МЕТОД ПОРОЖДЕНИЯ ПРАВИЛ МЕЖЪЯЗЫКОВОЙ МАШИННОЙ ТРАСКРИПЦИИ

В. К. Логачева (logacheva_vk@mail.ru)

Е. С. Клышинский (klyshinsky@mail.ru)

Институт Прикладной Математики РАН, Москва, Россия

В статье рассматривается метод генерации правил машинной транскрипции. Метод пригоден для применения к языкам различных групп. Генерация правил проводится на основе анализа коллекции имен собственных, в которой представлено написание как на языке оригинала, так и на выходном языке. Работа поддержана грантом РФФИ № 10-01-00800.

Ключевые слова: машинная транскрипция, правила машинной траскрипции, генерация правил, имена собственные.

NON-STOCHASTIC LEARNING OF CROSS-LANGUAGE TRANSLITERATION RULES FROM A SMALL DATASET

V. K. Logacheva (logacheva_vk@mail.ru)

E. S. Klyshinskii (klyshinsky@mail.ru)

Keldysh IAM Russian Academy of Sciences, Moscow, Russia

We present a language-independent method of generating rules for machine transliteration. The generation of rules is based on the analysis of a test dataset, which contains names written in the source language and their transliterations into the target language.

Key words: transliteration rules, machine transliteration rules, rules generation, proper nouns.

1. Introduction

Proper names cross-language transcription is an essential problem in many spheres — from linguistic topics, like machine translation or information retrieval, to some purely practical ones — for example, when translating documents or maps.

There are several ways to reproduce names with the means of other language:

- Translation (for example, Easter Island — остров Пасхи). Interpreters rarely use this way. Translation is impossible in many cases as proper names usually don't have any lexical meaning.
- Transliteration:
  - Strict transliteration — every letter of alphabet of the source language is associated with a letter in target language. This way of transfer can misrepresent phonetic appearance of the word as almost every language has di- or trigraphs — that is set combinations of letters that should be read in a specific way. Even rules of extended transliteration (rules that allow to transform one letter of source alphabet to two or more letters of target alphabet) are not always sufficient to define all relations between phonetics and graphics.
  - Transliteration with regard for phonetic appearance of the word. This method is usually called practical transcription.

At different times different approaches to translation of names entities were popular among translators, but since the middle of 20th century most of translators agree that name should keep its sounding. Progress in computational linguistics has raised a question of automatic transcription of proper names.

Currently there already exist a lot of various methods of cross-language transliteration. They are based on different approaches and use different techniques: stochastic state-finite automata, Viterbi decoding algorithm, learning of statistic machine translation systems. Vast majority of existing methods are based on statistics. This approach is often effective because it doesn't need to involve specific linguistic information. But its simplicity is paid with necessity in huge amounts of learning data, which is often inaccessible. While other groups of researches develop methods of automatic data retrieval or generate cross-language phoneme or letter mappings using monolingual corpora, we tried to work out a "clever" method of rules generation.

Our work is based on the system "Transscriba" [11]. This is a rule-based system, that means that transliteration model is a set of rules, constructed manually by expert. Such approach provides very high accuracy of transliteration, but it takes from two weeks to six months of an expert's work to construct a system of rules for one pair of languages. Moreover, "Transscriba" has one more drawback. Its method of strings transformation is ineffective as speed of parsing depends on amount of rules in the system. In Section 4 we demonstrate effective method of strings processing with generated rules. Section 5 introduces the method of automatic learning of cross-language transcription rules from small training set.

2. Related works

First attempts to work out system of rules of transliteration relate to pre-computer epoch. There are plenty of works that should rather be treated as recommendations for translator than a code of laws, but these recommendations have later become basis of formal rules used in machine transliteration systems [9, 10].

As for machine transliteration itself, one of the first works that had determined direction of many researches in this area is work [3]. It describes transliteration and back-transliteration between English and Japanese languages. The model is trained with modified Viterbi algorithm. Transliteration is accomplished by a chain of statistical finite-state transducers. Output of every automaton is an input of the next one.

Later this method was adapted for the Arabic language [1]. However, fundamental principle of those works was recognition of separate characters and their groups. It didn't allow to raise quality of transliteration. So researches started working with chained substrings [5]. Such replacement allowed to improve transliteration accuracy from 30% to 90%.

The above mentioned chain of finite-state automata served as a basis for many other methods. Jonathan Grael has developed an instrument that constructs a chain of automata that can be trained on user's data. This tool was widely used in many works on machine transliteration [2].

Another popular method is learning of cross-language mappings using phrase-based machine translation systems. While during translation minimal unit is a word and sentences are regarded as word successions, during transliteration minimal unit is a letter and main analyzed unit is a word.

There are a lot of techniques of machine transliteration. As we can't specify all of them in this paper, we will name main parameters in which different methods vary:

- **Letter / phoneme substitution** — some methods work with letters and substrings [5] and others transform letters in phonetic notation and look for phoneme cross-language mappings [3]
- **Statistical / rule-based models** — cross-language mappings can be acquired with statistic analysis of test data or using some heuristic model
- **Manual / automatic generation of learning data** — for statistic-based models size and quality of test data is very important. Some researches are satisfied with manually-constructed sets, others use multilingual dictionaries of names and terms [3, 5, 7], acquire parallel examples from bilingual corpora [6], or even learn on unilingual data [4].

3. Preliminaries

We will speak about transduction of word from one language to another. So our method deals with pairs of languages, one of which is a source language (language of original) and another is a target language (language of translation). Let we denote alphabet of source language as $\mathbf{V}_\mathbf{l}$, and alphabet of target language as $\mathbf{V}_\mathbf{o}$.

Let we denote letters of $\mathbf{V_o}$ and $\mathbf{V_I}$ with letters of Latin alphabet, strings of letters from $\mathbf{V_I}$ and $\mathbf{V_o}$ are denoted with letters of Greek alphabet. Letters *i*, *j*, *m* and *n* are reserved for enumerations.

The aim of present research is to work out a method that allows to generate cross-language letter mappings: in other words, to determine substitutions for letters or substrings of source language among letters or substrings of target language. Let we define rules of transliteration from source language into target language as these letter mappings.

In our implementation rule is a pair $\mathbf{r} = <\mathbf{p}, \beta>$, where:

$\mathbf{p} = <\mathbf{p_l}, \alpha, \mathbf{p_r}>$, where $\mathbf{p_l}$ and $\mathbf{p_r}$ are pre-condition and post-condition, α is a transduced string, $\mathbf{p_l} = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$, $\gamma_i \in V_I^+$, $\mathbf{p_r} = \{\delta_1, \delta_2, \ldots, \delta_m\}$, $\delta_i \in V_I^+$,

β — output string. String that substitutes for string α in target language.

Consequently the rule is applicable to the current position means that symbols from α are following from the current position, α is preceded by a substring that has symbols from $\mathbf{p_l}$ on appropriate positions and followed by a substring that has symbols from $\mathbf{p_r}$ on appropriate positions.

We sequentially look for rules that can be applied to the input string. If we find one, we move the current position by $|\alpha|$ symbols from current position and return β.

We learn rules of transliteration from a manually-constructed learning set, which consists of proper names in source language and their transliterations into target language.

During the process of rules generation we need some more information about the rule: how many times and in which situations it was applied. So when learning we use full rule format, that is represented as a triplet $\mathbf{r} = <\mathbf{p}, \beta, \mathbf{w}>$, where $\mathbf{p}$ and β are the same as in format listed above and $\mathbf{w} = \{<w_1, pos_1>, \ldots, <w_n, pos_n>\}$, where w_i is a word from the learning set that satisfies the rule $\mathbf{r}$, pos_i is a number that indicates position of the first symbol from α in w_i.

4. Method of string transformation

We needed a method of strings processing which is linear with respect to length of string and doesn't depend on amount of rules in the system.

We have decided to transform strings with state finite machine as it provides linear speed of parsing. We use an extended finite state transducer of the following structure: $g = <\mathbf{V_I}, \mathbf{V_O}, \mathbf{Q}, q_0, \mathbf{F}, \delta>$

$\mathbf{V_I}$ — input alphabet (source alphabet);

$\mathbf{V_O}$ — output alphabet (target alphabet) ;

$\mathbf{Q}$ — automaton's states set;

q_0 — initial state;

$\mathbf{F}$ — final states set;

δ — next-state function $Q \times VI \rightarrow <Q, a>$. It defines state we should move from the current state if we receive a certain input symbol, $a \in A$ is a set of actions committed during transition to the new state. A = {Out(), Shift()}:

Out(ω) — function returning a substring ω (can be empty) from the target alphabet.

Shift(*n*) — procedure that skips *n* symbols of the processed string. The default value of *n* is 1, that means that during a transition automaton moves to the next symbol.

Every final state has a transition to the initial state by the empty symbol.

The automaton is constructed from a set of rules (see section 3). Rules are sequentially added to the automaton. We start adding every rule from the initial state of the automaton.

If rule doesn't have context, that is $r(\mathbf{p}_l) = r(\mathbf{p}_r) = \varnothing$, we use the following algorithm of transformation.

Algorithm 1. Transformation of rule without context.

1. Add new state and move from the initial state to the new state by the first symbol from α
2. Move from current state n_i to state n_{i+1} by every following symbol of α (state n_{i+1} is newly constructed)
3. State n_z which we have come by the last symbol of α becomes a final state
4. Add action Out(β) to the transition to the final state n_z. In other words, if we meet substring α in a processed string, we add its transliteration (β) to the output string.

Note that value of function Shift() for every constructed transition is 1.

If rule has contexts, the procedure of its transformation is a little different.

Algorithm 2. Transformation of rule with context.

1. Add new state and move from the initial state to the new state by the first symbol from α and set Shift() = $|\gamma| \times (-1)$, $\gamma \in \mathbf{r}(\mathbf{p}_l)$ — after checking the first symbol we return back to check context;
2. Commit step 2 of Algorithm1 for every $\gamma_i \in \mathbf{r}(\mathbf{p}_l)$,
3. Commit step 2 of Algorithm1 for α
4. Commit steps 2–4 of Algorithm1 for every $\delta_i \in \mathbf{r}(\mathbf{p}_r)$ BUT set meaning of Shift() for the last transition to $|\delta\text{-}1| \times (-1)$, $\delta \in \mathbf{r}(\mathbf{p}_r)$ — after checking the rule and its post-context we return back to parse context separately.

Note that transformation demands that all $\gamma \in \mathbf{r}(\mathbf{p}_l)$ have the same length and all $\delta \in \mathbf{r}(\mathbf{p}_r)$ have the same length, yet there is no request to $|\gamma|$ and $|\delta|$ to be equal. Moreover, presence of one of contexts doesn't mean presence of the other.

Automaton constructed from system of rules using Algorithms 1 and 2 can turn out to be nondeterministic if there is a pair of rules r_1 and r_2 such that $r_1(\alpha) = <u_1, u_2, \ldots, u_n>$, $r_2(\alpha) = <v_1, v_2, \ldots, v_m>$ and $u_1 = v_1$. Of course one can process strings with nondeterministic automaton, but in this case it loses its advantage of speed. To keep this advantage we use standard procedure of determinization of state automaton [8].

5. Method of rules generation

Algorithm of rules generation can be divided into two main steps:

1. generation of initial rules;
2. generation of complicated rules.

5.1. Initial rules

We call "initial rules" all the rules that can be discovered through rather simple operations. Even so rules themselves can be nontrivial.

The core of rules generation process is association of substrings of original names with corresponding substrings of their translations. In other methods this process is purely statistical, but we use other approach.

Ideally name and its translation should consist of the same phoneme succession to be recognizable. For the initial stage of algorithm let we assume that in every language consonant phoneme is written down with one or more consonant letters and vowel phoneme — with vowel letters. From this assumption we deduce that consonant letters usually transform to consonant letters and vowels — to vowels.

In compliance with our hypothesis we divide each word (both original and translation) into groups of consonants and vowels. Let us define predicate isVowel(l), which returns **true** if l is vowel and **false** otherwise for every $l \in \mathbf{V}_\mathbf{I} \cup \mathbf{V}_\mathbf{O}$. For every word w = $l_1 l_2 \ldots l_n$ bound of group is placed between all such l_i and l_{i+1} that isVowel(l_i) ≠ isVowel(l_{i+1}).

So each name can be represented as a pair w = <**in**, **out**>, where:

in = $<in_1, in_2, \ldots, in_n>$ — set of nonempty chains of letters from $\mathbf{V}_\mathbf{I}$

out = $<o_1, o_2, \ldots, o_m>$ — set of nonempty chains of letters from $\mathbf{V}_\mathbf{O}$

Then for every word w from learning set, where |**in**| = |**out**|, we form n rules where n = |**in**| = |**out**|

$\mathbf{r}_i(\mathbf{p}_l) = \mathbf{r}_i(\mathbf{p}_r) = \varnothing$, $\mathbf{r}_i(\alpha) = in_i$, $\mathbf{r}_i(\beta) = o_i$.

In other words, we just match i-th group of original name with the i-th group of translated name, provided that name and translation have equal number of groups and i-th group of original have the same type (consonant or vowel) as i-th group of translation (see ex. 1). These mappings form the set of rules-candidates **R**.

Example 1

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

Example of generation of initial rules from names (Italian-English dataset). Vertical lines denote bounds of groups. Corresponding groups of original names and translations are united into rules

After having generated set of rules-candidates we reduce it. We remove rare rules: rules that occur in our set only once or twice. We consider them to be arbitrary letter combinations. Then we remove too big rules — rules whose left side is longer than 3 symbols. We suppose that it can be later explained with several shorter rules. Of course, this solution isn't always correct as one sound may be written down by four or more letters. So we don't remove rule with left side longer than 3 letters if its right side consists of only one letter.

We also remove rules that can be explained with shorter rules. Formally speaking, if for rule $\mathbf{r}_0$ = <**p**, β> exist $r_1, \ldots, r_n \in \mathbf{R}$ such that $\mathbf{r}_0(\alpha) = r_1(\alpha) + r_2(\alpha) + \ldots + r_n(\alpha)$ and $\mathbf{r}_0(\beta) = r_1(\beta) + r_2(\beta) + \ldots + r_n(\beta)$, then $\mathbf{r}_0$ should be removed.

After these operations the system of rules is rather adequate except of one detail. It contains ambiguous rules. We call rules $\mathbf{r}_1$ and $\mathbf{r}_2$ ambiguous if $\mathbf{r}_1(\alpha) = \mathbf{r}_2(\alpha)$, $\mathbf{r}_1(\beta) \neq \mathbf{r}_2(\beta)$ and $\mathbf{r}_1(\mathbf{p}_l) = \mathbf{r}_1(\mathbf{p}_r) = \mathbf{r}_2(\mathbf{p}_l) = \mathbf{r}_2(\mathbf{p}_r) = Æ$. Such cases can be sometimes explained with ambiguities of reading rules of the source language, but we should try to resolve them with the help of contexts. Up to now all the rules in our set had empty contexts ("$\mathbf{r}\hat{I}\mathbf{R}$: $\mathbf{r}(\mathbf{p}_l) = \mathbf{r}(\mathbf{p}_r) = Æ$). As all of our rules contain reference to words where they meet we can add contexts. Contexts are letters which precede ($\mathbf{p}_l$, left context) and follow ($\mathbf{p}_r$, right context) substring $\mathbf{r}(\alpha)$ in words where it is met. This approach is useful in many cases (see example 2).

Example 2

After employing contexts we receive from two ambiguous rules c → c and c → κ (French-Russian dataset) two rules:

{< a e i}c{a o} → κ

{< a e i u y}c{e i} → c,

that illustrate one of French reading rules: c is read as [s] ("c" in Russian) before front vowels (e, i).

5.2. Complicated rules

After the first stage of the algorithm we achieve a system of rules that can already be used for strings conversion. If reading rules of source language are plain enough, system of initial rules can transform strings correctly. But in many cases initial rules are not sufficient.

The second step of the algorithm aims to discover rules that can't be discovered at the first step.

The second step of the algorithm consists of several minor steps:

1. Divide names into syllables
2. Try to transform syllables according to the existing rules, generate new rules
3. Go to step 2

We divide all the names and their translations into syllables. Then we try to convert every syllable from source language to target language. If the conversion was failed, that means that the translation of the syllable can't be explained with the existing system of rules. In this case we add a new rule that can explain current syllable. Then, if there are any unexplained syllables left, we repeat step 2 until all of them are explained.

5.2.1. Syllabification

Term "syllable" in the present work is used not in traditional linguistic meaning.

Syllable is nonempty substring of a given word containing one or more vowels. We use the following rules of syllabification:

- Bound of a syllable in the word w = $l_1 l_2 \ldots l_n$ is between a vowel and a consonant, i.e. after letter l_i such that isVowel(l_i) = **true** and isVowel(l_{i+1}) = **false**;

- Set of elements none of which is vowel isn't separated out, including consonants at the end of word;
- Symbols "<" and ">" marking the beginning and the end of word are considered consonants.

Thereby a syllable is a chain C^*V^+ where C is a consonant and V is vowel. Syllable can have form $C^*V^+C^+$ only if it is a last syllable of a word and final set of consonants can't be separated because there is no vowels among them.

Let we define substitution as pair of strings $\zeta_i \to \eta_i$, where $\zeta_i = <u_1, \ldots, u_n>$ is an i-th syllable of original word and $\eta_i = <v_1, \ldots, v_m>$ is an *i*-th syllable of translated word.

Actually syllable is a combination of consonant group and vowel group, that were described in section 4.1.

5.2.2. Trial transformation

We divide words into syllables, because syllables are shorter and in a syllable it's easier to discover a substring that can't be processed with the existing rules, than in a word.

We try to apply existing system of rules to every syllable ζ of original word. If we get syllable η which is right part of substitution $\zeta \to \eta$, we move to the next syllable. Otherwise we should generate a new rule as any subset of existing rules doesn't give us proper result.

Applying rules to the substitution $\zeta \to \eta$ from left to right and from right to left we can represent the substitution as $<u_1, \ldots, u_k, \lambda, u_{k+1}, \ldots, u_n> \to <v_1, \ldots, v_q, \mu, v_{q+1}, \ldots, v_m>$, where $<u_1, \ldots, u_k> \to <v_1, \ldots, v_q>$ and $<u_{k+1}, \ldots, u_n> \to <v_{q+1}, \ldots, v_m>$. So we have three different situation depending on α and β:

- $\lambda = Æ$, $\mu \neq Æ$. We add a new rule r_i such that $r_i(\mathbf{p}_l) = u_{k-1}$, $r_i(\alpha) = u_k$, $r_i(\mathbf{p}_r) = u_{k+1}$, $r_i(\beta) = v_q + \mu$. — in other words, rule for symbol that precedes α. In some contexts it is substituted with two or more symbols.
- $\lambda \neq Æ, \mu = Æ$. This situation means that some of letters of $\mathbf{V}_I$ in some contexts are not read. We add a rule r_i such that $r_i(\mathbf{p}_l) = u_k$, $r_i(\alpha) = \lambda$, $r_i(\mathbf{p}_r) = u_{k+1}$, $r_i(\beta) = Æ$.
- $\lambda \neq Æ$, $\mu \neq Æ$. We add r_i such that $r_i(\mathbf{p}_l) = r_i(\mathbf{p}_r) = Æ$, $r_i(\alpha) = \lambda$, $r_i(\beta) = \mu$. If r_i conflicts with any of exiting rules we add context to r_i.

6. Experiments

6.1. Generated rules

We evaluated our method of rules generation on parallel collections of names in French, German, Spanish, Swedish, Mongolian, Arabic and Japanese. Target language of all test collections is Russian. Collections contain proper names from various sources. They were transcribed into Russian manually by an expert. We didn't estimate fullness of collections.

We received rules of transliteration from every of listed languages into Russian. Table 1 summarizes information about size of test collections and number of generated rules.

Table 1. Results comparison

Language	Size of collection	Number of rules generated with our tool	Number of rules written by expert
Arabic	1900	63	78
French	1900	160	356
German	4200	102	121
Japanese	7000	52	131
Mongolian	230	41	46
Spanish	1000	88	106
Swedish	1900	105	423

Systems of rules that were generated by our system contain fewer rules than systems written by experts. This fact can be explained in two ways. First of all, although experts relied on test collections while writing rules, they often followed their own knowledge of source language and added rules that could not have been deduced from test examples. Thus expert added rule "{<}ai → э" ("ai" in the beginning of the word should be transliterated as "э") in French-Russian rule system, despite the absence of suitable examples in the collection. Some of such rules were generated by our tool, but then excluded as rare and insignificant. And some others were just not generated. For example, expert added rules "aa → a" and "ya → я" for Arabic as well as machine did, but rule "yaa → я" exists only in human-written rules set. Expert's set of rules for Japanese-Russian transliteration contains rules for syllables "ha", "sha" and "cha", transcribed as "ха", "ся" and "тя", respectively. However, algorithm considers only one-symbol contexts, so it generated rule "a → я" with left context "h".

Secondly, rules that were generated by computer are compressed in comparison with expert's rules. For example, expert's variant of French-Russian rules contains a set of rules for substring "ai", while computer generated only one rule "ai → e", that covers all expert's transliterations (except of above-mentioned case of "ai" in the beginning of the word). Rules of Japanese syllabary transliteration were reduced in rules for particular substrings. Thus instead of three rules for syllables that start with "ch" machine generated one rule "ch → т". We should admit that this approach is more proper, but linguists are not used to such notation.

Aside from mentioned drawbacks the rules are consistent, cover major part of test collection and can be used for transliteration of proper names.

6.2. Finite State Automaton

Rules generated by our system were also used to construct finite-state transducer. We have checked quality of learning of our method. Finite state machine has transduced names from the training set. The method has shown rather good results (see Table 2), low values at the bottom of the table can be explained with inconsistencies or mistakes in training set.

Although the method already applicable to practical tasks, it can be still improved with statistics and more full usage of contexts, so results in Table 2 are not ultimate.

Table 2. Quality of learning

Language	Size of collection	Quality of learning
French	580	97%
Mongolian	232	98%
Tagalog	286	93%
Arabic	1606	87%
Spanish	1041	86%
Polish	1413	79%
Romanian	576	78%
Swedish	1629	74%

References

1. *Akho A. V., Lam M. S., Seti R., Ul'man D. D.* 2008. Compilators. Priciples, Technologies and Instruments [Kompiliatory. Printsipy, Tekhnologii I Instrumentarii].
2. *Al-Onaizan Y., Knight K.* 2002. Machine Transliteration of Names in Arabic Text. Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages.
3. *Ermolovich D. I.* 2005. Proper Names: Theory and Practice of Interlanguage Transmission [Imena Sobstvennye: Teoriia I Praktika Mezh"iazykovoi Peredachi].
4. *Giliarevskii R. S., Starostin B. A.* 1985. Foreign Names in Russian Text [Inostrannye Imena I Nazvaniia v Russkom Tekste].
5. *Graehl J.* 1997. Carmel Finite-state Toolkit, available at: http://www.isi.edu/licensed-sw/carmel
6. *Knight K., Graehl J.* 1998. Machine Transliteration. Computational Linguistics, 24(4) : 599–612.
7. *Practical* transcription of Proper Names in the Languages of the World [Prakticheskaia Transkriptsiia Lichnykh Imen v Iazykakh Narodov Mira]. 2010.
8. *Ravi S., Knight K.* 2009. Learning Phoneme Mappings for Transliteration without Parallel Data. Human Language Technology Conference archive Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics.
9. *Sherif T., Kondrak G.* 2007. Substring-Based Transliteration. Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages.
10. *Sproat R., Tao T., Zhai C.* 2006. Named Entity Transliteration with Comparable Corpora. Proc. of ACL.
11. *Zelenko D., Aone C.* 2006. Discriminative Methods for Transliteration. Proc. of EMNLP.