Самообучающаяся система машинной транскрипции с использованием нестохастического конечного автомата♣

© В.К. Логачева, Э.С. Клышинский

Российский государственный гуманитарный университет, ИПМ им. М.В. Келдыша РАН, г. Москва
logacheva_vk@mail.ru, klyshinsky@mail.ru

Аннотация

Описан метод транскрипции имен собственных, использующий конечный автомат и правила, записанные в виде продукций. Представлен метод автоматического порождения правил. Сгенерированные правила могут быть преобразованы в конечный автомат, с помощью которого осуществляется транслитерация.

1 Введение

Для электронных архивов большого объема очень важна автоматическая подготовка документов. При поступлении в библиотеку иноязычного документа для помещения его в каталог может понадобиться перевод его названия, аннотации, выходных данных. Системы машинного перевода зачастую оставляют имена собственные (в частности, фамильно-именные группы) без перевода, тогда как фамилия автора – самая важная информация о документе. Перевод имен вручную затруднителен не только потому, что на него тратится много времени, но и из-за различающихся в разных языках правил чтения, которые могут быть неизвестны переводчику, не владеющему данным языком.

Существует несколько подходов к передаче имени собственного средствами другого языка:

- перевод (например, Easter Island – остров Пасхи). Этот способ очень редко можно применить для имен собственных, так как они зачастую не имеют лексического значения;
- транслитерация:
  - строгая – сопоставление каждой букве исходного языка буквы языка перевода; этот способ может искажать звучание слова, так как почти во всех языках существуют диграфы – устойчивые сочетания букв, которые читаются особым образом; даже правил расширенной транслитерации, т. е. правил, допускающих сопоставление одной букве нескольких (sh → ш), не всегда хватает, чтобы описать все зависимости фонетики и графики, существующие в языке;
  - транслитерация с учетом фонетического облика слова, также называемая транскрипцией; в отличие от транскрипции в классическом лингвистическом понимании слова исходного языка записываются не особым фонетическим алфавитом, а символами целевого языка.

В разное время наибольшей популярностью пользовались разные подходы, но с середины XX века господствующей тенденцией стала передача звучания имени при его переводе на иностранный язык. С развитием компьютерной лингвистики и средств автоматизации встал вопрос о создании систем автоматической транслитерации и транскрипции имен собственных. К ранним работам в этой области относятся работы группы Найта. Первоначально был разработан метод, позволяющий проводить транслитерацию с японского языка на английский [1]. Вскоре этот метод был адаптирован для работы с арабским языком [2]. Основной задачей метода являлось обнаружение и восстановление имен собственных, уже переданных с использованием другого алфавита. В качестве основы использовался модифицированный алгоритм Витерби. Однако базовый принцип использования лишь отдельных символов алфавита не позволял получить высокие показатель качества передачи. В связи с этим стали применяться методы, использующие подстроки [3]. Применение подстрок взамен отдельных символов позволило повысить качество транслитерации примерно с 30 до 90%.

Еще одной важной задачей здесь является автоматическое обучение системы переходов конечного автомата, используемого в перечисленных методах. На данный момент развиваются методы как обучения с учителем [1], так и без него [4]. Последние работы позволили перейти к обучению по одноязычному корпусу [5], что привело, правда, к существенному падению качества передачи.

Данная работа основана на программе «Трансскриба» [6]. В отличие от приведенных работ, здесь используется не стохастический, а детерминированный подход. Лингвистам предлагается написать правила, которые поступают на вход системы. Метод позволяет получить высокую эффективность передачи, однако требует затрат на создание систе-

Труды 12й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» – RCDL'2010, Казань, Россия, 2010

мы правил. Следует заметить, что правила для одного языка создаются за срок от нескольких недель до нескольких месяцев, в зависимости от сложности задачи.

2 Метод транскрипции

Предлагаемый в данной работе метод объединяет удобство и скорость разбора входной цепочки с использованием конечных автоматов с удобством представления правил преобразования в привычном для человека виде. Изначально правила преобразования цепочек из одного языка на другой представляются в виде «входная цепочка» → «выходная цепочка». Скорость применения подобных правил будет зависеть от размеров базы и длины входной цепочки. В связи этим в данной статье для преобразования цепочек используется конечный автомат. Скорость разбора с использованием конечного автомата пропорциональна длине цепочки. Здесь мы считаем, что набор правил может быть как создан лингвистом, так и получен в ходе обучения по алгоритму, описанному ниже. Таким образом, при необходимости специалист работает в привычных ему терминах правил. Далее набор правил может быть расширен за счет дообучения. В другом случае набор правил может быть изначально автоматически сформирован с использованием предлагаемого метода. Набор правил преобразуется в конечный автомат, который используется для передачи имен собственных с языка на язык.

2.1 Структура конечного автомата

Используемый нами автомат представляет собой преобразователь. Его можно представить как множество $\mathbf{g} = <\mathbf{V_I}, \mathbf{V_O}, \mathbf{Q}, \mathbf{q_0}, \mathbf{F}, \boldsymbol{\delta}>$, где

$\mathbf{V_I}$ – входной алфавит (алфавит языка оригинала);

$\mathbf{V_O}$ – выходной алфавит (алфавит языка перевода);

$\mathbf{Q}$ – множество состояний автомата;

$\mathbf{q_0}$ – начальное состояние автомата;

$\mathbf{F}$ – множество конечных состояний;

$\boldsymbol{\delta}$ – функция переходов $\mathbf{Q}\times\mathbf{V_I} \rightarrow <\mathbf{Q}, a>$.

Здесь a – это действие, совершаемое при переходе, причем $a\in\mathbf{A}$ – множеству действий, выполняемых конечным автоматом при переходе; $\mathbf{A}$ = {out(), shift()}; Out(*s*) – функция, подающая на выход строку символов s (в том числе пустую) из алфавита перевода; Shift(*n*) – функция, выполняющая сдвиг рассматриваемой строки на n символов.

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

2.2 Описание правил транскрипции

За основу был взят метод, описанный в [6]. Но, поскольку в предлагаемом методе правила используются, прежде всего, для пополнения матрицы переходов конечного автомата и генерируются компьютером, а не человеком, их структура отличается от базового варианта. В нашем случае правило представляется в виде пары $r = <\mathbf{p}, c>$, где

p – непустое упорядоченное множество символов из алфавита языка оригинала;

c – цепочка символов из алфавита языка перевода, которой передается входная цепочка;

$\mathbf{p} = <\mathbf{p1}, \mathbf{p2}, \mathbf{p3}>$, где **p1** и **p3** – пред- и постусловия соответственно (левый и правый контекст правила), а **p2** – преобразуемая подстрока, $\mathbf{p2} = <v_1, v_2, \ldots, v_{n2}>$, где $v_i\in\mathbf{V_I}$, $i\in[1, n2]$, $\mathbf{p1} = <\{v_1\}, \{v_2\}, \ldots, \{v_{n1}\}>$, где $v_i\in\mathbf{V_I}$, $i\in[1, n1]$, $\mathbf{p3} = <\{v_1\}, \{v_2\}, \ldots, \{v_{n3}\}>$, где $v_i\in\mathbf{V_I}$, $i\in[1, n3]$, nj – количество символов j-й подстроки;

$c = <w_1, w_2, \ldots, w_{nc}>$, где $w_i\in\mathbf{V_O}$, $i\in[1,nc]$, nc – длина выходной строки.

Исходя из этого, правило применимо с текущей позиции, если на текущей позиции находится подстрока **p2**, перед ней – подстрока, на соответствующих позициях которой находятся символы из **p1**, а после нее – подстрока, перед ней – подстрока, на соответствующих позициях которой находятся символы из **p3**. Считается, что к входной строке последовательно ищутся применяемые правила. При нахождении такого правила текущая позиция сдвигается вправо на n2, а на выход подается *c*.

2.3 Преобразование правил в конечный автомат

Для правила $\mathbf{r} = <\mathbf{p}, c>$ процедура добавления в конечный автомат будет следующей. Здесь, за исключением начального состояния q_0, все состояния заводятся для данного правила заново. При этом дополнительным алгоритмом обеспечивается уникальность нумерации состояний.

Для правил, в которых левый контекст пуст, в КА добавляется переход из состояния q_{i-1} в состояние q_i по символу v_i из **p2**, $i\in[1,n2]$. При этом никакие действия не совершаются. Состояние q_{n2} помечается как конечное.

Для правил, в которых правый контекст пуст, в КА добавляется переход из q_{n2} в q_0 по пустому символу, сопровождаемый Shift(n2) и Out(*c*).

Для правил, в которых правый контекст не пуст, в КА добавляется переход из состояния q_{n2+i-1} в состояние q_{n2+i} по всем символам из $\{v_i\}$ из **p3**, $i\in[1,n3]$. Далее в КА добавляется переход из q_{n2+n3} в q_0 по пустому символу, сопровождаемый действиями Shift(n2) и Out(*c*).

Для правил, в которых левый контекст не пуст, для всех $q_f\in\mathbf{F}$ в КА добавляются переход из q_f в $q_{n2+n3+1}$ по всем символам $\{v_1\}$ из **p1** и переходы из состояния $q_{n2+n3+i-1}$ в состояние $q_{n2+n3+i}$ по всем символам из $\{v_i\}$ из **p1**, $i\in(1,n1]$. При этом никакие действия не совершаются. Далее в КА добавляется переход из $q_{n1+n2+n3}$ в q_0 по пустому символу, сопровождаемый действиями Shift(n2) и Out(*c*).

Подобные операции проводятся для всех правил.

3 Порождение правил

Для облегчения поиска соответствий в строках на языке оригинала и перевода каждое слово и его перевод делятся на слоги. Поскольку мы не имеем никакой информации о структуре языка и правилах

слогоделения в этом языке, деление производится по формальным признакам. В порождении правил участвуют только слова, для которых установлено взаимно однозначное соответствие слогов в оригинале и переводе.

3.1 Разделение слова на слоги

Термин «слог» в данном контексте употребляется в смысле, отличном от того, в котором он используется в трудах по фонетике. Поэтому необходимо дать его определение.

Слог – это непустое множество букв, содержащее один или более символов, обозначающих гласные звуки. Слово делится на слоги по следующим правилам:

- деление осуществляется по гласной букве; граница слога – после гласной;
- 2 и более гласных подряд не разделяются;
- 2 и более согласных не разделяются;
- множество элементов, среди которых нет гласной, не выделяется в отдельный слог;
- символы начала и конца слова считаются согласными буквами.

Дополнительное правило для русского языка:

- не отделять «ь» от предыдущей согласной.

Таким образом, слог – это цепочка вида C^*V^+, где C – буква, обозначающая согласный звук, V – буква, обозначающая гласный звук. Слог может иметь вид $C^*V^+C^+$ только в том случае, если это последний слог в слове, и заключительное множество согласных не может быть выделено в отдельный слог, так как среди них нет слогообразующего символа.

Формально почти все слоги, полученные по изложенным выше правилам, являются открытыми (то есть заканчиваются на гласную). Однако у слога присутствует параметр открытости/закрытости, определяющийся следующим образом: при делении для каждого слога проверяется количество начальных согласных в следующем слоге: если их больше одной, слог считается закрытым. Введение этого параметра связано с тем, что во многих языках (например, в английском и других языках германской группы) он может влиять на правила чтения гласных букв.

3.2 Порождение первичных правил

Первый этап порождения правил – порождение первичных правил. Первичными мы называем правила транслитерации, то есть правила, для которых $|\mathbf{p2}| = |c| = 1$.

Первичные правила порождаются на основе слогов вида CV → CV: i-му символу слога ставится в соответствие i-й символ его перевода при условии, что оба символа обозначают звуки одного и того же типа (гласные или согласные). При таком подходе вероятность порождения некорректного правила очень мала.

В порожденном множестве первичных правил могут содержаться неоднозначности, то есть пары правил $\mathbf{r1} = \langle \mathbf{p}, c \rangle$ и $\mathbf{r2} = \langle \mathbf{p}, c \rangle$, такие, что $\mathbf{r1}(\mathbf{p2}) = \mathbf{r2}(\mathbf{p2})$ и $\mathbf{r1}(c) \neq \mathbf{r2}(c)$. Такие ситуации в принципе возможны в языке, но на данном этапе неоднозначность скорее всего обозначает влияние на букву окружающих ее букв. Для избавления от неоднозначностей вводится новый тип правил – временные правила.

Временными правилами мы называем правила, используемые при обучении системы правил. Временное правило представляет собой тройку $r = \langle \mathbf{p}, c, \mathbf{s} \rangle$, где

p, *c* – те же, что в определенном выше формате правила (см. п. 2.2);

s – множество слогов, удовлетворяющих правилу.

При генерации правила в его множество s добавляется слог, на основе которого оно было сгенерировано. При обучении системы правил слог, к которому удалось применить правило, добавляется в множество **s**. Хранение всех слогов позволяет при обнаружении неоднозначностей в правилах составлять более полные контексты, а также выявлять частотность употребления правила (например, если правилу удовлетворяет только один слог, можно с большой вероятностью утверждать, что это исключение).

Избавление от неоднозначностей производится следующим образом.

Для каждого правила r_i, для которого существует $\mathbf{r_j}$, такое, что $\mathbf{r_i}(\mathbf{p2}) = \mathbf{r_j}(\mathbf{p2})$, $\mathbf{r_i}(c) \neq \mathbf{r_j}(c)$, для каждого s_k из $\mathbf{r_i}(\mathbf{s})$ составляется новое правило $\mathbf{r_{ik}} = \langle \mathbf{p}, c, \mathbf{s} \rangle$, такое, что $\mathbf{r_{ik}}(\mathbf{p2}) = \mathbf{r_i}(\mathbf{p2})$, $\mathbf{r_{ik}}(c) = \mathbf{r_i}(c)$, $\mathbf{r_{ik}}(\mathbf{p1}) = \langle \{v_1\} \rangle$, где $v_1 \in V_I$ – символ, предшествующий **p2** в слоге s_k, $\mathbf{r_{ik}}(\mathbf{p3}) = \langle \{v_2\} \rangle$, $v_2 \in \mathbf{V}_I$ – символ, следующий за **p2** в слоге s_k. В случае, если **p2** – начальная или заключительная подстрока в слоге, v_1 и v_2 берутся из предыдущего или следующего слога соответственно.

После порождения системы первичных правил производится расширение их контекстов на основе слогов вида $CnV \rightarrow CnV$, где $n > 1$. Каждый слог s_i, который не может быть целиком разобран с помощью существующей системы правил, можно представить как $\langle p_{i1}, \ldots, p_{ik}, \mathbf{p_x}, p_{ik+1}, \ldots, p_{in} \rangle \rightarrow \langle c_{i1}, \ldots, c_{ik}, \mathbf{c_x}, c_{ik+1}, \ldots, c_{im} \rangle$, где $\mathbf{p_x} \rightarrow \mathbf{c_x}$ – подстрока, не удовлетворяющая ни одному из существующих правил. Можно выделить три случая несоответствия px правилам:

- существует $\mathbf{r_i}$, такое, что $\mathbf{p_x} = \mathbf{r_i}(\mathbf{p2})$, $\mathbf{c_x} = r_i(c)$, но $\mathbf{p_{ik}} \not\subset \{v_{11}\}$ или $p_{ik+1} \not\subset \{v_{31}\}$ (т. к. $\mathbf{r_i}(\mathbf{p1}) = \langle \{v_{11}\} \rangle$, $\mathbf{r_i}(\mathbf{p3}) = \langle \{v_{31}\} \rangle$); в этом случае контекст правила $\mathbf{r_i}$ расширяется: $\mathbf{r_i}(\mathbf{p1}) = \langle \{v_{11}, p_{ik}\} \rangle$, $\mathbf{r_i}(\mathbf{p3}) = \langle \{v_{31}, p_{ik+1}\} \rangle$;
- не существует $\mathbf{r_i}$, такого, что $\mathbf{p_x} = \mathbf{r_i}(\mathbf{p2})$ и $\mathbf{c_x} = \mathbf{r_i}(c)$; в этом случае составляется новое правило $\mathbf{r_j}$, такое, что $\mathbf{r_j}(\mathbf{p1}) = \langle \{p_{ik}\} \rangle$, $\mathbf{r_j}(\mathbf{p2}) = \mathbf{p_x}$, $\mathbf{r_j}(\mathbf{p3}) = \langle \{p_{ik+1}\} \rangle$, $\mathbf{r_j}(c) = \mathbf{c_x}$, $\mathbf{r_j}(\mathbf{s}) = \{s_i\}$;
- существует $\mathbf{r_i}$, такое, что $\mathbf{p_x} = \mathbf{r_i}(\mathbf{p2})$, $p_{ik} \subset \{v_{11}\}$ и $p_{ik+1} \subset \{v_{31}\}$ (т. к. $\mathbf{r_i}(\mathbf{p1}) = \langle \{v_{11}\} \rangle$, $\mathbf{r_i}(\mathbf{p3}) = \langle \{v_{31}\} \rangle$), $\mathbf{c_x} \neq \mathbf{r_i}(c)$; это может объясняться одной из следующих причин:

- контекст правила r_i недостаточен для правильной интерпретации подстроки, и надо рассматривать не один, а несколько символов, предшествующих ей или следующих за ней. Например, для имени Marin → Марен (французский язык) было порождено правило i → e, которое при избавлении от неоднозначностей приобрело вид i{n}→ e. Но имя Marine → Марин не удовлетворяет этому правилу, потому что i переходит в e(э), только если после него стоит n и слог является закрытым (заканчивается на согласную). В этом случае нужно проверять не один, а два следующих за i символа;
- подстрока может читаться двумя различными способами в силу причин, не зависящих от контекста. В шведском языке å читается как «о», однако в машиночитаемых текстах без диакритик å заменяется на aa. Между тем сочетание двух шведских букв a будет передаваться на русский язык как «аа». В тестовой выборке для шведского языка можно встретить примеры неоднозначностей Baad → Баад, Baang → Бонг, Haapanen → Хаапанен, Haafman → Хофман, которые нельзя предугадать, имея в качестве обучающей выборки машиночитаемый файл. Значит, для каждого шведского имени, содержащего подстроку «аа», будет сгенерировано два варианта перевода.

В настоящее время нет алгоритма выбора одной из двух описанных причин и избавления от подобных неоднозначностей.

3.3 Порождение сложных правил

Более сложные правила порождаются при анализе слогов более сложной структуры, то есть слогов вида $C^pV^n \rightarrow C^qV^m$, где $p \neq q$ и/или $n \neq m$. В этом случае, как и в вышеописанном алгоритме пополнения контекстов с помощью слогов вида $C^nV \rightarrow C^nV$, каждый слог можно представить как $<p_{i1}, \ldots, p_{ik}, \mathbf{p_x}, p_{ik+1}, \ldots, p_{in}> \rightarrow <c_{i1}, \ldots, c_{ik}, \mathbf{c_x}, c_{ik+1}, \ldots, c_{im}>$, где $\mathbf{p_x} \rightarrow \mathbf{c_x}$ – подстрока, не удовлетворяющая ни одному из существующих правил. Можно выделить три случая несоответствия px правилам:

- $p_x = \varnothing$, $c_x \neq \varnothing$ – эта ситуация обозначает появление в транскрипции букв, которых не было в оригинале (при составлении правил транскрипции с произвольного языка на русский чаще всего в такой позиции встречается буква «ь», которая обозначает палатализацию предыдущего согласного, которая никак не отражается на письме в языке оригинала). В этом случае составляется правило $\mathbf{r_i}$, такое, что $\mathbf{r_i(p1)} = p_{ik-1}$, $\mathbf{r_i(p2)} = p_{ik}$, $\mathbf{r_i(p3)} = p_{ik+1}$, $\mathbf{r_i}(c) = c_{ik}+\mathbf{c_x}$. – иными словами, правило для предшествующего $\mathbf{p_x}$ символа, который в определенном контексте передается несколькими символами;
- $\mathbf{p_x} \neq \varnothing$, $\mathbf{c_x} = \varnothing$ – эта ситуация обозначает наличие букв, которые не читаются, а значит, не записываются в транскрипции (например, буквы *t*, *d*, *s* и др. во французском языке на конце слова). Для таких случаев составляется правило ri, такое, что $\mathbf{r_i(p1)} = p_{ik}$, $\mathbf{r_i(p2)} = \mathbf{p_x}$, $\mathbf{r_i(p3)} = p_{ik+1}$, $\mathbf{r_i}(c) = \varnothing$, то есть определяется контекст, в котором подстрока $\mathbf{p_x}$ не читается;
- $\mathbf{p_x} \neq \varnothing$, $\mathbf{c_x} \neq \varnothing$ – в случае, когда $|\mathbf{c_x}| = 1$, такая ситуация с большой вероятностью говорит о том, что $\mathbf{p_x}$ – устойчивое сочетание букв, обозначающее один звук – ди- или триграф, если же $|\mathbf{p_x}| = 1$, это означает появление буквы, обозначающей звук, которого нет в русском языке и для записи которого используется 2 или более русских букв. В обоих случаях составляется новое правило r_i, такое, что $\mathbf{r_i(p1)} = \mathbf{r_i(p3)} = \varnothing$, $\mathbf{r_i(p2)} = \mathbf{p_x}$, $\mathbf{r_i}(c) = \mathbf{c_x}$.

Во избежание появления недетерминированного вывода в конечном автомате стоит проверять каждое вновь добавляемое правило на однозначность. Если для добавляемого правила r1 среди уже принятых правил существует такое r2, что r1(p) и r2(p) имеют общий префикс, формируется правый контекст для правила с более короткой входной строкой.

После каждого этапа (порождение новых правил, расширение контекста существующих правил) осуществляется нормализация системы правил.

3.4 Нормализация системы правил

Нормализация системы правил состоит в избавлении от правил, полностью или частично дублирующих друг друга.

Для двух правил $\mathbf{r_1} = <\mathbf{p}, c, \mathbf{s}>$ и $\mathbf{r_2} = <\mathbf{p}, c, \mathbf{s}>$, где $\mathbf{r_1(p1)} = \mathbf{r_2(p1)} = \mathbf{r_1(p3)} = \mathbf{r_2(p3)} = \varnothing$, справедливо утверждение $\mathbf{r_1} = \mathbf{r_2}$, если $\mathbf{r_1(p2)} = \mathbf{r_2(p2)}$ и $\mathbf{r_1}(c) = \mathbf{r_2}(c)$. В этом случае правила $\mathbf{r_1}$ и $\mathbf{r_2}$ заменяются правилом $\mathbf{r_3} = \mathbf{r_1} \cup \mathbf{r_2} = <\mathbf{p}, c, \mathbf{s}>$, где $\mathbf{r_3(p1)} = \mathbf{r_3(p3)} = \varnothing$, $\mathbf{r_3(p2)} = \mathbf{r_1(p2)} = \mathbf{r_2(p2)}$, $\mathbf{r_3}(c) = \mathbf{r_1}(c) = \mathbf{r_2}(c)$ и $\mathbf{r_3(s)} = \mathbf{r_1(s)} \cup \mathbf{r_2(s)}$.

Два правила $\mathbf{r_1} = <\mathbf{p}, c, \mathbf{s}>$ и $\mathbf{r_2} = <\mathbf{p}, c, \mathbf{s}>$, для которых хотя бы одно из множеств $\mathbf{r_1(p1)}$, $\mathbf{r_2(p1)}$, $\mathbf{r_1(p3)}$, $\mathbf{r_2(p3)}$ не пусто, считаются эквивалентными, если $\mathbf{r_1(p2)} = \mathbf{r_2(p2)}$, $\mathbf{r_1}(c) = \mathbf{r_2}(c)$, $|\mathbf{r_1(p1)}| = |\mathbf{r_2(p1)}|$, $|\mathbf{r_1(p3)}| = |\mathbf{r_2(p3)}|$, и выполняется хотя бы одно из неравенств $\mathbf{r_1(p1)} \cap \mathbf{r_2(p1)} \neq \varnothing$, $\mathbf{r_1(p3)} \cap \mathbf{r_2(p3)} \neq \varnothing$. В этом случае правила $\mathbf{r_1}$ и $\mathbf{r_2}$ заменяются правилом $\mathbf{r_3} = \mathbf{r_1} \cup \mathbf{r_2} = <\mathbf{p}, c, \mathbf{s}>$, где $\mathbf{r_3(p1)} = \mathbf{r_1(p1)} \cup \mathbf{r_2(p1)}$, $\mathbf{r_2(p3)} = \mathbf{r_1(p3)} \cup \mathbf{r_2(p3)}$, $\mathbf{r_3(p2)} = \mathbf{r_1(p2)} = \mathbf{r_2(p2)}$, $\mathbf{r_3}(c) = \mathbf{r_1}(c) = \mathbf{r_2}(c)$ и $\mathbf{r_3(s)} = \mathbf{r_1(s)} \cup \mathbf{r_2(s)}$.

В конце этапа нормализации проводится проверка на предмет порождения «фантомных» правил, для которых имеются противоречащие им.

4 Реализация метода. Результаты экспериментов

В настоящее время программно реализованы лишь некоторые этапы описанного метода: деление на слоги, отбор слов, подходящих для участия в порождении правил, составление первичной системы правил, избавление ее от неоднозначностей путем добавления контекстов, проверка слогов более сложной структуры с помощью полученной системы правил и пополнение контекстов на основе этих слогов. Каждый этап завершается нормализацией системы правил. Примеры ниже демонстрируют различные типы полученных правил с контекстами:

(1) {aeuy}c{i}→c

{<r}c{e}→c
{<eins}c{aklo}→к
(2) {bcdgjklmnprstvz}e{bcdgklmnrstvwz}→e
{<}e{dlmntx}→э

Из примера (1) видно, что буква *c* передается на русский язык как «с», если после нее стоят буквы *i* или *e*, и как «к» в остальных случаях. Левый контекст здесь не имеет значения, но на первых этапах анализа невозможно определить, какая из букв – следующая или предыдущая – влияет на данную. Понять это можно, только собрав достаточное количество примеров. Если контекст включает все символы алфавита, его можно не учитывать. Если контекст включает большую часть символов алфавита, не объединенных каким-либо общим свойством (например, принадлежность к согласным буквам), его также можно не учитывать. Такая ситуация показана в примере (2): буква *e* передается на русский язык как «е» во всех позициях, кроме начальной позиции в слове.

Контексты правил в приведенных примерах, особенно в примере (1), не полны, но это не является серьезной проблемой, так как они могут быть пополнены при анализе слогов (и успешно пополняются в текущей версии системы). Гораздо более серьезной проблемой являются пересекающиеся контексты в правилах с одинаковой входной строкой и разными выходными. Это может обозначать как недостаточность существующего контекста, так и неоднозначность в правилах чтения для данного языка. При анализе только фонетического уровня эта проблема не может быть решена.

Процедура нормализации контекста еще не формализована, поэтому она не включена в настоящую версию алгоритма.

5 Обсуждение метода

Метод в том виде, в котором он представлен в данной статье, имеет некоторые недостатки.

Неизвестно, универсален ли метод, так как еще не все этапы реализованы программно, а генерация правил по этому алгоритму вручную проводилась на примере французского и немецкого языков. Для некоторых языков, например, английского, метод может оказаться неприменимым. Впрочем, в случае английского языка даже составленные экспертом-лингвистом правила не всегда являются однозначными.

Надо отметить также низкую устойчивость метода к ошибкам в тестовых данных и исключениям из правил. Для каждого слога, не объясненного уже существующими правилами, порождается новое правило, все слоги имеют одинаковый вес. С другой стороны, это свойство позволяет генерировать адекватные системы правил с помощью сколь угодно малой тестовой выборки, при условии, что в ней отражены основные правила орфографии данного языка.

Механизм использования контекста в настоящее время также несовершенен. В правилах используются только «простые» контексты: составляются множества символов, которые могут стоять на n-й позиции слева и справа от рассматриваемой цепочки. Однако иногда необходимо учитывать более сложные условия: закрытый слог, последний (первый) слог в слове и др.

Данный метод, в отличие от многих предшествующих, использует информацию о буквах, но весь анализ на данном этапе происходит исключительно на фонетическом уровне – то есть в слове не выделяются морфемы и не учитывается принадлежность буквы к той или иной морфеме. Отчасти это объясняется тем, что систему планируется использовать для машинной транскрипции фамильно именных групп произвольных языков, и зачастую единственной доступной информацией об этих языках будет список тестовых примеров (то есть слов с переводом). Но в некоторых случаях принадлежностью буквы к префиксу или суффиксу определяется, как она будет читаться, и невозможно сформировать соответствующее правило, рассматривая слово исключительно на фонетическом уровне.

Литература

[1] Al-Onaizan Y., Knight K. Machine transliteration of names in arabic text// Proc. of the ACL Workshop on Computational Approaches to Semitic Languages, 2002.

[2] Klyshinsky E., Maximov V., Yolkeen S. Cross-language transcription of proper names// Language Forum. – 2008. – V. 34, No 1. – P. 137-152.

[3] Knight K., Graehl J. Machine transliteration// Computational Linguistics. – 1998. – V. 24, No 4. – P. 599-612.

[4] Ravi S., Knight K. Learning phoneme mappings for transliteration without parallel data// Proc. of Human Language Technology Conf. The 2009 Annual Conf. of the North American Chapter of the Association for Computational Linguistics, 2009.

[5] Sherif T., Kondrak G. Substring-based transliteration//Proc. of the ACL Workshop on Computational Approaches to Semitic Languages, 2007.

[6] Tao T., Yoon S., Fister A., Sproat R., Zhai C. Unsupervised named entity transliteration using temporal and phonetic correlation// Proc. of EMNLP, 2006.

Self-learning system of machine transliteration using non-stochastic finite state automaton

V. Logacheva, E. Klyshinsky

The article describes a method of proper names transliteration that uses a combination of finite automaton and translation rules in the form of production. We introduce a method of automatic rules generation. The method uses a set of proper names and their transliterations. Then rules can be translated into finite automaton which conducts the transliteration method.

♣ Работа выполнена при частичной финансовой поддержке РФФИ (проект 10-01-00800)