РОССИЙСКАЯ АКАДЕМИЯ НАУК
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ имени М.В. Келдыша

Логачева В.К., Клышинский Э.С., Галактионов В.А.

Автоматическая генерация правил транскрипции и машинная транскрипция имен собственных с использованием конечного автомата

Москва
2012

Логачева В.К., Клышинский Э.С., Галактионов В.А.

Автоматическая генерация правил транскрипции и машинная транскрипция имен собственных с использованием конечного автомата

Аннотация
В работе предлагается метод автоматической генерации правил транскрипции имен собственных на основе анализа обучающей выборки. Процесс генерации правил разбивается на два этапа: выделение простых (первичных) правил и генерация сложных правил. Для выделения первичных правил используется новая методика выравнивания. Для проведения транскрипции предлагается конвертировать правила в конечный автомат и проводить транскрипцию по нему.

Logacheva V.K., Klyshinsky E.S., Galaktionov V.A.

Transliteration rules generation and named entities transliteration conducted by a finite-state machine

Abstract
The work proposes a method of cross-lingual transliteration rules generation. The method is based on analysis of a training set. Process of generation is divided into two stages: the first is extraction of simple (initial) rules, the second one is generation of complicated rules. The first stage uses the new alignment algorithm described in the paper. The transliteration is conducted by a finite-state machine constructed from a system of transliteration rules.

Работа выполнена при поддержке РФФИ, грант № 10-01-00800-а

Содержание

1. Предварительные замечания

Как уже было сказано выше, тема настоящей работы – передача слов с одного языка на другой. Таким образом, описываемый метод работает с двумя языками, один из которых является языком-источником (язык, из которого взято имя), другой – языком-приемником (язык, на который переводится имя). Алфавит языка-источника обозначим через $\mathbf{V_I}$, алфавит языка-приемника – через $\mathbf{V_O}$.

Цепочки символов (в том числе пустые) входного и выходного языков будем обозначать греческими буквами. Символы входного и выходного языков обозначим буквами *u* и *v*.

Целью работы является создание метода, позволяющего генерировать межъязыковые соответствия подстрок, другими словами, определить отображение множества подстрок языка-источника в множество подстрок целевого языка. Каждое из таких соответствий подстрок назовем правилом транслитерации.

Правила генерируются из обучающего множества пар имен исходного языка и их записей на целевом языке. Обучающее множество составлено вручную экспертом.

Определим правило как пару $\mathbf{r} = <\mathbf{p}, \beta>$, где:

p – левая часть правила – преобразовываемая цепочка символов входного алфавита, возможно, с контекстами;

β – правая часть правила – цепочка символов выходного алфавита (возможно, пустая), которая соответствует левой части правила в целевом языке.

Левая часть правила определяется следующим образом:

$\mathbf{p} = <\mathbf{p_l}, \alpha, \mathbf{p_r}>$, где $\mathbf{p_l}$ и $\mathbf{p_r}$ – левый и правый контексты соответственно, α – преобразовываемая строка, $\mathbf{p_l} = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$, $\gamma_i \in \mathbf{V_I}^+$, $\mathbf{p_r} = \{\delta_1, \delta_2, \ldots, \delta_m\}$, $\delta_i \in \mathbf{V_I}^+$;

Исходя из этого, правило применимо с текущей позиции, если на текущей позиции находится подстрока α, перед ней полностью представлена одна из подстрок из $\mathbf{p_l}$, после нее – подстрока из $\mathbf{p_r}$. Считается, что к входной строке последовательно ищутся применяемые правила. При нахождении такого правила текущая позиция сдвигается вправо на |α| символов, а на выход подается β (под обозначением |α| здесь и далее понимается длина подстроки α).

При обучении системы машинной транскрипции необходима дополнительная информация о правиле: как часто и в каких словах обучающей выборки оно встретилось. Поэтому при обучении используется расширенный формат правила: $\mathbf{r} = <\mathbf{p}, \beta, \mathbf{w}>$, где **p** и с те же, что в основном формате, а $\mathbf{w} = \{<w_1, pos_1>, \ldots, <w_n, pos_n>\}$, где w_i – слово из обучающего множества, к которому применимо данное правило, pos_i – позиция, на которой находится первый символ из $\mathbf{p_2}$ в слове w_i.

2. Метод порождения правил

Алгоритм порождения правил транскрипции состоит из двух этапов: порождение первичных правил и порождение сложных правил

2.1. Порождение первичных правил

Первичными правилами называются правила транскрипции, получаемые с помощью простых операций, совершаемых на первом этапе. Порождение первичных правил основывается на нескольких предположениях.

Суть практической транскрипции состоит в том, чтобы передать слово исходного языка на целевой язык с сохранением фонетического облика. То есть, в идеальном случае слово на языке-источнике и языке-приемнике должны состоять из одинаковых последовательностей фонем. Зачастую это невозможно в силу того, что наборы фонем в разных языках различны. В этом случае фонема исходного языка, отсутствующая в целевом языке, передается ближайшей по звучанию фонемой. Например, отсутствующий в японском языке звук [l] при переводе на японский иностранных имен и терминов передается с помощью звука [r], потому что это единственный в японском сонорный неносовой согласный. Фонемный состав

оригинала и перевода слова не всегда совпадает, однако можно с уверенностью сказать, что согласные фонемы языка-источника передаются согласными фонемами языка-приемника, а гласные фонемы – гласными.

Исключение составляют, пожалуй, только аппроксиманты (или глайды) – подтип сонорных согласных, при образовании которых речевой тракт принимает промежуточное положение между положением для образования гласных и согласных шумных звуков. Они считаются согласными, так как при их произнесении все же образуется шум, и преграды в речевом тракте мешают аппроксимантам достигнуть уровня звучности гласных звуков. Тем не менее, звуковые колебания у этих звуков более интенсивны, чем у остальных согласных [2]. Поэтому нередко, если в языке-приемнике отсутствует аппроксимант, который есть в языке-источнике, этот аппроксимант может передаваться как согласной буквой, так и гласной. Например, английский звук [w] при записи английских имен создает неоднозначности транскрипции, разрешавшиеся по-разному в зависимости от господствующей традиции перевода: «Вильям» или «Уильям», «Ватсон» или «Уотсон» и пр. Но поскольку глайды составляют довольно небольшой процент фонем языка, на этапе порождения первичных правил ими можно пренебречь.

Далее предположим, что в большинстве случаев гласный звук передается одной или несколькими гласными буквами, согласный звук – одной или несколькими согласными буквами. А поскольку ранее мы уже предположили, что каждой фонеме исходного языка соответствует фонема целевого языка того же типа (гласная или согласная), можно сделать вывод, что каждой группе согласных букв в оригинале слова соответствует группа гласных букв в переводе слова, группе гласных букв – группа гласных. На этом предположении и основывается первый этап порождения правил.

Для каждой буквы $v \in \mathbf{V_I} \cup \mathbf{V_O}$ определим предикат isVowel(v), который возвращает **true**, если буква является гласной и **false** в противном случае (таким образом, буквы русского алфавита «ь» и «ъ», не обозначающие никакого звука, а потому не относимые ни к гласным, ни к согласным буквам, а также символы начала и конца слова будут также рассматриваться как согласные).

Разделим каждое имя из обучающего множества и его перевод на группы гласных и согласных. Каждая группа должна содержать буквы одного типа. Границы групп в слове $\tau = v_1 v_2 \ldots v_n$ находятся между v_i и v_{i+1} такими, что isVowel(v_i) ≠ isVowel(v_{i+1}).

Таким образом, каждое имя может быть представлено как пара $\mathbf{q} = \langle \mathbf{in}, \mathbf{out} \rangle$, где

$\mathbf{in} = \{in_1, in_2, \ldots, in_n\}$ – множество непустых цепочек букв из алфавита $\mathbf{V_I}$, множество групп оригинала имени;

$\mathbf{out} = \{o_1, o_2, \ldots, o_m\}$ – множество непустых цепочек букв из алфавита $\mathbf{V_O}$, множество групп перевода имени.

Для каждой пары **q** из обучающего множества, для которого количество групп в оригинале и переводе имени совпадают, порождается n правил транскрипции ($n = |\mathbf{in}| = |\mathbf{out}|$): $\mathbf{r}_i(\mathbf{p_l}) = \mathbf{r}_i(\mathbf{p_r}) = \varnothing$, $\mathbf{r}_i(\alpha) = in_i$, $\mathbf{r}_i(\beta) = o_i$, $i \in [1, n]$.

Иными словами, i-я группа оригинала имени ставится в соответствие i-й группе перевода, при соблюдении двух условий: (1) в оригинале и переводе одинаковое количество групп (2) в 1-й группе оригинала содержатся буквы того же типа, что и в i-й группе перевода. Подобная операция возможна, так как справедливо

Утверждение 1.

Для возможности генерации первичных правил из пары **q** необходимо и достаточно, чтобы $|\mathbf{in}| = |\mathbf{out}|$ и isVowel(v_1) = isVowel(u_1), $v_1 \in in_1$, $u_1 \in o_1$.

Доказательство.

Под возможностью использования пары **q** понимается совпадение типов букв всех соответствующих групп из оригинала и перевода, то есть выполнение условия isVowel(v_1) = isVowel(u_1), $v_1 \in in_i$, $u_1 \in o_j$, $i = 1,\ldots, |in|$, $j = 1,\ldots, |out|$.

Через isVowel(in_x) будем обозначать тип букв, составляющих группу in_x, через isVowel(o_x) – тип групп, составляющих o_x.

По определению операции разделения слова на группы гласных и группы согласных для любого in_i из **in** isVowel(in_i) ≠ isVowel(in_{i-1}) и isVowel(in_i) ≠ isVowel(in_{i+1}), то есть для любой группы тип составляющих ее букв не равен типу букв, составляющих соседние группы. Для любой группы o_i из **out** это утверждение также верно.

Таким образом, если isVowel(in_i) = isVowel(o_i), то isVowel(in_{i+1}) = isVowel(o_{i+1}).

Из этого следует, что если в данном слове w выполняется равенство isVowel(in_1) = isVowel(o_1), то для него верно также, что isVowel(in_i) = isVowel(o_i), где i = 1,…, n, n – мощность меньшего из множеств **in** и **out**. Если же |**in**| = |**out**|, то n = |**in**| = |**out**|, а значит, условие isVowel(in_i) = isVowel(o_i) выполняется для всех in_i из **in** и o_i из **out**. Невыполнение начальных условий приводит к невозможности генерации правил на основе пары **q**. Таким образом, данные условия являются необходимыми. Пара **q** может не отвечать дополнительным условиям, то есть, начальные условия являются достаточными. □

Применение данного утверждения на практике позволяет существенно сэкономить вычислительные затраты при нахождении кандидатов в правила, так как нет необходимости проверять соответствие типов букв во всех группах пары **q**.

Неравное количество групп в оригинале и переводе означает наличие нечитаемой буквы или сочетания букв (например, Kate → Кейт, Atheret → Атере). Если в обучающем множестве содержится большое количество слов с неравным количеством групп, это может снизить качество обучения на первом этапе. В этом случае слово также может участвовать в порождении правил, пока соблюдено условие (2).

Образованные таким образом правила составляют множество **R**, множество кандидатов в правила (см. пример 1).

Пример 1. Пример порождения первичных правил.

R | u | gg | ie | r | o M | a | cch | i
R | u | dzh | e | r | o M | a | kk | i
r → r, u → u, gg → dzh, ie → e, o → o, m → m, a → a, cch → kk, i → i

Границы групп обозначены вертикальными линиями. Соответствующие группы из оригинала и перевода имени объединяются в правила.

На данном этапе множество **R** не может использоваться в качестве системы правил транскрипции, так как оно избыточно и неоднозначно. Под избыточностью понимается существование для строки S_I на языке оригинала и ее перевода S_O более одной последовательности правил r_1, …, r_n ∈ **R** таких, что $S_I = r_1(\alpha) + r_2(\alpha) + \ldots + r_n(\alpha)$ и $S_O = r_1(\beta) + r_2(\beta) + \ldots + r_n(\beta)$ (см. пример 2). Неоднозначной система правил называется, если для строки S_I на языке оригинала существует более одной строки S_O на языке перевода, которая может быть получена применением к S_I правил из **R** (см. пример 3).

Пример 2. Избыточность системы правил.

Возьмем систему правил шведско-русской транскрипции R = {a→a, l→л, e→e, rs→ш, rst→шт, t→т, i→и, g→г}. Эта система избыточна, так как для слова Allerstig → Аллештиг из обучающего множества существует две последовательности правил, с помощью которых оригинал может быть преобразован в перевод: K_1 = {a→a, l→л, e→e, rs→ш, t→т, i→и, g→г} и K_2 = {a→a, l→л, e→e, rst→шт, i→и, g→г}, $K_1, K_2 \subseteq R$.

Пример 3. Неоднозначность системы правил.

Возьмем систему правил французско-русской транскрипции R = { a→a, l→л, l→ль, m→м, e→e, d→д }. Эта система неоднозначна, так как для слова Almeda из обучающего множества существует два варианта перевода: Альмеда и Алмеда, – которые могут быть

получены применением к оригиналу последовательностей правил K_1 = { a→a, l→ль, m→м, e→e, d→д } и K_2 = { a→a, l→л, m→м, e→e, d→д } соответственно, $K_1, K_2 \subseteq R$.

Для избавления от избыточности из **R** удаляются следующие правила:

- Редкие правила – правила, встретившиеся в обучающем множестве менее χ раз. Велика вероятность того, что выделенное соответствие подстрок оригинала и перевода является исключением из правил или ошибкой эксперта. χ – параметр, который может быть настроен в зависимости от объема обучающего множества, здесь примем $\chi = 3$;
- Правила, которые могут быть объяснены с помощью других правил. Правило $\mathbf{r_0}$ будет удалено, если найдутся такие $r_1, \ldots, r_n \in \mathbf{R}$, что $\mathbf{r_0}(\alpha) = \mathbf{r}_1(\alpha) + \mathbf{r}_2(\alpha) + \ldots + \mathbf{r}_n(\alpha)$ и $\mathbf{r_0}(\beta) = \mathbf{r}_1(\beta) + \mathbf{r}_2(\beta) + \ldots + \mathbf{r}_n(\beta)$;
- Слишком длинные правила – правила, левая часть которых состоит более чем из ψ символов (здесь $\psi = 3$). Скорее всего, такое правило может быть объяснено с помощью других правил, или будет объяснено на следующих этапах обучения. Исключение составляют правила, правая часть которых состоит из одного символа – они не могут быть разложены на более мелкие правила. Например, при сокращении множества **R** правило «ouia → уиа» будет удалено, правило «tsch → ч» – не будет.

Для избавления **R** от неоднозначностей, то есть пар правил $\mathbf{r_1}$ и $\mathbf{r_2}$ таких, что $\mathbf{r_1}(\mathbf{p_2}) = \mathbf{r_2}(\mathbf{p_2})$, $\mathbf{r_1}(c) \neq \mathbf{r_2}(c)$ $\mathbf{r_1}(\mathbf{p_1}) = \mathbf{r_1}(\mathbf{p_3}) = \mathbf{r_2}(\mathbf{p_1}) = \mathbf{r_2}(\mathbf{p_3}) = \varnothing$. Для избавления системы правил от неоднозначностей в неоднозначные правила вводятся контексты. Система правил неоднозначна, если в ней присутствует хотя бы одна пара правил $\mathbf{r_1}$ и $\mathbf{r_2}$ таких, что $\mathbf{r_1}(\alpha) = \mathbf{r_2}(\alpha)$, $\mathbf{r_1}(\beta) \neq \mathbf{r_2}(\beta)$ $\mathbf{r_1}(\mathbf{p_l}) = \mathbf{r_1}(\mathbf{p_r}) = \mathbf{r_2}(\mathbf{p_l}) = \mathbf{r_2}(\mathbf{p_r}) = \varnothing$. Появление таких правил может быть вызвано неоднозначностями правил чтения входного языка, но нередко правила чтения для данной буквы зависят от ее позиции в слове и соседних букв, и неоднозначности можно разрешить включением в правила контекстов (см. пример 2). Контекстами в данном случае называются буквы, которые предшествуют ($\mathbf{p_l}$, левый контекст) строке α правила в словах, где встречается это правило, или следуют за ней ($\mathbf{p_r}$, правый контекст).

2.2. Порождение сложных правил

Система правил, порожденная на первом этапе, для многих обучающих множеств является полной. Однако существует несколько типов правил, которые не могут быть сгенерированы с помощью разделения слова на группы гласных и согласных и их анализа:

- Правила, строки α и/или β которых состоят из букв разных типов (гласных и согласных). Например, правило qu → к для французско-русской транскрипции;
- Правила для непроизносимых букв в конце или в начале слова.

Порождение таких правил осуществляется путем анализа слов обучающего множества с помощью уже существующей системы правил. Этап порождения сложных правил можно разделить на следующие подэтапы:

1. Разделение слов из обучающего множества на слоги;
2. Пробный разбор слогов, порождение правил;
3. «Склеивание» неразобранных слогов;
4. Возврат к пункту 2.

Разделение слов на слоги

Термин «слог» в данной работе используется не в классическом лингвистическом значении минимальной единицы речевого потока. Слог в данной работе – подцепочка слова, границы которой определяются по следующим правилам:

- Граница слога в слове $w = l_1 l_2 \ldots l_n$ ставится между гласной и согласной, то есть после буквы l_i такой, что isVowel(l_i) = true и isVowel(l_{i+1}) = false;
- цепочка букв, среди которых нет гласной, не выделяется в отдельный слог, а присоединяется к предыдущему;

Таким образом, слог – это группа согласных и следующая за ней группа гласных. За группой гласных может следовать группа согласных, если ею заканчивается слово. Слог может состоять также и из одной группы гласных, если это первая группа слова. Но наиболее частотная форма слога – группа гласных и группа согласных.

Слова обучающего корпуса делятся на слоги, так как слоги меньше целых слов, и при работе с ними легче выделить соответствия букв входного и выходного алфавитов.

Пробный разбор и порождение правил

На первом этапе алгоритма составляется система правил **R**, которая уже может быть использована для преобразования строк, пусть и не всегда результат ее применения к строке будет корректен с точки зрения правил чтения языка оригинала. Эта система правил также используется и самим алгоритмом для порождения новых правил. Проводится пробный разбор слогов по правилам из **R** для выявления подстрок, не объясняемых системой правил.

Рассмотрим слог и его перевод $\lambda \rightarrow \mu$, где $\lambda = \langle u_1, \ldots, u_n \rangle$, $\mu = \langle v_1, \ldots, v_m \rangle$, $\lambda \in \mathbf{V_I}^+$, $\mu \in \mathbf{V_O}^+$. Требуется найти последовательность правил транскрипции $K = \langle r_1,\ldots,r_k \rangle$, принадлежащих системе правил **R**, таких, что последовательным применением K к строке λ можно получить строку μ. В случае, если **R** не объясняет данный слог, требуется показать, что такой последовательности K для данного слога не существует.

Пробный разбор осуществляется следующим образом. Сначала слог разбирается слева направо. Ищется правило, левая часть которого совпадает с начальной подстрокой строки λ длины i, а правая часть – с начальной подстрокой строки μ длины j (где i и j – длина левой и правой частей правила соответственно) и не существует правила с левой частью длиннее i, удовлетворяющего тому же условию. От строки λ «отрезается» начальная подстрока длины i, от строки μ – начальная подстрока длины j и разбор цепочек λ и μ продолжается с i+1-го и j+1-го символов соответственно.

Если достигнут конец цепочек λ и μ, значит, слог может быть объяснен системой правил R. В этом случае он больше не используется алгоритмом, так как с его помощью не удастся породить новых правил. В противном случае слог аналогично разбирается справа налево: ищется самое длинное правило, левая часть которого совпадает с конечной подстрокой λ, правая – с конечной подстрокой μ.

Если обозначить через $u_1, \ldots, u_n$ и $v_1,\ldots,v_m$ символы, объясненные правилами, а через $\boldsymbol{\lambda_x}$ и $\boldsymbol{\mu_x}$ – подстроки оригинала и перевода слова соответственно, не объясненные правилами, слог, не полностью объясненный существующими правилами, может быть записан одним из четырех способов:

- $\langle u_1,\ldots,u_i, \boldsymbol{\lambda_x} \rangle \rightarrow \langle v_1,\ldots,v_j, \boldsymbol{\mu_x} \rangle$ – слог был частично разобран слева;
- $\langle \boldsymbol{\lambda_x}, u_{i+1},\ldots, u_n \rangle \rightarrow \langle \boldsymbol{\mu_x}, v_{j+1},\ldots, v_m \rangle$ – слог был частично разобран справа;
- $\langle u_1,\ldots,u_i, \boldsymbol{\lambda_x}, u_{i+1},\ldots, u_n \rangle \rightarrow \langle v_1,\ldots,v_j, \boldsymbol{\mu_x}, v_{j+1},\ldots, v_m \rangle$ – слог был частично разобран справа и слева;
- $\langle \boldsymbol{\lambda_x} \rangle \rightarrow \langle \boldsymbol{\mu_x} \rangle$ – слог не был разобран.

Последовательность правил, применяемых к данной строке, создает разбиение этой строки на подстроки, каждая из которых является левой частью какого-либо правила. Если слог не был разобран целиком, значит, какие-то из правил, создающих разбиение этого слога, отсутствуют в системе правил. Если же слог был разобран только с одной стороны, то, возможно, границы слогов в слове, из которого был взят данный слог, не совпадают с границами правил, с помощью которых можно осуществить корректный перевод этого слова (см. пример 4).

Пример 4. Слово, в котором границы правил и слогов не совпадают. В первой строке показано разделение слова на слоги. Во второй – границы применяемых правил транслитерации: m→м, y→и, r→р, ill→ий, a→а. Одно из правил (ill→ий) находится на границе двух слогов: и левая, и правая часть его делятся границей слога на две части. Слоги этого слова не могут быть корректно преобразованы по отдельности.

М у|r i|l l а м и|р и|й а
М|у|r|i l l|а м|и|р|и й|а

По этой причине слоги, разобранные только с одной стороны или не разобранные вообще, не участвуют в порождении правил на данном этапе.

Слог же, частично разобранный с двух сторон, как уже говорилось, имеет форму $<u_1,\ldots,u_i, \boldsymbol{\lambda_x}, u_{i+1},\ldots, u_n> \rightarrow <v_1,\ldots,v_j, \boldsymbol{\mu_x}, v_{j+1},\ldots, v_m>$, где $\boldsymbol{\lambda_x} \rightarrow \boldsymbol{\mu_x}$ – подстрока с переводом, не удовлетворяющая ни одному правил системы **R**.

Можно выделить три случая несоответствия слога правилам:

- $\boldsymbol{\lambda_x} = \varnothing$, $\boldsymbol{\mu_x} \neq \varnothing$. В систему **R** добавляется новое правило r_k такое что $r_k(\mathbf{p_l}) = u_{i-1}$, $r_k(\alpha) = u_i$, $r_k(\mathbf{p_r}) = u_{i+1}$, $r_i(\beta) = v_j + \boldsymbol{\mu_x}$, другими словами, правило транслитерации для символа, предшествующего $\boldsymbol{\lambda_x}$, который в некоторых контекстах передается двумя или более символами.
- $\boldsymbol{\lambda_x} \neq \varnothing$, $\boldsymbol{\mu_x} = \varnothing$. Такая ситуация значит, что некоторые символы из $\mathbf{V_I}$ в определенном контексте не читаются. К системе **R** добавляется правило r_k такое что $r_k(\mathbf{p_l}) = u_i$, $r_k(\alpha) = \boldsymbol{\lambda_x}$, $r_k(\mathbf{p_r}) = u_{i+1}$, $r_k(\beta) = \varnothing$.
- $\boldsymbol{\lambda_x} \neq \varnothing$, $\boldsymbol{\mu_x} \neq \varnothing$. В этом случае добавляется правило r_k такое что $r_k(\mathbf{p_l}) = r_k(\mathbf{p_r}) = \varnothing$, $r_k(\alpha) = \boldsymbol{\lambda_x}$, $r_k(\beta) = \boldsymbol{\mu_x}$. Если в системе **R** есть правило r_q такое что $r_q(\alpha) = \boldsymbol{\lambda_x}$ (то есть, если введение правила r_k в систему нарушает ее однозначность), то к правилу r_k добавляются контексты.

«Склеивание» слогов

В предыдущем разделе было показано, что можно с достаточной уверенностью породить на основе слога, разобранного не целиком, новое правило, только если этот слог частично разобран и справа, и слева. В противном случае существует вероятность того, что правило, которое должно быть применено к данному слогу, распространяется не только на этот слог, но и на соседний.

Необходимым условием того, что границы слога не разрывают правило, надо убедиться, что слог разобран с обеих сторон или что в соседнем слоге нет неразобранной подстроки, примыкающей к данному слогу.

Для этого применяется процедура агглютинации («склеивания») слогов.

Если слог не разобран справа, проверяется следующий за ним слог. Если следующий слог не разобран слева, то два эти слога объединяются в один слог, который будет частично разобран с обеих сторон и на основе которого можно будет породить новое правило. Если следующий слог разобран полностью (или, по крайней мере, разобран слева), значит, граница слога не разрывает правило, и для порождения нового правила достаточно будет текущего слога. Если же следующий слог вообще не разобран, он присоединяется к текущему и проверяются все следующие за ним слоги, пока не будет найден частично разобранный справа или полностью разобранный слог, чтобы процесс склеивания мог быть остановлен.

Заметим, что, поскольку у каждого слова обучающего множества есть маркеры начала и конца слова – «<» и «>», а в систему правил перед началом обучения добавляются правила <→< и >→>, первый слог каждого слова является частично разобранным слева, а последний слог – частично разобранным справа.

После перегруппировки слогов повторяется их пробный разбор и порождение правил.

3. Результаты экспериментов

3.1. Описание обучающих данных

Метод порождения правил транскрипции для обучения требует параллельного корпуса имен на исходном и целевом языках. Во всех базах, использованных для экспериментов,

целевым языком был русский. Базы были составлены вручную экспертами-лингвистами. Использованные базы перечислены в таблице 1.

Исходный язык	**Объем базы (количество слов)**
Японский	7006
Китайский	4671
Немецкий	4208
Арабский	2109
Шведский	1777
Польский	1435
Испанский	1041
Французский	760
Румынский	576
Словенский	504
Тагальский	293
Монгольский	232
Хинди	161

Таблица 1. Обучающие выборки

Исходным алфавитом всех баз является латинский алфавит без диакритических знаков (исключение составляет немецкий, в котором помимо стандартной латиницы используются символы ä, ö и ü). Их отсутствие объясняется тем, что многие корпуса составлены из имен, взятых из документов машиносчитывамого формата. Отсутствие диакритических знаков часто создает неоднозначность, снижающую качество правил. Например, в шведском языке есть символ «å», который по правилам практической транскрипции должен передаваться на русский язык символом «о». Машиносчитываемый формат документов не допускает диакритических знаков, поэтому при формировании таких документов символ «å» заменяется на сочетание «аа». Исходя из этого при анализе обучающей выборки формируется правило «аа → о». Но в той же выборке присутствует также сочетание «аа», которое не является заменой символа «å», а составлено из двух «а», поэтому и на русский язык должно передаваться как «аа». Это порождает неразрешимую в рамках данной задачи неоднозначность, тогда как на самом деле никакой неоднозначности нет.

В базах языков, пользующихся не латинским алфавитом (японский, китайский, хинди и пр.), использована их латинская транскрипция (для японского – транскрипция по системе Ромадзи, для китайского – по системе Пиньин). Однако латинский алфавит не является принципиальным условием работы системы. В качестве исходного и целевого алфавитов могут быть использованы любые алфавиты. Единственное ограничение метода состоит в том, что письмо должно быть именно алфавитным (возможно, слоговым, как в японском языке), но не иероглифическим.

3.2. Оценка правил транскрипции

На основе имеющихся корпусов были построены системы правил транскрипции с этих языков на русский. Для оценки правил было проведено сравнение систем правил, сгенерированной автоматически по описанному в статье методу, с системами правил, написанных вручную экспертом для системы «Трансскриба» (см. таблицу 2).

Сокращение количества правил может быть объяснено двумя причинами. Во-первых, некоторые правила не были сгенерированы. Подобная ситуация сложилась по нескольким причинам. Некоторые правила генерировались экспертами исходя из знаний языка, но примера на их употребление в коллекцию помещено не было. Так, во французском языке эксперт поместил правило, что «ai» в начале слова переходит в «э», тогда как подобного примера в коллекции не было. Часть правил отсеивалась как встретившиеся только один раз и незначимые. И, наконец, некоторые правила просто не были порождены предлагаемым

методом. Так, например, для арабского языка экспертом было записано правило «yaa → я», тогда как программой были получены правила «aa → а» и «ya → я» (которые также есть у эксперта), а правило «yaa → я» порождено не было. Для японского языка экспертом были записаны слоги «ha», «sha» и «cha», передаваемые как «ха», «ся» и «тя», соответственно. Но в связи с тем, что программой рассматривались только контексты длиной в один символ, было порождено единое правило: «a» с левым контекстом «h» или «j» передается как «я».

Язык	Сгенерировано правил	Правил эксперта
Арабский	63	78
Испанский	88	106
Немецкий	102	121
Монгольский	41	46
Французский	160	356
Шведский	105	423
Японский	52	131

Таблица 2. Сравнение результатов

Во-вторых, при генерации произошло сжатие правил по сравнению с правилами, составленными экспертами. Так. Например, во французском языке экспертами в связи с особенностями применявшегося ранее метода был порожден целый комплект правил, касающихся прочтения сочетания «ai» в различных контекстах, тогда как в данной версии было сгенерировано лишь одно - «ai → е», покрывающее все эти случаи (кроме «ai» в начале слова). Для японского языка слоговая азбука была редуцирована в правила чтения отдельных сочетаний. Так, для трех слогов, начинающихся с «ch» было сгенерировано единое правило «ch → т». Заметим, что подобный подход является корректным, но может оказаться непривычным для лингвиста, корректирующего правила.

В остальном правила являются корректными, покрывают большую часть тестовой базы и позволяют проводить транскрипцию имен собственных для языков без разработанных правил машинной транскрипции.

3.3. Численная оценка результатов

Было оценено качество обучения метода – то есть, насколько полно сгенерированные правила объясняют обучающую выборку. Для этого порожденные правила были применены к обучающей выборке. Результат транскрипции оценивался по нескольким параметрам:

- процент строк, для которых хотя бы один из предложенных вариантов перевода оказался корректным, то есть совпал с настоящим переводом (напомним, что система правил транскрипции почти всегда остается неоднозначной из-за неоднозначностей правил чтения языка оригинала, поэтому для части слов предлагается несколько возможных вариантов перевода); эта мера обозначена как Correct Transliteration (**CT**);
- процент строк, для которых был предложен единственный вариант перевода, оказавшийся корректным; мера обозначена как Unique Correct Transliteration (**UCT**);
- мера неоднозначности перевода – среднее количество выдаваемых системой вариантов транскрипции строки; мера обозначена как Average Transtiteration Variant (**ATV**);

Результаты оценки качества обучения представлены в таблице 3.

Для мер CT и UCT указано количество правильно и однозначно переданных слов соответственно, в скобках указано, какой процент тестовой выборки составили эти слова.

Для оценки полноты построенной системы правил с точки зрения правил чтения данного языка и качества транскрипции необходимо было провести транскрипцию на тестовой

выборке имен – то есть, на параллельном корпусе имен, не использовавшемся при генерации правил. Распространенной практикой является отделение от обучающих данных десятой части и проверка на ней результатов обучения. Однако маленький размер обучающих корпусов для многих языков (напр. хинди, монгольский) не позволил провести такую проверку. Выборка, составляющая десятую часть маленького корпуса, нерепрезентативна, к тому же, ее исключение из обучающего множества может привести к тому, что некоторые правила не будут порождены (например, если какой-либо характерный для данного языка диграф встретился в данных только один раз, и слово, содержащее его, попало в тестовую выборку). Было решено выделить тестовые множества имен только для тех корпусов, в которых больше тысячи слов, так, чтобы тестовая выборка составляла по крайней мере сто слов.

Исходный язык	**CT**	**UCT**	**ATV**
Японский	7005 (**99%**)	4778 (**68%**)	1,38
Китайский	4468 (**95%**)	4173 (**89%**)	1,06
Немецкий	3484 (**82%**)	3247 (**77%**)	1,07
Арабский	2102 (**99%**)	1793 (**85%**)	1,19
Шведский	1576 (**88%**)	905 (**50%**)	1,61
Польский	1424 (**99%**)	1174 (**81%**)	1,2
Испанский	1025 (**98%**)	777 (**74%**)	1,33
Французский	678 (**89%**)	227 (**29%**)	2,66
Румынский	565 (**97%**)	295 (**52%**)	1,78
Словенский	502 (**99%**)	354 (**70%**)	1,3
Тагальский	257 (**87%**)	225 (**76%**)	1,14
Монгольский	231 (**100%**)	227 (**98%**)	1,02
Хинди	160 (**99%**)	152 (**94%**)	1,05

Таблица 3. Оценка качества обучения метода.

При оценке результатов транскрипции тестовой выборки использовались те же меры, что и при оценке качества обучения метода: процент слов, для которых хотя бы один из предложенных вариантов перевода оказался правильным, процент слов, для которых правильный вариант перевода был единственным вариантом, предложенным системой, вероятность правильного перевода.

Кроме этих мер, были введены две новые меры оценки ошибки, основанные на расстоянии Левенштейна. Расстоянием Левенштейна называется количество операций добавления, удаления или замены символа, необходимых для преобразования строки A в строку B [1].

Меры оценки ошибки:

- Среднее нормированное расстояние Левенштейна. Для каждого имени тестовой выборки определяется нормированное расстояние Левенштейна: расстояние Левенштейна, деленное на длину имени. Таким образом, если перевод, предложенный системой, полностью совпадает с фактическим переводом, мера для данного слова будет равна нулю. Заметим также, что мера рассчитывается для каждого варианта перевода, предложенного системой, а затем делится на общее количество вариантов (а не на общее количество строк в тестовой выборке). В таблице обозначено как Average Normalized Levenstein Distance (ANL);
- Среднее расстояние Левенштейна для неправильно переданных слов. Average Error (AE).

Результаты проверки качества транскрипции представлены в таблице 4.

Язык	**Объем тестовой выборки**	**CT**	**UCT**	**ANL**	**AE**
Японский	701	664 (**94%**)	496 (**70%**)	0,043	1,125
Китайский	468	436 (**93%**)	406 (**86%**)	0,026	1,23
Немецкий	421	339 (**80%**)	325 (**77%**)	0,032	1,176
Арабский	211	169 (**80%**)	160 (**75%**)	0,054	1,5
Шведский	178	149 (**83%**)	82 (**46%**)	0,131	1,33
Польский	144	138 (**95%**)	123 (**85%**)	0,027	1,5
Испанский	105	99 (**94%**)	76 (**72%**)	0,038	1,125

Таблица 4. Оценка качества транскрипции.

4. Анализ результатов

Предложенный метод обладает некоторыми недостатками, ухудшающими качество транскрипции:

- Низкая устойчивость к ошибкам в тестовой выборке и исключениям из правил
- Неполное использование контекстов
  - Не используется информация о закрытости / открытости слога, номер слога в слове
- Не рассматривается морфологический уровень слова – префиксы, суффиксы, слова с несколькими корнями
  - На стыке морфем могут измениться правила чтения устойчивых сочетаний букв:
    - Sherlock – Шерлок
    - Bellshouse – Беллзхаус

Для избавления от первого недостатка в дальнейших работах планируется более полно использовать статистическую информацию. Каждое правило может быть оценено по частоте его встречаемости, правила с очень низкой встречаемостью могут быть удалены из системы или помечены как исключения, что при достаточно большом количестве обучающих данных может помочь исправить ошибки в самих данных. Если каждому правилу будет присвоен вес, при выдаче нескольких вариантов транскрипции самый вероятный (то есть, полученный применением более частотных правил) может быть помещен в начало списка, что повысит информативность результата для пользователя.

А более полное использование контекстов при формировании правил уменьшит неоднозначность в системе, хотя и не снимет ее, так как неоднозначность часто является характеристикой самого языка.

Последний же недостаток является не недостатком, а скорее одним из побочных эффектов условий поставленной задачи: метод должен порождать правила транскрипции для любой пары языков, имея минимум информации об этих языках (единственная доступная информация – алфавиты исходного и целевого языков и списки гласных букв этих языков). Введение же морфологической информации, такой как характерные для языка суффиксы, существенно усложнит и подготовку обучающих данных, и сам процесс обучения.

Были проведены эксперименты, показавшие, что правила, порожденные методом, покрывают обучающую выборку в среднем на 95%. Испытания на тестовой выборке также показали высокий результат – от 80% до 95% для разных языков. Этот результат по качеству приближается к результатам обучения с помощью статистических методов на довольно больших базах, тогда как в настоящей работе использовались небольшие обучающие корпуса.

5. Процедура преобразования строки с помощью системы правил

Будем говорить, что правило **r** *применимо* к строке с текущей позиции, если на текущей позиции находится подстрока **r**(α), перед ней полностью представлена одна из подстрок из $\mathbf{r}(\mathbf{p}_l)$, после нее – подстрока из $\mathbf{r}(\mathbf{p}_r)$.

Определим процедуру преобразования строки с помощью системы правил как функцию $\psi = TrR(\mathbf{R}, \varphi)$, где **R** – система правил транскрипции с языка L_1 на язык L_2, φ – цепочка символов алфавита языка L_1. Функция TrR возвращает цепочку ψ символов алфавита языка L_2. Преобразование строки с помощью системы правил **R** производится следующим образом:

1. Текущей позицией входной строки назначается первый символ.
2. Формируется пустая выходная строка.
3. В системе правил **R** ищутся все правила, применимые к строке с данной позиции.
4. Из найденных правил выбирается правило с самой длинной левой частью. Если таких правил более одного, формируется более одной выходной строки (по количеству найденных правил). Если не найдено ни одного правила, текущий символ без изменения в обрамлении специальных символов, обозначающих отсутствие перевода (например, символов подчеркивания «_») приписывается ко всем выходным строкам, текущая позиция сдвигается на 1.
5. Ко всем выходным строкам присоединяется подстрока β найденного правила. Если существует более одной выходной строки и при этом было найдено более одного правила, множеством выходных строк после данного шага будет множество конкатенаций всех пар из декартова произведения существующих выходных строк и строк β всех найденных правил (см. пример 5).
6. Текущая позиция сдвигается на $|\alpha|$ символов вправо.
7. Пункты 3-6 повторяются, пока не достигнут конец слова.

Пример 5. Поиск множества выходных строк в случае неопределенности.

Пусть существует множество уже порожденных выходных строк {«а», «о»} и множество правых частей применяемых правил {«б», «в»}. К каждой строке из первого множества должна быть добавлена каждая строка из второго множества. Таким образом, результирующее множество выходных строк будет равно {«аб», «ав», «об», «ов»}.

Будем считать, что системы правил $\mathbf{R_1}$ и $\mathbf{R_2}$ эквивалентны, если для любой строки φ языка L_1 $TrR(\mathbf{R_1}, \varphi) = TrR(\mathbf{R_2}, \varphi)$.

6. Структура конечного автомата

Конечные автоматы (КА) часто используются в задачах обработки текстов на естественном языке, так как обеспечивают линейную скорость обработки строк. Среди прочего, применяются они и для решения задачи машинной транскрипции [3]. В практических приложениях используются конечные автоматы двух типов: распознающие и преобразующие. Распознающие конечные автоматы по данной строке определяют, принадлежит ли она заданному языку. На вход подается строка, разбор начинается с первого символа строки из начального состояния автомата. По каждому следующему символу осуществляется переход в очередное состояние конечного автомата. Если из данного состояния не существует перехода по текущему символу, или достигнут конец строки, когда автомат находится не в конечном состоянии, строка не принадлежит языку, который распознает данный автомат. Если же по достижении конца строки текущим состоянием автомата является конечное состояние, строка принадлежит языку.

Преобразующие КА помимо продвижения вперед по строке при переходе в новое состояние осуществляют, например, формирование выходной строки или преобразование текущей. Таким образом, они возвращают не только ответ на вопрос, является ли данная строка словом некоторого языка, но и новую строку.

Предлагаемый в данной работе КА является расширенным. Он представляет собой преобразователь, то есть, получая на вход строку, формирует на ее основе выходную строку в соответствии с правилами, на основе которых построен. Расширенным же он является потому, что, в отличие от классических КА, может осуществлять движение по строке не на один символ, а на несколько, причем в любом направлении. Подобный КА можно представить как множество $\mathbf{g} = <\mathbf{V_I}, \mathbf{V_O}, \mathbf{Q}, q_0, \mathbf{F}, \theta>$, где:

$\mathbf{V_I}$ – входной алфавит (алфавит языка оригинала);
$\mathbf{V_O}$ – выходной алфавит (алфавит языка перевода);
Q – множество состояний автомата;
$q_0 \in \mathbf{Q}$ – начальное состояние автомата;
$\mathbf{F} \subset \mathbf{Q}$ – множество конечных состояний;
θ – функция переходов $\mathbf{Q} \times \mathbf{V_I} \rightarrow <q, n, \mathbf{A}>$.

Здесь $q \in \mathbf{Q}$ – это состояние, в которое должен быть произведен переход, $n \in \mathbb{Z}$ – количество символов, на которое должен быть произведен сдвиг по разбираемой строке при переходе, $\mathbf{A} \subset \{\mathbf{V_O}^*\}$ – множество выходных строк, приписанных переходу.

Из каждого конечного состояния существует переход в начальное состояние по пустому символу.

7. Процедура преобразования строки с помощью конечного автомата

Определим функцию преобразования строки с помощью конечного автомата $\psi = TrA(\mathbf{g}, \varphi)$, где **g** – конечный автомат, предназначенный для транскрипции имен с языка L_1 на язык L_2, φ – цепочка символов алфавита языка L_1. Функция TrA возвращает цепочку ψ символов алфавита языка L_2.

Транскрипция с помощью конечного автомата осуществляется следующим образом:

1. Текущим символом *l* назначается первый символ входной строки, текущим состоянием q назначается начальное состояние q_0.
2. Вычисляется значение функции переходов $\theta(q, l) \rightarrow <q', n, \mathbf{A}>$. Из текущего состояния производится переход в состояние q'. При переходе производится сдвиг по разбираемой строке на *n* символов.
    a. Если $q' \in \mathbf{F}$, то есть, достигнуто конечное состояние, происходит переход по пустому символу в начальное состояние. То есть $\forall f \in \mathbf{F}, \theta(f, \varepsilon) = <q_0, 0, \varnothing>$.
    b. Если $A \neq \varnothing$ (это возможно только если $q' \in \mathbf{F}$), следует обновить результат транскрипции: он составит множество конкатенаций всех пар из декартова произведения существующих выходных строк и выходных строк, приписанных к данному переходу (аналогично с декартовым произведением из описания функции TrR).
    c. Если для пары <q, *l*> функция θ не определена, символ *l* присоединяется к выходной строке (или строкам) без изменения в обрамлении специальных символов, обозначающих отсутствие перевода (аналогично определению функции TrR), и осуществляется переход в q_0.
3. Пункт 2 повторяется, пока не достигнут конец слова.

В принципе, при разборе не важно, оказалось ли текущим состоянием конечное в тот момент, когда был достигнут последний символ строки. Но если этого не произошло, то для последних символов не была возвращена выходная подстрока, а значит, следует подать на выход оставшуюся часть входной цепочки с обрамлением из специальных символов.

8. Построение конечного автомата

Конечный автомат строится на основе системы правил. Каждое правило преобразуется в цепочку переходов КА. Первым состоянием каждой такой цепочки является единственное начальное состояние автомата, последнее же состояние цепочки – конечное. Из каждого конечного состояния проводится переход по пустому символу в начальное состояние. Таким образом, переход по цепочке от начального состояния к конечному при разборе слова равносилен применению правила транскрипции. После применения правила автомат возвращается в начальное состояние – то есть, готов применить следующее правило.

8.1. Преобразование правила транскрипции в конечный автомат

Правило можно записать как $<\mathbf{p_l}, \alpha, \mathbf{p_r}> \rightarrow \beta$, где α – входная строка, β – выходная строка, замещающая строку α при переводе, $\mathbf{p_l} = \{\gamma\}$ и $\mathbf{p_r} = \{\delta\}$ – контексты. Обозначим строку α как последовательность символов строки $\alpha = <l_0 \ldots l_n>$.

Преобразование правила к конечному автомату производится следующим образом.

1. При наличии левого контекста:
   a. Определяется значение функции $\theta(q_0, l_0) = <q_1, -|\gamma|, \varnothing>$, где l_0 – первый символ α, $\gamma \in \mathbf{p_l}$, q_1 – новое состояние.
   b. Для каждой строки $\gamma = <t_1 \ldots t_k>$ левого контекста создаются пути такие, что $\forall\gamma \in \mathbf{p_l} \rightarrow \theta(q_i, t_i) = <q_{i+1}, 1, \varnothing>$, $1 \le i \le k$, причем все пути должны начинаться в состоянии q_1 и заканчиваться в состоянии q_k, но их промежуточные состояния q_j, $1<j<k$, должны быть различными.
2. При отсутствии левого контекста $q_k = q_0$.
3. Создается путь для строки α: $\forall l_i \in \alpha \rightarrow \theta(q_{k+i}, l_i) = <q_{k+i+1}, 1, \varnothing>$, $0 \le i < n$. При наличии правого контекста $\theta(q_{k+n}, l_n) = <q_{k+n+1}, 1, \varnothing>$, при отсутствии правого контекста $\theta(q_{k+n}, l_n) = <q_f, 1, \beta>$, $q_f \in \mathbf{F}$.
4. При наличии правого контекста создаются пути для каждого правого контекста: $\forall\delta \in \mathbf{p_r}$, $\delta = <s_1 \ldots s_m> \rightarrow \theta(q_{k+n+i+1}, s_i) = <q_{k+n+i+2}, 1, \varnothing>$, $1 \le i < m$, $\theta(q_{k+n+m}, s_m) = <q_f, -|\delta|+1, \beta>$, $q_f \in \mathbf{F}$. Аналогично с процедурой создания путей для левых контекстов промежуточные состояния путей для разных δ должны быть различны.
5. $\forall f \in \mathbf{F}$, $\theta(f, \varepsilon) = <q_0, 0, \varnothing>$

Таким образом, происходит проверка первого символа из α, возврат назад по цепочке на длину левого контекста, проверка левого контекста, проверка строки α, проверка правого контекста и возврат назад к символу, следующему за последним символом α с возвратом в начальное состояние.

Примеры генерации цепочек переходов в КА можно видеть на рис. 1 и рис. 2.

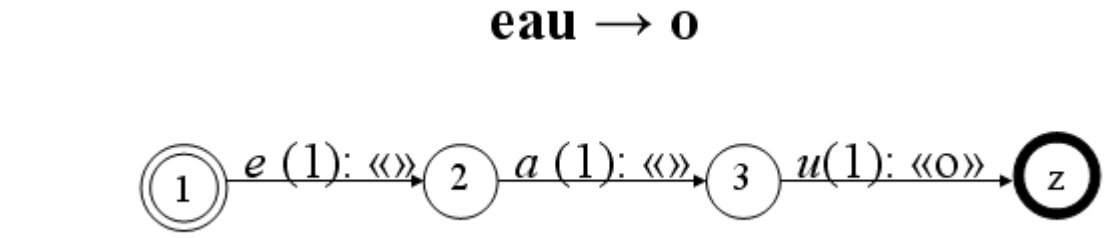

Рис. 1. Преобразование правила без контекста «eau→o» к конечному автомату

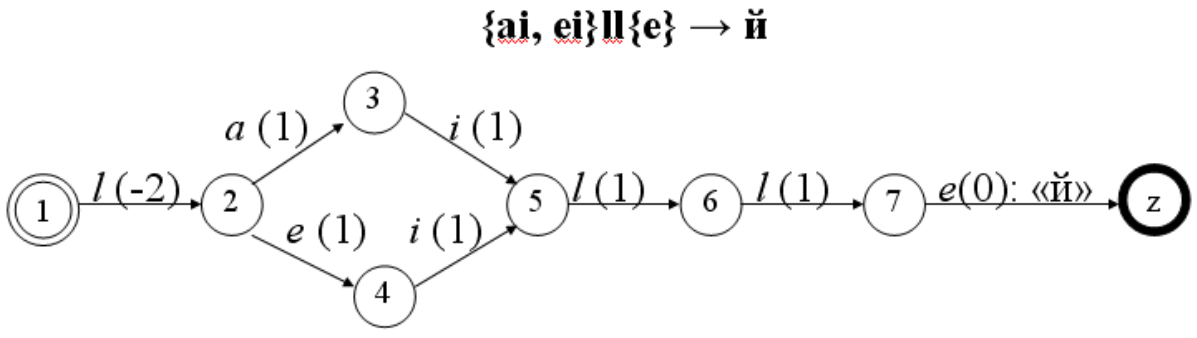

Рис 2. Преобразование правила с контекстом {ai, ei}ll{e} → й к конечному автомату

Заметим, что для корректного преобразования правила с контекстом к КА все цепочки в каждом из контекстов должны иметь одинаковую длину. При этом цепочки из левого

контекста не обязаны быть такой же длины, как цепочки правого контекста. Кроме того, правила должны быть проверены на предмет непротиворечивости, то есть не должно существовать правил $\mathbf{r}_1$ и $\mathbf{r}_2$, таких, что $\mathbf{r}_1(\alpha)$ является префиксом $\mathbf{r}_2(\alpha) + \mathbf{r}_2(\mathbf{p}_r)$.

Заметим также, что контексты являются полностью независимыми друг от друга: не предъявляется требования к существованию обоих контекстов. В системе правил вполне допустимы правила вида $\mathbf{p}_l\ \alpha\rightarrow\beta$, $\alpha\ \mathbf{p}_r\rightarrow\beta$ или $\alpha\rightarrow\beta$, то есть без правого, без левого контекста или без обоих контекстов.

8.2. Детерминированный конечный автомат

Конечный автомат является детерминированным (ДКА), когда для каждой входной цепочки символов существует только одна последовательность переходов от начального состояния до конечного. Это значит, что для каждого состояния КА должно существовать не более одного перехода по каждому из символов входного алфавита. При построении конечного автомата согласно процедуре, описанной в предыдущем разделе, это правило может быть нарушено. Из состояния q_0 существуют переходы по первым символам строки α каждого из правил. Если в системе правил содержится хотя бы одна пара правил, у которых совпадают первые символы строк α, то автомат, построенный на основе этой системы, будет недетерминированным (НКА) (см. рис. 3). Разбор строк может быть проведен и с помощью недетерминированного конечного автомата, но будет потеряно главное преимущество КА – линейная скорость обработки строк.

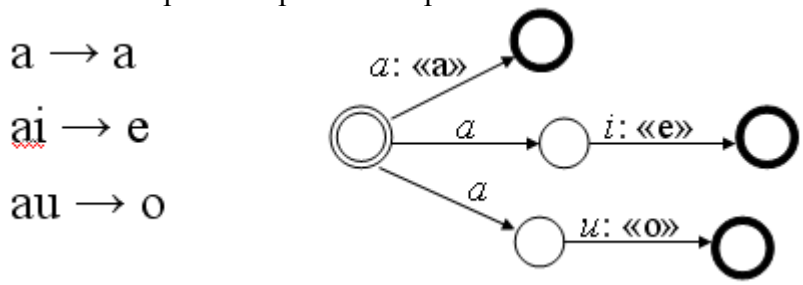

Рис 3. НКА, построенный на основе системы правил {a→a, ai→e, au→o}

Недетерминированность конечного автомата следует отличать от неоднозначности самой системы правил, на основе которой он строится. Под неоднозначностью понимается существование для строки φ на языке оригинала более одной строки ψ на языке перевода, которая может быть получена применением к φ правил из **R**.

Если система правил неоднозначна, в результате преобразования некоторых строк с ее помощью может получиться более одной выходной строки. Однако недетерминированность результата транскрипции вовсе не означает недетерминированности конечного автомата. Верно и обратное: из однозначной системы правил может быть получен НКА. Ниже будет показано, что результатом работы используемого нами КА может быть множество строк, причем автомат будет детерминированным. Также будет приведена процедура преобразования расширенного КА к детерминированному виду.

8.3. Процедура преобразования НКА в ДКА

В целом процедура преобразования НКА в ДКА, используемая в данной системе, стандартна и описана, например в [5], но из-за специфических особенностей расширенного конечного автомата она была немного изменена (см. рис. 4).

Для каждой вершины q_1, из которой исходит n-1 дуг в вершины $q_2,\ldots,q_n$, помеченных одним символом:

- Создается новая вершина q', в которую входит дуга из вершины q_1, помеченная тем же символом;
- Из вершины q' проводятся все дуги, которые выходили из вершин $q_2,\ldots,q_n$;
- Вершины $q_2,\ldots, q_n$ удаляются.

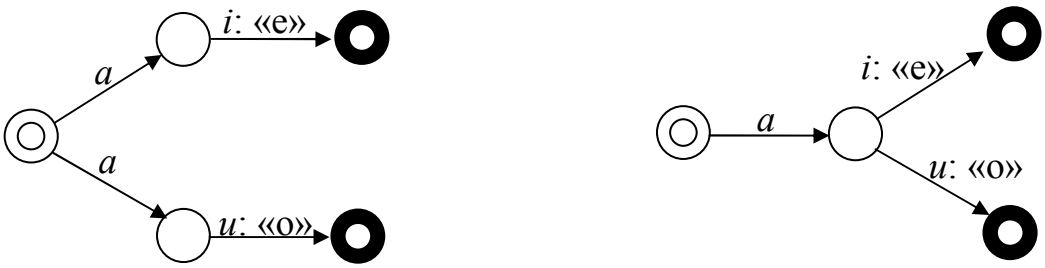

Рис 4. НКА (слева) и результат его приведения к детерминированному виду

Отметим, что в приведенной процедуре, в отличие от стандартной, не происходит удаления е-дуг (переходов по пустому символу), так как они являются аналогом перехода от одного правила к другому и необходимы для возвращения к начальному состоянию, с которого начинаются все правила.

Следует также отметить, что для конечного автомата, построенного на основе описанной выше системы правил, такая процедура преобразования к ДКА не вполне подходит. Если требуется объединить два перехода, одному из которых приписан кортеж <q, *l*, $\mathbf{A_1}$>, предполагающий переход по разбираемой строке на 1 символ вправо, а другому – <q, *-l*, $\mathbf{A_2}$> – переход на один символ влево, не ясно, какое значение *n* должно быть приписано новому объединенному переходу (см. пример 6).

Пример 6. Недетерминированный конечный автомат, не упрощаемый при помощи стандартной процедуры

На рисунке 5 показана ситуация, когда из одного состояния (начальное, на рисунке помечено двойной незакрашенной рамкой) существует несколько переходов в другие состояния по одному и тому же символу (l), причем эти переходы нельзя соединить в один, так как им присвоены разные значения n, то есть, при разборе нужно будет сдвинуться одновременно на один символ вправо и влево по разбираемой строке, что создает недетерминированность.

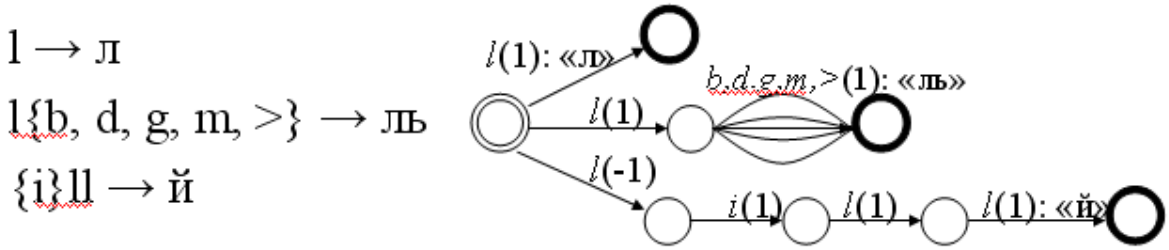

Рис 5. Недетерминированный автомат, построенный на основе системы правил {l → л, l{b, d, g, m, >} → ль, {i}ll → й}

Для преодоления этой проблемы требуется провести некоторые преобразования самой системы правил еще до построения конечного автомата. Этот комплекс преобразований представляет собой приведение правил к единообразному виду и потому назван *унификацией* системы правил.

8.4. Унификация системы правил

Унификация системы правил включает в себя четыре процедуры: добавление контекстов по умолчанию, выравнивание контекстов, добавление правил по умолчанию, удаление неоднозначных контекстов.

Добавление контекстов по умолчанию

Эта процедура служит для предотвращения ситуации, описанной в примере 6.

Если для одной и той же буквы есть правило с контекстом и правило без контекста, правилу без контекста добавляется контекст по умолчанию (пример 7).

Пример 7. Пример добавления контекстов

В результате процедуры добавления контекстов по умолчанию правила системы **R** = {l → л, {i}ll → й, l{b, d, g, m, >} → ль} преобразуются следующим образом:

l → л » {*}l{*} → л

{i}ll → й » {i}ll{*} → й

l{b, d, g, m, >} → ль » {*} l{b, d, g, m, >} → ль

где * - контекст по умолчанию, то есть, контекст, состоящий из всех символов входного алфавита. Для краткости записывается символом «*».

Что касается символа *, необходимы некоторые разъяснения его функции. Выше было сказано, что он обозначает любой символ входного алфавита. Это действительно так, но с некоторыми оговорками. В вышеприведенном примере 7 правила «l → л» и «l{b, d, g, m, >} → ль» формально вступали в конфликт, так как второе предписывает перевести символ «l» как «ль» в случае выполнения контекста, тогда как первое предписывает перевести тот же символ как «л» независимо от его контекста. Однако неявно правила с контекстом имеют приоритет перед правилами без контекста, так как вторые описывают общий случай, а первые – особые условия. После добавления контекста ко всем правилам, они все становятся равны, и теперь уже налицо явный конфликт не только между первым и третьим, но и между первым и вторым правилами системы. Поэтому символ «*» следует понимать не как все символы алфавита, а все символы алфавита, кроме уже перечисленных. То есть, символ «*» в правом контексте первого правила будет значить все символы, кроме символов «b», «d», «g», «m», «>» (из третьего правила) и «l» (из второго правила), левые контексты первого и третьего правил обозначают все символы кроме «i». Такое определение не очень удобно при работе с правилами, но после построения конечного автомата на основе этих правил становится понятен смысл определения. Если из некоторого состояния существуют переходы по символам *u*, *v*, и *, то по любому поступившему на вход символу *l* следует перейти по переходу, помеченному символом *, если только символ *l* не равен символу *u* или *v*.

Выравнивание контекстов

Эту процедуру можно назвать расширением процедуры добавления контекстов по умолчанию. Как уже было отмечено, к контекстам предъявляется требование одинаковой длины всех входящих в них цепочек. Если в контексте правила есть цепочки разной длины, выбирается цепочка наибольшей длины, остальные цепочки дополняются до нужной длины символом по умолчанию («*»). Причем цепочки левых контекстов дополняются слева, а цепочки правых контекстов – справа, так как значащая часть контекста должна непосредственно соседствовать с цепочкой α правила. То есть, контекст {e, ai, o}, если это левый контекст, будет преобразован в контекст {*e, ai, *o}, или в контекст {e*, ai, o*}, если это правый контекст. Напомним, что * - символ, заменяющий все символы алфавита, то есть, на самом деле запись цепочки «*e» обозначает множество цепочек {ae, be, ce, …, ze}.

Добавление правил по умолчанию

Данный этап унификации правил направлен на то, чтобы сделать систему правил универсальной: то есть, способной преобразовать любую цепочку символов входного алфавита. Поскольку система правил служит не для распознавания корректных строк данного языка, а для их преобразования, нет необходимости в строгом ограничении количества и качества правил.

Суть преобразования состоит в том, чтобы преобразовать любой символ входного алфавита независимо от того, в каком он находится контексте. Если для символа $l \in \mathbf{V_I}$, не

существует правила $\alpha \rightarrow \beta$ (без контекста), где $\alpha = l$, составляется правило $\alpha \rightarrow \beta$, где $\alpha = l$, а β – правая часть самого распространенного (имеющего больше всего употреблений в данной обучающей выборке) правила с контекстом для α (см. пример 8).

Если же для символа l в системе правил не существует ни одного правила $\mathbf{p_r}\ \alpha\ \mathbf{p_l} \rightarrow \beta$ ($\mathbf{p_r}$ и $\mathbf{p_l}$ могут быть пустыми), такого, что $\alpha = l$, никаких усовершенствований системы на этапе унификации не может быть произведено. В этом случае принято считать, что символ l в данном языке может употребляться лишь в строго определенных контекстах (например, символ *q* во французском языке употребляется только в подстроке *qu*). Но поскольку такой строгий контекст – довольно редкий случай, он не может сильно ухудшить качество транскрипции. Если же в слове все-таки встретится символ в не предусмотренном для него контексте, он будет передан в результат без перевода, так как появление такого символа, скорее всего, обозначает ошибку во входных данных (например, попытку перевести слово не с того языка).

Пример 8. Добавление правила по умолчанию

В системе правил **R** присутствуют следующие правила для символа «с»:

с{e, i} → с – встретилось 100 раз

с{a, o, u} → к – встретилось 200 раз

На этапе добавления правил по умолчанию в систему **R** будет добавлено правило с→к

Удаление неоднозначных контекстов

В принципе неоднозначность системы правил транскрипции – нормальное явление, вытекающее из неоднозначности графической системы языка. Но некоторые примеры неоднозначности делают невозможным построение конечного автомата, поэтому при унификации правил мы считаем нужным избавиться от них.

Пусть существуют правила $\mathbf{r_1}$ и $\mathbf{r_2}$, такие что $|\mathbf{r_1}(\alpha)| < |\mathbf{r_2}(\alpha)|$ и $\mathbf{r_1}(\alpha) + \gamma = \mathbf{r_2}(\alpha)$ или $\mathbf{r_1}(\alpha) + \gamma = \mathbf{r_2}(\alpha) + \delta$, где $\gamma \in \mathbf{r_1}(\mathbf{p_r})$, $\delta \in \mathbf{r_2}(\mathbf{p_r})$. Иными словами, конкатенация левой часть какого-либо правила с его контекстом совпадает с левой частью другого правила (или с конкатенацией левой части другого правила с контекстом, при условии, что левые части правил не равны). В этом случае предпочтение отдается правилу с более длинной левой частью ($\mathbf{r_2}$), из правого контекста более короткого правила ($\mathbf{r_1}$) удаляется контекст, создающий неоднозначность (γ).

Покажем, что эта процедура не отражается на результате транскрипции. Согласно процедуре преобразования строк с помощью системы правил, из применимых правил выбирается правило с самой длинной левой частью α. Таким образом, если существуют правила $\mathbf{r_1}$ и $\mathbf{r_2}$, такие, что $\mathbf{r_2}(\alpha) = \mathbf{r_1}(\alpha) + \mathbf{r_1}(\gamma)$, $\gamma \in \mathbf{r_1}(\mathbf{p_r})$, они оба будут применимы к строке, содержащей подстроку $\mathbf{r_2}(\alpha)$. Но в такой ситуации правило $\mathbf{r_1}$ никогда не будет применено. Следовательно, удаление неоднозначных правых контекстов $\mathbf{r_1}$ не изменяет результаты преобразования.

Здесь необходимо заметить, что в связи с выбором самой длинной цепочки сам автомат ограничивает свою выдачу. То есть при наличии правил $\mathbf{r_1}$ и $\mathbf{r_2}$, таких, что $\mathbf{r_2}(\alpha) = \mathbf{r_1}(\alpha) + \mathbf{r_1}(\gamma)$, $\gamma \in \mathbf{r_1}(\mathbf{p_r})$, выдаваться будет одна цепочка, тогда как правильным было бы выдать обе. Но в большинстве исследованных нами случаев подобное поведение является правильным.

8.5. Эквивалентность НКА и ДКА

ДКА, полученный из НКА, должен быть эквивалентен ему, то есть, распознавать ту же грамматику. Однако в настоящей работе используется расширенный конечный автомат, позволяющий переходы по разбираемой строке как вправо, так и влево. Для доказательства утверждения, что преобразование НКА в ДКА сохраняет множество разбираемых цепочек, будет сформулировано

Утверждение 2. Эквивалентность расширенных НКА и ДКА.

Процедура приведения расширенного конечного автомата к детерминированному виду сохраняет множество разбираемых цепочек.

Доказательство.

Используемый в работе автомат отличается от классического возможностью перехода по текущей строке не только вправо, но и влево на произвольное число символов. Покажем, что для этого автомата с учетом накладываемых на правила грамматики ограничений в результате процедуры преобразования НКА в ДКА получится эквивалентный автомат.

По определению автомата переходы влево по строке возможны только перед началом проверки левого контекста правила или после проверки правого контекста.

Переход назад для проверки левого контекста производится после просмотра первого символа преобразуемой подстроки (строки α в правиле). Пусть *l* – первый символ строки α. Будет создано новое состояние q', объединяющее все переходы из начального состояния по символу *l*. Пусть $\mathbf{R}_l$ – подмножество системы правил **R**, включающее все правила, левая часть которых начинается с символа *l*. Таким образом, состояние q' объединит цепочки всех правил из $\mathbf{R}_l$. Если у правил из $\mathbf{R}_l$ контексты разной длины, при создании q' может возникнуть неопределенность, связанная с тем, что разным переходам из начального состояния по *l* приписаны разные значения функции θ. Напомним, что функция θ переходов КА по текущему состоянию и текущему входному символу возвращает кортеж <q, *n*, **A**>, где q – новое состояние, *n* – сдвиг по входной строке, выраженный целым числом, **A** – множество выходных строк. Очевидно, что q для разных переходов по *l* будет различным. Множество **A** для перехода в q' будет составлять объединение всех **A** для исходных переходов. Проблему представляет *n*, так как детерминированный автомат должен однозначно определять процедуру разбора – то есть, при переходе не должно происходить сдвига сразу на 1 и на 2 символа влево.

Однако в соответствии с процедурой выравнивания контекстов строки, входящие в левый контекст каждого правила класса эквивалентности $\mathbf{R}_l$ должны быть одинаковой длины. Значит, при формировании перехода в q' его выходной кортеж <q, *n*, **A**> будет однозначно определен, так как во всех исходных кортежах, входящих в $\mathbf{R}_l$, значения *n* совпадают.

Рассмотрим переход назад по строке после проверки правого контекста. Необходимость объединить несколько переходов с отрицательным значением *n* может возникнуть в нескольких случаях:

- Если существует два правила $\mathbf{r_1}$ и $\mathbf{r_2}$ таких, что $\mathbf{r_1}(\alpha) = \mathbf{r_2}(\alpha)$;
- Если существует два $\mathbf{r_1}$ и $\mathbf{r_2}$, таких, что $\mathbf{r_2}(\alpha) = \mathbf{r_1}(\alpha) + \mathbf{r_1}(\gamma)$, $\gamma \in \mathbf{r_1}(\mathbf{p_r})$.

В первом случае, если у правил пересекающиеся правые контексты разной длины (например, у одного правила контекст *e*, у другого – *e* в конце слова, то есть *e*>), после проверки более короткого контекста происходит переход в конечное состояние со сдвигом влево, тогда как проверка более длинного контекста еще не закончена. Однако процедура выравнивания контекстов предусматривает и такие ситуации, поэтому у правил с совпадающей левой частью должна совпадать длина не только левых, но и правых контекстов. Таким образом, возврат назад будет осуществляться одновременно и на одинаковое расстояние.

Во втором случае мы считаем, что основная часть правила (строка α) имеет приоритет над контекстом, поэтому в процедуру унификации правил входит, помимо прочего, удаление контекстов, которые в объединении с левой строкой правила образуют строку, совпадающую со строкой другого правила. То есть, не возникает необходимости объединять переходы со значениями <q, *i*, **A**>, <q, *-j*, **A**>, где $i, j \in \mathbf{N}$.

Таким образом, при объединении переходов из одного состояния по одному и тому же символу число и направление шагов по разбираемой строке для этих переходов всегда будет совпадать. □

9. Эквивалентность автомата системе правил

Конечный автомат **g** может быть назван эквивалентным системе правил **R**, если для любой строки $\gamma \in \mathbf{V_I}^+$ выполняется равенство TrR(**R**, γ) = TrA(**g**, γ).

Очевидно, что автомат, не эквивалентный системе правил, не следует использовать для решения задачи, так как требуется преобразовывать строки с помощью системы правил, а конечный автомат был выбран только из-за высокой скорости обработки строк. Поэтому к автомату предъявляется требование эквивалентности системе правил, для доказательства которого будет сформулировано

Утверждение 3. Эквивалентность системы правил конечному автомату.

Процедуры преобразования строки с помощью системы правил и с помощью конечного автомата, построенного на основе этой системы правил по описанному алгоритму, эквивалентны, то есть, для любой строки над алфавитом $\mathbf{V_I}$ результатом их работы будут одинаковые строки или множества строк над алфавитом $\mathbf{V_O}$.

Доказательство.

При преобразовании строки γ с помощью системы правил **R** строка делится на подстроки $\gamma = <\gamma_1, \gamma_2, \ldots, \gamma_n>$, где $\gamma_i = \mathbf{r_i}(\alpha)$. Заметим, что в КА по процедуре построения переход из начального состояния в конечное возможен только по цепочке, сгенерированной на основе какого-либо правила. Это можно доказать следующим образом. При построении недетерминированного КА генерируются пути, соединяющие начальное состояние с конечным по цепочке, включающей в себя символы из $\mathbf{p_l}$, α, $\mathbf{p_r}$, то есть, прохождение по недетерминированному КА возможно только при соблюдении указанного условия.

Теперь покажем, что выбор передаваемой цепочки в КА осуществляется так же, как при разборе с использованием системы правил. Пусть существует два правила $\mathbf{r_1}$ и $\mathbf{r_2}$, таких, что $\mathbf{r_1}(\alpha) = l_0 \ldots l_k$, $\mathbf{r_2}(\alpha) = l_0 \ldots l_k l_n$, то есть, левая часть второго правила является левой частью первого правила с добавлением одного символа. Пути в графе конечного автомата, построенные для этих двух правил, будут совпадать до последнего перехода. Пусть после просмотра подстроки $l_0 \ldots l_k$ текущим состоянием является состояние A_k. Из него по определению построения КА существует два перехода: переход по символу l_n, ведущий в конечное состояние, которому приписана выходная строка $\mathbf{r_2}(\alpha)$, и переход по символу «*», обозначающему все символы кроме l_n. Таким образом, как и в случае разбора по правилам, к подстроке $l_0 \ldots l_k l_n$ будет применено $\mathbf{r_2}$, как обладающее самой длинной строкой α.

Следовательно, входная строка будет разбита на одинаковые подстроки как в случае применения правил, так и в случае КА. Подстроки передаются одними и теми же правилами. Следовательно, и выходные цепочки будут генерироваться одни и те же в обоих случаях, а их конкатенация будет давать один и тот же результат. То есть транскрипция с использованием системы правил и КА будут эквивалентны. □

10. Скорость работы конечного автомата

Известно, что автомат обрабатывает строки с линейной скоростью [5]. Это верно для использующихся в задачах автоматической обработки текста классических конечных автоматах. Однако описанный выше автомат, является расширенным, так как имеет дополнительную функциональность: может сдвигаться по обрабатываемой строке на произвольное число символов, причем как вправо, так и влево. Эта особенность делает неочевидным свойство линейности скорости работы КА. В связи с этим будет доказано

Утверждение 4. Линейность скорости работы конечного автомата.

Транскрипция строки длины n с помощью расширенного конечного автомата осуществляется за время, линейное относительно n.

Доказательство.

Известно, что разбор строки конечным автоматом имеет сложность O(n), где n – длина строки. При разборе строки с помощью классического конечного автомата строка

просматривается один раз и при каждом переходе происходит сдвиг на один символ, таким образом, разбор состоит из n переходов.

В расширенном КА, осуществляющем транскрипцию, возможен переход по строке на произвольное количество символов как вправо, так и влево.

Как это было отмечено выше, переход по строке влево проводится только при проверке контекстов. Перед проверкой левого контекста КА сдвигается назад по строке на длину контекста, так как до этого текущим символом является первый символ применяемого правила. После проверки правого контекста автомат также сдвигает назад текущую позицию на длину контекста, так как контекст не входит в правило и после применения текущего правила должен быть разобран отдельно. Таким образом, величина сдвига влево определяется длиной контекста. Если ограничений на длину контекста нет, то сложность разбора строки в худшем случае составит $O(n^2)$, то есть, при проверке каждого символа нужно будет проверить его контекст, включающий все остальные символы строки.

Но система правил, на основе которой строится конечный автомат, в соответствии с методом порождения правил не может включать в себя правила с контекстом длиннее двух символов. Это значит, что при разборе строки для преобразования каждого из ее символов в худшем случае придется проверить сам символ, два предшествующих и два следующих за ним. То есть сложность разбора составит O(5n). Такая оценка сложности для строк длиннее 5 символов будет меньше квадратичной. Что же касается строк длины 5 символов и меньше, сложность их разбора все равно не достигнет O(5n), так как для крайних символов строки нет возможности проверить контексты длины 2.

11. Эксперименты по транскрипции с использованием конечного автомата

Описанный метод транскрипции был реализован программно. Были проведены вычислительные эксперименты, направленные на проверку гипотезы о том, что описанный в статье конечный автомат обладает более высокой скоростью обработки строк по сравнению с методами, использующими перебор правил.

Метод транскрипции, реализованный в системе «Трансскриба», обладает сложностью $O(M \cdot N)$, где M – количество правил транскрипции, N – длина строки [4]. Метод транскрипции с помощью конечного автомата, как было показано выше, обладает линейной сложностью относительно длины разбираемой строки – O(N).

Была проведена транскрипция имен из тестовых корпусов для различных языков (см. таблицу 1, языки оригинала перечислены в таблице, язык перевода для всех корпусов – русский). Имена из каждого корпуса были переведены с помощью обоих методов транскрипции. При экспериментах с системой «Трансскриба» использовались правила, написанные для этой системы вручную, правила для КА были сгенерированы с помощью системы автоматического порождения правил транскрипции, описанного в начале работы. Разный объем правил не позволяет точно оценить выигрыш по скорости, но все же дает представление о порядке.

Язык	Количество слов	Трансскриба		Конечный автомат	
		Количество правил	Время (мс)	Количество правил	Время (мс)
Шведский	1884	423	780	115	63
Французский	1882	356	970	160	50
Испанский	1066	107	320	87	15
Монгольский	232	46	90	41	<1
Китайский (система Уэйда)	2680	337	940	59	60

Таблица 5. Результаты вычислительных экспериментов

Если бы транскрипция с помощью разных систем правил проводилась бы с помощью перебора правил («Трансскриба»), выигрыш в скорости был бы примерно равен разнице в объеме систем правил. Однако эти цифры оказываются выше:

- шведский язык: быстрее в 12 раз (в 3,6 раз меньше правил);
- французский язык: быстрее в 19 раз (в 2,2 раза меньше правил);
- китайский язык: быстрее в 16 раз (в 5,6 раз меньше правил).

Показательны монгольский и испанский – в них при сопоставимом количестве правил скорость перевода увеличилась в 90 и 21 раз соответственно. Такое сильное увеличение скорости для монгольского можно объяснить тем, что в правилах для этого языка почти не используются контексты, то есть, на каждый рассматриваемый символ приходился один переход, а не 3-5, как в других языках.

Заметим также, что время транскрипции китайских имен и шведских имен с помощью конечного автомата сопоставимо, хотя для шведского языка правил почти в два раза больше. В шведском корпусе в 1,4 раз меньше имен, но китайские имена короче, поэтому объем обработанных строк можно считать равным. Это указывает на независимость времени обработки строк от объема используемой системы правил.

12. Список литературы

1. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848.
2. [Кодзасов 2001] Кодзасов С. В., Кривнова О. Ф., Общая фонетика, учебник. – М.:РГГУ, 2001.
3. K. Knight and J. Graehl. 1998. Machine transliteration. Computational Linguistics, 24(4):599–612.
4. Бондаренко В.А., Ёлкин С.В., Клышинский Э.С., Слёзкина О.Ю. Практическая транскрипция фамильно-именных групп для машиночитаемых документов // Сб. трудов Международного семинара «Диалог-2002». М., 2002.
5. Ахо А., Ульман Д., Теория синтаксического анализа, перевода и компиляции, М.: «Мир», 1978.