма IX.1 получается язык, находящийся с рассматриваемым в отношении $\Lsh\!\!\Rsh$.)

XI. «ЕСТЕСТВЕННЫЕ» ЧАСТИЧНЫЕ ПОРЯДКИ НА МНОЖЕСТВЕ ЯЗЫКОВ [25]

В этом разделе мы определяем два «естественных» частичных порядка на множестве языков. В принципе возможно определение ещё нескольких (также «естественных») частичных порядков – см., например, [10], [11], [12]. Из них мы выбрали такие два порядка, которые:

- во-первых, легко применимы в рассматриваемых нами задачах,
- и, во-вторых, строятся на разных принципах.

Более того, мы в статье будем пользоваться только первым из них – но практически всё сказанное про него остаётся верным и при использовании второго [26]. Заранее отметим, что оба приведённых определения озаглавлены как «определение-алгоритм» – это подчёркивает то, что они оба являются *конструктивными*.

Определение-алгоритм XI.1: *первый* частичный порядок на множестве языков.

1. Рассматриваем произвольный конечный язык; пусть это

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

где

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

Последовательность

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

будем обозначать записью $Le(A)$ [27]; при этом для непустых последовательностей считаем

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

При этом последовательность $Ta(A)$ может быть пустой, обозначаем таковую $()$.

2. Для двух конечных языков A и B:

- если $A = B = ()$, полагаем $A \simeq B$ [28];

[22] Здесь выполняется критерий однозначности декодирования – но существуют и «более интересные» примеры.

[23] Называть получающийся при этом вариант «слишком маленьким» нелогично – см. далее.

[24] Подробные вычисления можно провести, например, с помощью компьютерных программ.

[25] Можно привести и альтернативное название этого раздела – заменив «множество языков» на «элементы глобального надмоноида свободного моноида». (Ещё раз отметим, что нередко применяемое для последнего объекта сокращённое название «супермоноид» не очень удачно.)

[26] Да и, по-видимому, любого из дугих «естественных»; именно поэтому мы слово «естественный» и употребляем.

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

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

- иначе если $A = ()$, полагаем $A \prec B$;
- иначе если $B = ()$, полагаем $B \prec A$ [29];
- иначе если $He(A) < He(B)$, полагаем $A \prec B$;
- иначе ответ ($A \simeq B$, либо $A \prec B$, либо $A \succ B$) выбираем так же, как для последовательностей $Ta(A)$ и $Ta(B)$ соответственно.

Таким образом на множестве языков определяется частичный порядок [30], [13], [14], [15]. Обозначения $\preccurlyeq$ и $\succcurlyeq$ при необходимости также употребляем обычным образом: ($\prec$ либо $\simeq$) и ($\succ$ либо $\simeq$) соответственно.

Конец формулировки определения.

Понятно, что «интересные для нас» языки [31]

$$A = \{0, 10, 11\} \quad \text{и} \quad B = \{00, 01, 1\} \tag{5}$$

для определённого таким образом частичного порядка «равны». (Точнее – они находятся в отношении $\simeq$.)

А для языков

$$\begin{gathered} A = \{0, 10, 11\} \quad \text{и} \\ C = \{0, 10, 100, 11\} = A \cup \{100\} \end{gathered} \tag{6}$$

согласно определению первого частичного порядка получаем $A \prec C$.

Определение-алгоритм XI.2: *второй* частичный порядок на множестве языков.

1. Для слов $u, v \in \Sigma^*$ будем в этом определении считать, что:

- $u \prec v$, тогда и только тогда, когда $u \subseteq opref(v)$;
- $u \preccurlyeq v$, тогда и только тогда, когда $u \subseteq pref(v)$.

2. Для языков $A, B \subseteq \Sigma^*$ будем считать, что $A \preccurlyeq B$, тогда и только тогда, когда

$$(\forall u \in A)\,(\exists v \in B)\,(u \preccurlyeq v),$$

и при этом

$$(\forall u \in A)\,(\nexists v \in B)\,(v \prec u).$$

3. Считаем, что $A \prec B$, в случае, если $A \preccurlyeq B$, и при этом $A \neq B$ [32].

Конец формулировки определения.

Как видно из приведённых определений, обозначения для порядков применяются одинаковые – что не помешает изложению: прежде всего потому, что, как уже было отмечено, в основном будет применяться первый частичный порядок [33]. Однако здесь мы рассмотрим применение второго порядка – для уже определённых выше пар языков.

Так, языки A и B (5) по второму частичному порядку несравнимы. А для языков A и C (6) выполнено условие $A \prec C$.

XII. ЗАДАЧА ПОСТРОЕНИЯ ОПТИМАЛЬНОГО ИНВЕРСНОГО МОРФИЗМА: НЕДЕТЕРМИНИРОВАННЫЙ ПОЛИНОМИАЛЬНЫЙ АЛГОРИТМ

В этом разделе мы рассматриваем недетерминированный полиномиальный алгоритм решения второй задачи – задачи построения оптимального инверсного морфизма. В отличие от недетерминированного же алгоритма из раздела V (часть I), в рассматриваемой здесь задаче задан только один язык A. Требуется найти язык B над тем же алфавитом Σ, такой что $A \in \mathfrak{mp}^+(B)$. При этом, как и в ранее рассмотренном случае извлечения корня из языка, алгоритм этого раздела даёт возможность получить:

- как *один*, т. е. *первый* подходящий ответ [34],
- так и *все возможные* подходящие ответы – причём из них апостериори можно выбрать *один, оптимальный* подходящий ответ.

Алгоритм XII.1: построение (оптимального) инверсного морфизма.

Вход: язык $A \subseteq \Sigma^*$.

Выход: язык $B \subseteq \Sigma^*$, такой что $A \in \mathfrak{mp}^+(B)$.

Метод.

Шаг 1. Построить язык C – множество потенциальных корней (любой степени) из A:

$$C = \overline{\sqrt[*]{A}};$$

пусть $|C| = M$ (число таких корней равно M). Зафиксировать номер для каждого из элементов языка C.

Шаг 2. Выбрать очередное подмножество D множества C из общего числа $2^M - 1$ непустых подмножеств его элементов. Если таких множеств больше не существует, то выход из алгоритма.

Шаг 3. Если $A \in \mathfrak{mp}^+(D)$, то добавление множества D в ответ. При этом если необходимо получить только один вариант ответа, то выход из алгоритма.

Шаг 4. Переход на шаг 2.

Примечание. К приведённому алгоритму (конечно же, к тому его варианту, который выдаёт все возможные подходящие языки) можно «добавить» *оптимальность* – т. е. среди найденных подходящих языков D выбирать после их получения оптимальный по некоторому заранее заданному критерию [35].

Конец описания алгоритма.

Отметим, что как и в случае алгоритма из раздела V (часть I), приведённый алгоритм является недетерминированным полиномиальным (относительно «размера задачи» – т. е. суммы длин всех слов исходного языка A) – это очевидно.

В заключение раздела отметим следующее. Согласно «студенческим» алгебраическим определениям, рассматриваемое нами бинарное отношение $\diamondsuit$ разбивает весь класс языков на подклассы; об этом мы кратко уже писали, см. [16, Sect. 7] [36]. Согласно какому-либо частичному порядку, в каждом из таких подклассов существует

[29] Этот пункт определения можно опустить, рассматривая переобозначения языков и обозначение $A \succ B$ для $B \prec A$. Аналогичные переобозначения подразумеваем всюду далее – и «лишних» пунктов не приводим.

[30] В данном случае – являющийся полным.

[31] Это – два различных максимальных префиксных кода над одним и тем же алфавитом.

[32] Возможная альтернативная формулировка – $A \preccurlyeq B$, и при этом неверно, что $B \preccurlyeq A$.

[33] Второй частичный порядок – а также, возможно, и некоторые другие частичные порядки на множествах языков – мы предполагаем подробнее рассматривать в последующих публикациях.

[34] «Первый попавшийся».

[35] Такое же примечание возможно и для алгоритма из раздела V (часть I) – однако в том случае подобный вариант, по-видимому, менее интересен.

[36] Однако какого-либо исследования подобных подклассов мы практически не проводили. Можно считать, что такая тема представляет собой возможный предмет следующих публикаций.

некоторый минимальный язык D, причём, аналогично разделу X, можно сказать следующее:

> *мы пока не утверждаем, что такой язык* D *в каждом подклассе может быть выбран единственным образом.*

В заключение раздела отметим, что в предыдущих построениях желательно применять какой-либо полный линейный порядок. И, конечно, любой выбираемый нами частичный порядок не обязан совпадать с одним из порядков, определённых выше в разделе XI.

Как уже было отмечено, оставшийся материал статьи будет приведён в части III.

Список литературы

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

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

[3] Яблонский С. *Введение в дискретную математику. Учебное пособие для вузов*. – М., Высшая школа. – 2001. – 384 с.

[4] Новиков Ф. *Дискретная математика для программистов*. – СПб, Питер. – 2009. – 304 с.

[5] 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.

[6] 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.

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

[8] Мельников Б., Мельникова А. *Полиномиальный алгоритм построения конечного автомата для проверки равенства бесконечных итераций двух конечных языков* // International Journal of Open Information Technologies. – 2021. – Vol. 9. No. 11. – P. 1–10.

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

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

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

[12] Мельников Б. *Регулярные языки и недетерминированные конечные автоматы (монография)*. – М., Изд-во Российского государственного социального университета. – 2018. – 179 с. – ISBN 978-5-7139-1355-7.

[13] Скорняков Л. (ред.) *Общая алгебра. Том 1*. – М., Наука. – 1990. – 592 с.

[14] Хаусдорф Ф. *Теория множеств*. – М., УРСС. – 2007. – 304 с.

[15] Гуров С. *Булевы алгебры, упорядоченные множества, решетки: определения, свойства, примеры*. – М., Либроком. – 2013. – 352 с.

[16] 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.

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.

Semi-lattices of the subsets of potential roots in the problems of the formal languages theory. Part II. Constructing an inverse morphism

Boris Melnikov

***Abstract*—In the paper, we consider all possible subsets of the set of potential roots forming in some situations semi-lattices, by intersection and / or by union. Such structures arise in two similar problems in the theory of formal languages. Specifically, for some given finite language, we consider the problem of extracting the root of a given degree and the problem of constructing an optimal inverse morphism, where optimality can be defined, for example, as the length of the maximum word of a language that is an inverse morphic image.**

In both of the above cases, it is necessary to construct the set of so-called potential roots, i.e., such words of the alphabet in question, for each of which some degree of it is included in the source language. It is important to note that the construction of the set of all potential roots can be performed using a simple polynomial algorithm. Exponential algorithms for both problems are obvious: we just need to sort through all subsets of the set of these potential roots, and choose the appropriate one among these subsets. Therefore, the problem is to describe possible polynomial algorithms for these problems.

For both of these problems, the possible existence of two semi-lattices available on a pre-constructed set of subsets of potential roots is of interest.

Among other things, in the paper we present the formulation of one important hypothesis of the theory of formal languages, in which we can assert that a special subset of a set of languages, each of which is an inverse morphic image of a given language, forms not only a half-lattice by union, but also a half-lattice by intersection (and, therefore, a lattice).

The proposed second part of the paper is more devoted to the second of the problems under consideration, i.e., to the problem of constructing an optimal inverse morphism.

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

References

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

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

[3] Yablonskiy S. *Introduction to Discrete Mathematics. Study Guide for Universities*. – Moscow, Vysshaya Shkola. – 2001. – 384 p. (in Russian).

[4] Novikov F. *Discrete mathematics for programmers*. – Saint Petersburg, Piter. – 2009. – 304 p. (in Russian).

[5] 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).

[6] 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).

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

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

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

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

[11] 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).

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

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

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

[15] Gurov S. *Boolean algebras, ordered sets, lattices: definitions, properties, examples*. – Moscow, Librokom. – 2013. – 352 p. (in Russian).

[16] 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.

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`.