

AUTOMATIC JAPANESE TRANSLITERATION WITH A FORMALISM FOR PRESYNTACTIC ANALYSIS

Julien Quint

National Institute of Informatics, Hitotsubashi 2-1-2-1312, Chiyoda-ku, Tokyo 101-8430, Japan
quint@nii.ac.jp

ABSTRACT

A Japanese transliteration system, primarily used for the romanization of Japanese dictionary entries, is presented. As the system was implemented in a quick and dirty way, we show how to reimplement it cleanly and efficiently with the help of the Sumo formalism, a framework for the development of presyntactic applications. This paper describes Sumo, its foundation and control language, and outlines the implementation of the romanizer using the Sumo tool.

INTRODUCTION

The Japanese language has a notoriously complex writing system, mixing *hiragana* (ひらがな), *katakana* (カタカナ), *kanji* (漢字), and roman characters (*rōmaji*). Transliteration, the process of rewriting a text using a different set of writing conventions, is useful when dealing with such a complex writing system: phrase books use *rōmaji* so that their readers are able to read the Japanese sentences; Japanese input systems provide a way to input thousands of different *kanji* using a regular keyboard and roman characters or *hiragana*.

Transliteration is a perfect example of a presyntactic linguistic task: linguistic knowledge is needed, but a full syntactic analysis of the text is not necessary. The term “presyntactic analysis” is convenient as a shorthand for grouping together all the steps of analysis that take place before syntactic analysis. The main ones are text segmentation, morphological analysis, part-of-speech tagging, noun phrase extraction and shallow parsing. Note that what is often called “preprocessing” (normalization, transcoding, etc.) is included too.

This paper describes an automatic system for the

romanization of Japanese. The system transcribes any text from *kanji* to *rōmaji* without any user intervention. The original task intended for this system is to provide example fields in *rōmaji* form in the entries of a bilingual dictionary, so it must be able to deal with full sentences of unrestricted Japanese text. It can as a result be applied to any Japanese text, like a Web page.

This application is a good example of a common phenomenon in the domain of presyntactic applications: at first, a quick and dirty implementation is made and it evolves from there. Then, as new features and improvements are added, the system becomes more and more crafty. However, for a presyntactic application of the sort, there existed no formalism in which to express the problem in an elegant and efficient way.

Sumo (*Segmentation Universelle Multiple par Ordinateur*), aims at filling this gap. Its initial goal was to provide a generic, universal framework for the development of text segmentation applications in any language [6, 7]. It became quickly obvious that since segmentation has strong ties to other presyntactic processes (notably morphological analysis), the coverage of Sumo needed to be broadened to all sorts of presyntactic applications. It is based on the following observations and ideas.

1. Although there is no unified formalism for presyntactic analysis, finite-state techniques are widely used in the field, albeit in different contexts and flavors. Finite-state calculus is well-defined and therefore a very sound basis on which to build Sumo.
2. All languages have different views of what constitutes a word, a phrase or a sentence [2]; often, native speakers of the same language do not agree on this. Some languages, such as thai [10],

are very complex in that respect. Sumo addresses this multilingual issue by providing an open document structure that allows to manage several level of segmentation at once.

3. There is a large variety of processes in the domain of presyntax, so Sumo also provides high-level control structures to program any application using the Sumo data structures.

These three points are detailed in the first three sections. They are then illustrated by the real-life example of the romanizer described above.

1. FINITE-STATE CALCULUS

Regular expressions are a well-defined formalism, implemented very efficiently by finite-state automata. Many linguistic phenomena can be accurately described by means of regular expressions and compiled to finite-state devices. These devices are finite-state automata (FSA) which test whether a given word belongs to a regular language, and finite-state transducers (FSTs), which express a regular relation between two languages.

A very interesting advantage of finite-state calculus is that all of the data manipulated during text processing is in the form of automata and transducers. As an example, a morphological analyzer is compiled into a FST mapping surface forms to lexical forms. Morphological analysis is the application of the transducer to the input text, itself converted trivially to a FSA.

Another advantage is that it is possible to reverse a FST: the morphological analyzer then becomes a generator, mapping lexical forms to surface forms. A use of this property is found in section 5.2.

Sumo provides the ability to compile FSAs and FSTs and combine them in any way. The usual rational operations are available: concatenation, complement, composition, intersection, Kleene star, Kleene plus, projection, difference, cross-product, optionality, inversion, union and reversal (note that some of them only apply to FSA), as well as determinization and state minimization.

1.1. WEIGHTED FINITE-STATE CALCULUS

Sumo makes use of several extensions that have been proposed to classical finite-state calculus. The first one

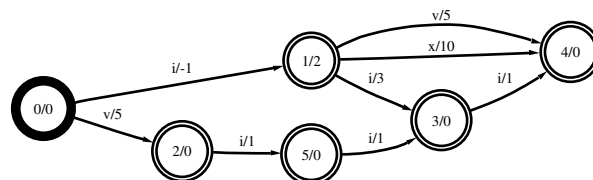


Figure 1: WFSM recognizing roman numerals 1 through 9

is the addition of weights in automata and transducers to make weighted finite-state automata and transducers (WFSAs and WFSTs.) Arcs and initial and final states are weighted, so that it is not only possible to verify if a word belongs to the language but also to give it a cost. It is then possible to apply statistical techniques to disambiguate analyses made by transducers by selecting paths with optimal cost.

In NLP, weights are often defined over the so-called *tropical semi-ring*. Weights are real numbers. The weight of a path in a transducer is the sum of all weights in the paths (initial state, final states, and traversed arcs). If a word is matched by several paths, the one with minimum weight is selected. Figure 1 shows such a transducer which recognizes roman numerals between 1 and 9 and associates a weight corresponding to the value of the number (e.g. the string *iv* has a weight of $0-1+5+0=4$).

Weighted finite-state transducers are closed under all the regular operations defined by Sumo. By default, Sumo uses only weighted finite-state transducers; it is trivial to build unweighted transducers in a weighted case. Sumo also borrows the weighted rewrite rules of [5]. These rules provide syntactic shorthands for the definition of complex transducers, as well as a basis for the identification rules that will be shown later.

1.2. OTHER EXTENSIONS

But what about the presyntactic phenomena that cannot be described by regular expressions? If a phenomenon can only be described by a context-free grammar, then simple finite-state devices cannot deal with it. A way around this problem is to apply the same transducer to the input repeatedly until a fixed point is reached [9]. Note that this process is not guaranteed to ever halt, as a fixed point may never be reached. Sumo defines the operator *%** for such an operation (*%* being the operator for composition, and the star recalling Kleene's star.)

Another very powerful extension to regular expressions is “synchronized” regular expressions [1]. With this very powerful extension, it is possible to handle phenomena that even context-free grammars cannot describe, such as reduplication (*e.g.* plural in Malay). An example can be shown in Perl, where the regular expression syntax allows for “backreferences.” The pattern

```
/^(\\w+)\\1$/
```

matches a string if and only if it is of the form *ww*. The first part of the pattern (`(\\w+)`) matches a sequence of alphanumeric characters; the second part (`\\1`) matches the sequence of characters that was previously recognized. This extension is not available as such in the finite-state component of Sumo; however, identification rules allow this construct.

2. MANAGING DOCUMENTS IN SUMO

2.1. THE SUMO DOCUMENT STRUCTURE

Sumo stores a textual document as a layered structure, where a layer is a view of the document at a given level of segmentation. Levels of segmentation are chosen freely and have a linguistic signification (like words or sentences) or not. This document structure is used throughout the processing, representing the current state of the document.

At a given level of segmentation, all document elements are atomic units. They cannot be divided any further, but can be combined to form elements that belong to a higher level (*e.g.* words can be combined to form sentences). In Sumo, processing is usually done bottom-up, and segmentation is viewed not so much as a division of a large text in smaller units, but as the combination of elementary units (starting from letters) to form higher-level constituents.

These atomic document elements are called segmentation items, or just “items.” A segmentation level is a collection of such items. Since at any level of segmentation there may be a large number of possible combinations (*i.e.* segmentation ambiguity), they are stored in a directed acyclic graph (DAG) where every path corresponds to a possible segmentation. This representation technique is also used in speech recognition systems. And since Sumo is built on finite-state

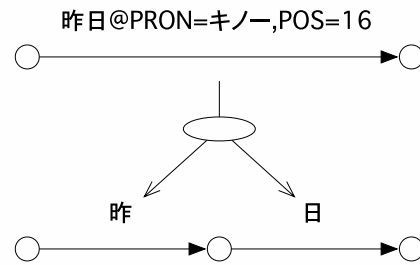


Figure 2: Sumo document fragment for 昨日

foundations, the DAGs are actually acyclic WFSAs, where arcs are weighted and labelled with segmentation items.

Sumo items are more than just the string label of an arc. They consist of a handle string and a list of attributes. An example item is:

```
昨日@PRON=キノー, POS=16
```

This is the morpheme 昨日 (*kinō*, yesterday) with two attributes, PRON (for pronunciation) and POS (part-of-speech).

Lastly, segmentation levels are linked together by relations of composition. These are oriented relations between two sub-paths (consecutive sequences of items) in two adjacent. The item 昨日, at the morpheme level, is linked to the two characters 昨 and 日 at the character level, by such a relation. All morphemes are linked thusly to characters. Figure 2 shows a graphical representation of such relations and illustrates a fragment of a Sumo document with two levels, characters (at the bottom), and morphemes. In this example, the weights were omitted as they are not used in the romanizer.

2.2. OPERATIONS ON THE SUMO DOCUMENT STRUCTURE

The structure can be manipulated through several sets of operators acting at different levels. As a level in the document is actually a WFSA, then the level itself can be manipulated just like any other WFSA with rational operators as defined in section 1.

These rational operators act on existing levels in the structure. New levels are created in two ways: first, by explosion of a transducer into two automata, linked by trivial composition relations. The second way is the application of identification rules. These rules are similar

to weighted rewrite rules; they identify items in level n and combine them to create new items in level $n+1$, creating the composition relations at the same time. The morpheme level in figure 2 may have been created by the application of the following rule to the character level:

昨日 -> 昨日@PRON=キノー, POS=16

This rule is read: when the sequence of two items 昨 and 日 is read (in the character level), create a new item 昨日 in the word level, with the corresponding attributes. Such rules can become more complicated as conditions and variables can be introduced, e.g.

$$\begin{aligned} \$a \$b => [\$a \$b] \quad || \quad ?\{\$a \text{ eq } \$b\} \\ \{\$a = (a|b)^*, \$b=(a|b)^*\} \end{aligned}$$

recognizes the word of the form ww over $\{a, b\}$.

It is also possible to delete and extract levels from the structure; as compositional relations are transitive, when a level is deleted, composition relations can be redirected. In Sumo, new levels are often introduced, and then discarded; once again, this *modus operandi* can be traced back to finite-state techniques: a given task divided in several simple subtasks, all corresponding to a transducer; the composition of all these transducers create a single transducer handling the original task.

Lastly, Sumo defines pseudo-rational operations on the document structure itself. As an example, the union of two structures (having the same number of levels) is realized by the union of the WFAs at each level.

3. THE SUMO CONTROL LANGUAGE

Typically, a Sumo application processes a document in a bottom-up fashion; starting from a seed level, further levels are built in order to produce the analysis of the text, by selecting the desired information from the structure. However, Sumo does not impose any one way to carry on an analysis.

A powerful control language, similar in many ways to Perl, allows to build complete applications that can operate on the structure arbitrarily and can provide some level of user interaction. A command-line interface is also provided to interact with Sumo through a shell.

4. AUTOMATIC ROMANIZATION OF JAPANESE

Reading a Japanese text is very difficult as *kanji* have a different pronunciation depending on their context. Romanization aims at producing a string of *rōmaji* characters corresponding to the correct pronunciation of a Japanese text. In this sense, this task is actually very similar to text-to-speech synthesis, where a phonetic representation of the text is built for a speech synthesizer. The example below shows a sentence written in *kanji* (meaning “Yesterday, I went to Jimbōchō.”) and its romanized form:

私は昨日神保町へ行きました。
Watashi wa kinō Jimbōchō e ikimashita.

4.1. ROMANIZATION SYSTEMS

The romanization produced above is just one of the many possible romanizations for the same sentence. There are four main romanization systems: the Hepburn system (ヘボン式ローマ字) which is the one used above; the modified Hepburn or *hyōjun* system (標準式ローマ字); the *kunrei* system (訓令式ローマ字); and the *nihon* system (日本式ローマ字).

The first two have a Western origin and can represent more “modern” forms (like ヴァ, *va*, or トウ, *tu*.) The last two were designed to be more faithful to the Japanese phonological system. In the *kunrei* system for instance, the example above would come out as “*Watasi wa kinō Zinbōtyō e ikimasita.*”

Outside of these systems, there are also different spelling variants. When macrons (as in \bar{o}) are not available, one can use accents (\hat{o}) or a digraph (*ou* or *oh*). It is important that the romanizer produces a consistent output, as opposed to people who seem happy to mix and match systems and spellings; but at the same time, it should not impose any system on its users and let them choose which one they prefer.

4.2. PRONUNCIATION ISSUES

Japanese can be written using only *kana* characters. This form of writing can then easily be translated into *rōmaji* as every romanization scheme defines a fairly straightforward mapping between *kana* characters and *rōmaji* strings.

However, Japanese pronunciations has some quirks that must be taken into account. As an example, the

character は is realized as *ha* in all systems, but as in the sample sentence above, the *particle* は is actually pronounced *wa*. In the same fashion, the particle へ, normally *he*, is really pronounced *e*. There are a few other cases that need part-of-speech information (*i.e.* this is a particle) to be handled correctly.

4.3. AUTOMATIC ROMANIZATION WITH THE CHASEN MORPHOLOGICAL ANALYZER

It is now possible to outline a method for automatic romanization of unrestricted Japanese text:

1. transliterate all *kanji* to *kana*, then
2. translate all *kana* to their romanized form using a transliteration table for the chosen romanization scheme.

The prototype system is built on top of ChaSen [4], a morphological analyzer for Japanese. ChaSen takes care of the first step by providing a pronunciation entry (in *katakana*) for every word in its dictionary. Moreover, the output of ChaSen can be configured freely. Since the pronunciation and the part-of-speech of every word is needed, only this information is selected. The result of the morphological analysis for the example sentence looks like (for the sake of clarity, the original form was omitted):

ワタシ/14 ワ/65 キノー/16 ジンボーチ
 ヨー/11 エ/61 イキ/47 マシ/74 タ/74 。/78

The words are separated by a white space. The part-of-speech of every word is attached (here as a numerical code) after a slash. Note that punctuation needs also to be rewritten and is tagged accordingly.

The second step is then to rewrite the output from ChaSen in *rōmaji* by following a given romanization system. To this end, transliteration tables are used, mapping all characters to their roman equivalent. Moreover, some special cases need to be taken into account:

- the character ー following a vowel indicates a long vowel (*e.g.* アー gives *ā*);
- the character ッ (small ッ) followed by a consonant indicates a long consonant (*e.g.* ニッコウ gives “Nikkō”); at the end of a word, it is realized as ’;

- the character ヂ at the end of a word is realized as *n’*;
- in the Hepburn system, ヂ is realized as *m* before a *b* or a *p*.

There is also a lot of special characters (punctuation, roman characters in double-width, etc.) to take into account because the Japanese character sets are so large.

Note that ChaSen gives the correct information for the pronunciation of particles. It also recognizes proper nouns in a text, so that they can be capitalized: see “Jimbōchō” or “Nikkō” above.

4.4. REMAINING ISSUES

There are two main outstanding issues. The first one concerns segmentation of the sentence. As can be seen from the ChaSen output, the segmentation of the sentence is different from the one desired for the romanized output (namely, イキ + マシ + タ versus “*iki-mashita*.”) ChaSen prefers a very fine-grained tokenization, which makes sense as we have argued above (the segmentation units are atomic and cannot be split further, but can be recombined), however, the result can be confusing for a student of Japanese.

The solution is then to recombine some tokens by using the part-of-speech information. Currently, the system is quite naive and can only recognize morphemes marked as prefix (which are glued back to the following token), and suffix or postposition (which are glued back to the previous one). Such a simple approach is unfortunately insufficient and a more complex grammar has to be written, using the part-of-speech information given by ChaSen.

The second outstanding issue has to do with long vowels. Sequences such as *aa* or *ou* should usually be realized as *ā* or *ō*. But consider the word 場合, バア イ. It must be realized as *baai* and not **bāi*, because it is composed of two *kanji*, *ba+ai*, and the “long vowel rule” does not apply across *kanji* boundaries.

These cases are tricky to handle because it is necessary to know not only the pronunciation of the words, but also the individual pronunciation of all of its constituent *kanji*. ChaSen does not provide this information; it must therefore be recovered using a *kanji* dictionary.

5. THE ROMANIZER IN SUMO

In this section the implementation of the romanizer is sketched roughly to give a better idea of what Sumo is. As the goal is not (yet!) to rewrite ChaSen, steps one and two of the prototype are kept. Therefore, the core of the system will be a transliteration WFST mapping *kana* to *rōmaji*.

Actually, four different transducers are built, as a different transducer is needed for every system. Every transducer is built in the same way as they are mostly equivalent. A transducer is built step by step, taking care of the the order of application, then all composed into the final transliteration transducer.

First step is the “regular” transliteration of *katakana*. This is separated into two sub-steps: first, the digraphs (such as キヤ, *kya*), then the single *katakana*. This is easily done by rewrite rules:

```
( キ ヤ -> k y a || ... ) %
( カ -> k a || キ -> ki ... ) ;
```

The next steps are the same as shown in section 4.3. Note that this transducer only works on characters (*katakana* and roman letters), so it only does the “pure” transliteration operations and leaves the operations needing more information (capitalization, long-vowel processing, resegmentation) to the parent application.

5.1. THE SUMO APPLICATION

Sumo’s input is not the original text, but the output from ChaSen. In order to produce a Sumo structure such as the one of figure 2, two options are possible: the first one is to read the output of ChaSen as a regular text file, and write a small Sumo application to transform the structure. The other option is to extend Sumo by writing a small filter application (in Perl or in the language in which Sumo will eventually be implemented) that creates the structure from ChaSen’s output.

The next step is to create a new level of *katakana* characters above the level of words. This is done by extracting the attribute *PRON* of every word and exploding the value into characters. The third (upmost) level is now the *katakana* level. This level can be transformed into *rōmaji* by applying the transliteration transducer. To finish, the middle level (complex words) can be simply erased to produce a two-level structure,

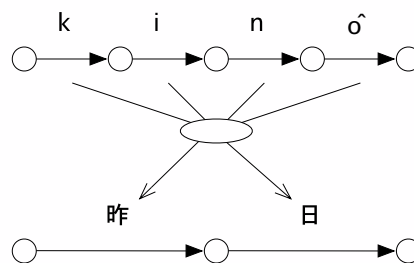


Figure 3: Romanization result for 昨日

showing the correspondance between *kanji* and *rōmaji*. The result structure is shown in figure 3.

This outline is very shallow and doesn’t address many questions such as the insertion of whitespace in the romanized text (whitespace can be inserted in the *rōmaji* output between every word by a simple identification rule); resegmentation (taking place before the transliteration is actually done), etc.

5.2. EXTENDING THE ROMANIZER

The Sumo implementation of the romanizer is extensible and can easily provide more functionality. By reversing the transliteration transducer, a transducer mapping roman characters to *katakana* is obtained, which is really a rudimentary input method. To make it actually useful, it is best to create first the union of the transliteration transducers for all systems: people are inconsistent when they write in *rōmaji* and the input method should be forgiving.

It is also worth noting that this reversal creates ambiguities: there may be several *katakana* sequences for the same *rōmaji* sequence (e.g. by typing “tu,” the user may mean ツ or トウ.) Real-world input methods deal with this sort of ambiguity through user interaction. In this system, it could be interesting to add weights to transliterations, so that more likely candidates are chosen over less likely occurrences.

A simple romanizer can be turned into a more generic transliteration platform in Sumo. The same ChaSen output can also be used to produce *furigana* annotations, that is indication of the pronunciation of words written in *kanji*. This is a good alternative to *rōmaji* for the students of Japanese as it makes them exercise their reading skills.

CONCLUSION AND FUTURE WORK

Although completely defined, the Sumo formalism is still being prototyped. Therefore, the romanizer presented above still lives as a Perl program rather than a Sumo application. Nevertheless, this romanizer is usable and comments and criticisms are more than welcome. The interested reader is encouraged to try the following URL:

<http://papillon.ex.nii.ac.jp/romaji/>

The future of the romanizer was already outlined, the first step being a new implementation in Sumo when the prototype can support it. Extensive testing and evaluation should also be performed; it is not known yet precisely how this romanizer stands up against existing systems such as Kakasi [3], although the current state looks promising.

The initial goal of Sumo is to be completely language-independent. In this paper, the only sort of transliteration that was considered was between *kanji*, *kana* and *rōmaji*. Using the same basic technique to build a romanizer for Chinese, Thai, or Arabic, as well as similar transliteration applications (*e.g.* restoring lost accents in a French text) would illustrate this ability of Sumo.

Another goal of Sumo is to be able to deal with arbitrary levels of text. A good illustration is a new project based around Sumo to provide a platform for semantic annotation of texts [8]. After morphological analysis, or shallow parsing, of a text document, words and entities can be annotated with semantic informations contained in dictionaries that can be looked up in context using the information provided by the presyntactic analysis.

Many thanks go to Ulrich Apel, co-author and initiator of the romanization system presented here. The Sumo system was designed during a PhD project under the direction of Professor Christian Boitet at the GETA-CLIPS-IMAG laboratory in Grenoble and of Jean-Pierre Chanod in the MLTT team at the Xerox Research Centre Europe in France.

REFERENCES

- [1] Giuseppe Della Penna, Benedetto Intrigila, Enrico Tronci and Marisa Venturini Zilli, “Synchronized regular expressions”, *Acta Informatica* 39(1), pages 31–70, January 2003.
- [2] Gregory Grefenstette and Pasi Tapanainen, “What is a Word, What is a Sentence? Problems of Tokenization”, *Proceedings of COMPLEX 94*, pages 79–87, 1994.
- [3] <http://kanjidict.stc.cx/kakasifilt.php>
- [4] Yuji Matsumoto, Akira Kitauchi, Tatsuo Yamashita and Yoshitaka Hirano, “Japanese Morphological Analysis System ChaSen version 2.0 manual”, Nara Institute of Science and Technology, April 1999.
- [5] Mehryar Mohri and Richard Sproat, “An Efficient Compiler for Weighted Rewrite Rules”, *ACL 96*, pages 231–238, 1996.
- [6] Julien Quint, “A Formalism for Language-Independent Text Segmentation”, *Proceedings of COLING 2000*, pages 656–662, Saarbrücken, Germany, August 2000.
- [7] Julien Quint, “Spécification et réalisation d’un formalisme générique pour la segmentation multiple de documents textuels multilingues”, PhD thesis in Computer Science, Université Joseph Fourier, Grenoble I, France, November 2002.
- [8] Julien Quint, “Presyntactic Analysis for Semantic Annotations: A Preliminary Proposal”, *Proceedings of MLABNET 2003*, Nikkō, Japan, March 2003 (*to appear*).
- [9] Emmanuel Roche, “Two Parsing Algorithms by Means of Finite State Transducers”, *Proceedings of COLING 94*, Kyoto, pages 431–435, August 1994.
- [10] Virach Sornlertlamvanich, Tanapong Potipiti, Chai Wutiwiwatchai and Pradit Mittrapiyanuruk, “The State of the Art in Thai Language Processing”, *Proceedings of ACL 2000*, Hong Kong, October 2000.