Technical Report Peer-Reviewed

Inputting Writing Systems with Medium Complexity: A Generalized Input Method Editor AKKHARA and Case Study on Myanmar Script

Chenchen Ding[†], Masao Utiyama[†] and Eiichiro Sumita[†]

In this study, an input method editor called AKKHARA is developed to accommodate writing systems comprising several tens to hundreds of symbols. As an engineering realization, AKKHARA accepts and applies a set of rewrite rules with priorities such that the alternation, substitution, and normalization of character strings are applied alongside the keystrokes. Compared with general key-character editors, AKKHARA provides a greater flexibility for Romanization-based rule editions. Compared with the input methods developed for Chinese and Japanese, AKKHARA is lightweight and easy to maintain. As an application case of AKKHARA, this study illustrates the realization of a Romanization-based Myanmar input method using the Unicode standard. A version of AKKHARA for Microsoft Windows was released that supports Unicode characters with customizable functions for rewriting rule editions.

Key Words: *Abugida writing systems, Input method editor, Myanmar script*

1 Introduction

Various alphabet writing systems have been used to record numerous languages worldwide. These alphabet systems use tens of distinguishable symbols to transcribe phonemes. Conversely, languages, such as Chinese and Japanese, apply logogram systems totally or partially in their orthographies, thus requiring thousands of symbols to represent morphemes.

This study focuses on a group of syllabic systems that lie between the aforementioned extremes. In this report, they are referred to as having a *medium complexity*. Compared with logogram systems, they are still based on phonetics, although a redundancy in spelling may be retained to distinguish morphemes; compared with an alphabet system, they assign symbols based on syllables rather than phonemes, and may apply a nonlinear combination to characters.

Traditionally, these syllabic systems are input by direct keystrokes and treated as common alphabets. Because of the larger symbol set, the keyboard layout is crowded and requires one or

[†] Advanced Translation Technology Laboratory, Advanced Speech Translation Research and Development Promotion Center, National Institute of Information and Communications Technology.

more alternating keys. However, these systems are not as complex as those used in Chinese or Japanese scripts, wherein language-dependent input methods are indispensable. Therefore, an input editor that is 1) lightweight, 2) language-independent, and 3) editable and maintainable, is expected to efficiently input such writing systems.

Motivated by these requirements, the `AKKHARA` editor developed in this study provides a general solution. `AKKHARA` repeatedly applies rewrite operations to convert keystrokes into specific character strings. Therefore, compared with a keyboard layout editor, `AKKHARA` has a greater ability and flexibility in applying string operations than simply appending new characters using keystrokes. This is important for writing systems with certain redundancies in their encoding systems, because some complex combined glyphs can be realized in more than one ways. Thus, the rewrite operations can normalize the input to generate consistently encoded textual data.

Temporary input methods for Chinese and Japanese scripts are typically based on common Romanization systems, such as *Pinyin* and *Rōmaji*. However, numerous languages that do not use the Latin alphabet lack a consistent Romanization system. In `AKKHARA`, a set of rewrite rules is organized in an editable textual file in a structured format, thereby enabling users to customize the input method according to their preferences.

Predictions based on words, phrases, or even sentences, supported by large-scale data, are a common function provided by Chinese and Japanese input methods. Ding et al. (2018) investigated an auto-complete method for several abugida systems that can achieve a high accuracy, even with limited training data. This is a direction for the future development of `AKKHARA`. `AKKHARA` primarily provides a layer for the local decisive conversion between raw keystrokes and writing systems for users to generate correctly coded and normalized strings. Based on this layer, sophisticated functions, such as context-based prediction can be further developed.

The remainder of this study is organized as follows. Section 2 introduces the linguistic background of writing systems. Section 3 reviews related works and applications. Section 4 provides a general overview and instructions for rewriting rule editions to define an input method on arbitrary character sets. To illustrate the application of `AKKHARA`, an input method of the Myanmar script (Ding et al. 2019) is used as a case study. Section 5 reviews the original design of the Myanmar input method and describes `AKKHARA`'s more extendable realization. Section 6 concludes the report and presents ideas for future work.

2 Linguistic Background

Table 1 lists the main types of writing systems used worldwide, with a comparison of the

number of symbols applied. Generally, an analytic phonetic writing system, e.g., a typical alphabet or abjad system, contains approximately 30 ($\approx 10^{1.5}$) symbols. This is because the average phonetic inventory of natural languages contains 22 ± 3 consonants (Maddieson 2013a) and five or six vowels (Maddieson 2013b). By contrast, a logogram system may contain thousands of symbols for daily use. For example, the *Table of General Standard Chinese Characters*[1] contains 8,105 characters for Chinese, and the *List of Jōyō Kanji*[2] contains 2,136 characters for Japanese.

Figure 1 compares character distributions across different writing systems. Parallel data from the Asian Language Treebank (Riza et al. 2016)[3] were used. The differences between an

Type	# Symbol	Example
Logogram	$> 10^3$	Chinese character
Phonogram		
– syllabic	$10^{1.5}$–10^3	Japanese kana
– abugida	$10^{1.5}$–10^3	Devanagari script
– alphabet	$\approx 10^{1.5}$	Latin script
– abjad	$\approx 10^{1.5}$	Arabic script

Table 1 Types of writing systems and the scale of symbols used in them. This report refers to the abugida/syllabic systems as having *medium complexity*.

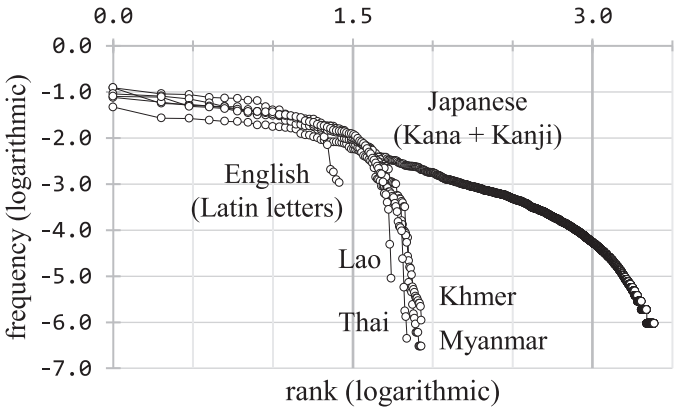

Fig. 1 Character distribution of Latin letters in English, the *kana* and *kanji* in Japanese, and the four abugida systems of Myanmar, Khmer, Thai, and Lao. Frequency is the normalized relative frequency, i.e., empirical probability; the rank is by the descending order of the frequency.

[1] http://www.gov.cn/gzdt/att/att/site1/20130819/tygfhzb.pdf
[2] https://www.bunka.go.jp/kokugo_nihongo/sisaku/joho/joho/kijun/naikaku/pdf/joyokanjihyo_20101130.pdf
[3] https://att-astrec.nict.go.jp/member/mutiyama/ALT/

alphabet system (English), a mixed syllabic and logogram system (Japanese), and four abugida systems (Myanmar, Khmer, Thai, and Lao) are presented. Evidently, the four abugida systems with medium complexity share similar features, but differ from the alphabet/logogram systems. Generally, these medium complexity systems contain more commonly used symbols and a longer tail of obscure characters compared with general alphabet systems. However, their complexity is still far from that of a logogram system.

For scripts supported by the *Unicode Basic Multilingual Plane*, the following are taken as the objects of `AKKHARA`, where the number of points assigned by Unicode is provided in parentheses. Most scripts are abugidas; herein the dagger symbol indicates that the script is syllabic.

Devanagari (128), Bengali (96), Gurmukhi (80), Gujarati (91), Oriya (91), Tamil (72), Telugu (98), Kannada (89), Malayalam (118), Sinhala (91), Thai (87), Lao (82), Tibetan (211), Myanmar (160), Ethiopic (358), Cherokee† (92), Unified Canadian Aboriginal Syllabics[4] *(640), Khmer (114), Limbu (68), New Tai Lue (83), Tai Tham (127), Balinese (121), Sundanese (66), Batak (56), Lepcha (74), Vai† (300), Bamum† (88), Saurashtra (82), Javanese (91), Cham (83), Tai Viet (72),* and *Meetei Mayek (56).*

Herein, systems with lesser complexity, such as *Tai Le (35)* were excluded. For brevity, only the names of the scripts and number of assigned points are listed. One script may be used in different writing systems,[5] wherein only a portion of the symbols is used, or extended symbols are required.

3 Related Work and Applications

Popular operating systems (OS) such as Microsoft Windows and Macintosh provide keyboard layouts for various writing systems. Several keyboard layout editors, such as `Keyman`,[6] have been developed for more sophisticated layout editions. These input editors focus on key-character mapping with extensions for handling contextually dependent keystrokes.

For writing systems using Chinese characters, the `RIME`[7] input method engine has customization functions that allow users to design input schemes. This project supports obscure ancient fonts and dialect-based input methods. Regarding the medium complexity scripts of interest in this study, the `RIME` configuration is excessively complex.

[4] Essentially Abugida but encoded as syllabary.
[5] For example, the Devanagari is used by the languages of Hindi, Marathi, Sindhi, Nepali, and Sanskrit.
[6] `https://keyman.com/`
[7] `https://github.com/rime`

Ding et al. (2019) proposed an input method for Myanmar script that can be formulated using a complex automaton. Rather than key-character mapping, the basic idea concerns converting strings such that operations, such as looped alternations, substitutions for combined glyphs, and normalizations for ambiguities and redundancies, in Unicode can be modeled in a uniform framework. However, this method is applicable only to the particular writing system, and the overall automaton is hard-coded. `AKKHARA` follows the basic idea of the specific input method and extends it to a language-independent input editor.

4 Functions of `AKKHARA`

4.1 Interface

`AKKHARA` can be installed and used as a common input editor for `Windows`.[8] It supports multiple input methods that can be selected from a list. Figure 2 shows the launch screen of `AKKHARA`, and Fig. 3 shows an image of typed and converted strings. For a selected input method, the details can be customized and viewed in the interface, as shown in Fig. 4.

4.2 Format

In the background, an input method is defined using a text file named `*.akkhara`, where the symbol `*` denotes the name of the input method. Each line of the file is organized in the following format.

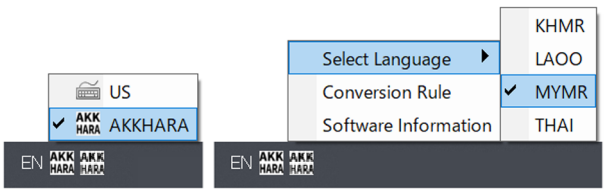

Fig. 2 Interface launching `AKKHARA`. Four scripts are listed: Khmer, Laos, Myanmar (selected), and Thai.

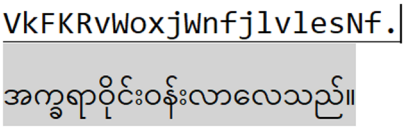

Fig. 3 Input interface. Pressing the `space` bar inputs the bottom converted Myanmar script. Pressing the `enter` key inputs the top original string.

[8] `https://att-astrec.nict.go.jp/member/ding/my-akkhara.html`

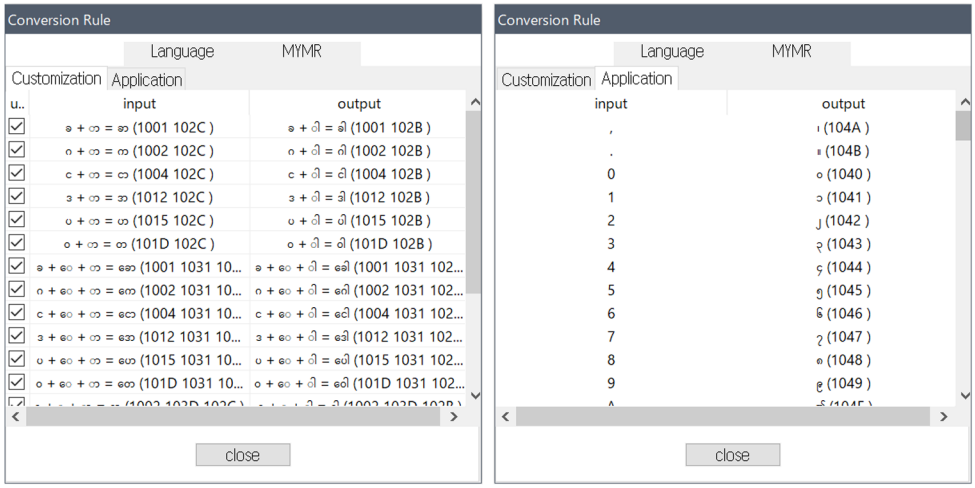

Fig. 4 Interface showing details of the input method selected. The left screen shows a customization wherein specific rules are enabled/disabled; the right screen lists all of rules enabled.

$$\texttt{input} \;|||\; \texttt{output} \;|||\; \texttt{priority} \;|||\; \texttt{status}, \tag{1}$$

where, `input` and `output` are the Unicode strings before and after a rewriting operation, respectively. `priority` is a non-negative integer, and rules with smaller values are always applied before rules with larger values. These rules are for local operations and applied ad-hoc after each keystroke. `priority` can also take a value of -1 for a global final normalization after all other operations have been conducted. `status` is assigned a value of `0`, `1`, or -1. The values of `1` and -1 are flags for the rules enabled and disabled by customization, respectively, and `0` denotes rules that are always applied.

The rewriting operations can actually describe a regular language because they always rewrite one side of a string with the input processing.[9] Therefore, `AKKHARA` can realize any input method as long as the required conversions can be formulated by a regular language. Note that `priority` is unnecessary if the rules are deliberately organized and strictly operated in a predefined order. `priority` can help users group rules for editing and maintenance. Rules that do not require explicit orders can be assigned with identical `priority`.

An `akkhara` file can be created and edited by a general text editor, such as `Notepad` on

[9] Rules with a `priority` of `-1` conduct an overall rewriting on a string, which is a more powerful operation than a regular language. This is provided only to handle certain intrinsic encoding problems in Unicode, which should not bother users.

`Windows`. For the `input` and `output` fields in a rule, ASCII characters[10] can be used directly; other characters should be represented using hexadecimal Unicode. The rules should be integrated by the increasing values of `priority` from `0`, ending with those with a `priority` of -1. A reboot is required to reload an `akkhara` file after modification.

The following subsections provide general descriptions of editing the rules for realizing alternations, substitutions, and normalizations of character strings. The descriptions omit the `status` of the rules as this depends on user preference.

4.3 Simple Mapping

For basic key-character mapping, simple rules can be described, as follows.

$$\mathtt{a_1} \;|||\; \mathtt{u_1} \;\cdots\; \mathtt{u_n} \;|||\; \mathtt{0} \;|||, \tag{2}$$

where $\mathtt{a_1}$ is an ASCII character available on a general US keyboard; and $\mathtt{u_1} \;\cdots\; \mathtt{u_n}$ is a Unicode string composed of one or more characters to be input by the keystroke of $\mathtt{a_1}$.

For these basic mappings that directly use ASCII characters, `priority` is generally set to 0, because no other prior rules are applied.

4.4 Simple Conversion

Rules (3)–(5) provide a typical example for realizing a simple conversion function. Using two keystrokes, $\mathtt{a_1}$ and $\mathtt{a_2}$, $\mathtt{u_3}$ can be input. Note that the keystrokes of $\mathtt{a_1}$ and $\mathtt{a_2}$ have their own input functions and that $\mathtt{u_3}$ is generated by a rule with a `priority` of `1`. This guarantees that Rule (5) executes after Rule (4).

$$\mathtt{a_1} \;|||\; \mathtt{u_1} \;|||\; \mathtt{0} \;|||, \tag{3}$$

$$\mathtt{a_2} \;|||\; \mathtt{u_2} \;|||\; \mathtt{0} \;|||, \tag{4}$$

$$\mathtt{u_1}\;\mathtt{u_2} \;|||\; \mathtt{u_3} \;|||\; \mathtt{1} \;|||, \tag{5}$$

This type of rule is suitable for characters with stable digraph Latin transcriptions, such as using `h` for aspiration in many systems. In cases where a digraph may cause ambiguities or where users may intentionally disable the conversion, the following rules can be added as solutions.

$$\mathtt{a_0} \;|||\; \mathtt{200B} \;|||\; \mathtt{0} \;|||, \tag{6}$$

$$\mathtt{200B}\;\mathtt{u_2} \;|||\; \mathtt{u_2} \;|||\; \mathtt{-1} \;|||, \tag{7}$$

[10] ASCII characters here are letters and marks that can be directly input by keystrokes on a `US` keyboard.

Here, $\texttt{a}_0$ inputs an invisible separator by Rule (6), and the zero-width space of `200B` is deleted by Rule (7) as a final normalization with a `priority` of -1. Subsequently, a sequence of keystrokes $\texttt{a}_1\ \texttt{a}_0\ \texttt{a}_2$ forcibly inputs the sequence $\texttt{u}_1\ \texttt{u}_2$.

In practice, the conversion key $\texttt{a}_2$ and separator key $\texttt{a}_0$ can be customized based on the Romanization preference, frequency of ambiguities, and keystroke distributions.

4.5 Conversion by Multiple Keystrokes

As a natural extension of simple conversion, a conversion key can be used multiple times to convert characters through a series. Generally, Rules (3)–(5) can be further extended using the following rules. Using these rules, $\texttt{u}_{\texttt{k}+1}$ can be converted by one keystroke of $\texttt{a}_1$, followed by $k-1$ keystrokes of $\texttt{a}_2$.

$$\texttt{u}_3\ \texttt{u}_2\ |||\ \texttt{u}_4\ |||\ \texttt{1}\ |||, \tag{8}$$

$$\dots$$

$$\texttt{u}_\texttt{k}\ \texttt{u}_2\ |||\ \texttt{u}_{\texttt{k}+1}\ |||\ \texttt{1}\ |||, \tag{9}$$

$$\dots$$

Note that the `priority` values from Rule (8) can be set to 1. As the rewrite is implemented after each keystroke, the order of these rules does not need to be distinguished. When one is applied, the previous rewrite has already been completed.

Conversion using multiple keystrokes is suitable for a series of redundant characters for the same phoneme in a specific writing system, where only one or two of them are frequently used. Therefore, the inputs of remaining obscure ones can be grouped by the sequential conversion.

4.6 Looped Conversion

As another type of conversion, a single keystroke can be repeated to input a group of characters in a looped manner. Rules (10)–(12) realize a function in which a series of keystrokes $\texttt{a}_1$, $\texttt{u}_1$ and $\texttt{u}_2$ can be converted in a looped manner. That is, an odd number of keystroke(s) of $\texttt{a}_1$ can input $\texttt{u}_1$, and even number for $\texttt{u}_2$.

$$\texttt{a}_1\ |||\ \texttt{u}_1\ |||\ \texttt{0}\ |||, \tag{10}$$

$$\texttt{u}_1\ \texttt{u}_1\ |||\ \texttt{u}_2\ |||\ \texttt{1}\ |||, \tag{11}$$

$$\texttt{u}_2\ \texttt{u}_1\ |||\ \texttt{u}_1\ |||\ \texttt{1}\ |||, \tag{12}$$

Similarly, by replacing Rule (12) with the following rules, looping can be extended to an

arbitrary number of characters.

$$\mathtt{u_2\ u_1\ |||\ u_3\ |||\ 1\ |||}, \tag{13}$$

$$\ldots$$

$$\mathtt{u_{k-1}\ u_1\ |||\ u_k\ |||\ 1\ |||}, \tag{14}$$

$$\mathtt{u_k\ u_1\ |||\ u_1\ |||\ 1\ |||}, \tag{15}$$

This type of looped conversion can be used for a set of symbols (mostly two or three in practice) that share identical Romanization by one letter, such that they can be quickly converted using identical keystrokes.

4.7 Normalization of Variants

In some writing systems, context-dependent variants of specific symbols are used. If $\mathtt{u_1}$ and $\mathtt{u_1'}$ are two variants of one character, while encoded separately by Unicode, then the following rules realize an example of context-dependent conversions.

$$\mathtt{u_2\ u_1\ |||\ u_2\ u_1'|||\ 2\ |||}, \tag{16}$$

$$\mathtt{u_3\ u_2\ u_1'\ |||\ u_3\ u_2\ u_1\ |||\ 3\ |||}, \tag{17}$$

By Rule (16), when $\mathtt{u_1}$ comes after $\mathtt{u_2}$, it is substituted by $\mathtt{u_1'}$. When $\mathtt{u_1}$ comes after a string of $\mathtt{u_3\ u_2}$, it is kept as it is, which is further supported by Rule (17). Note that the `priority` of such normalization rules begins from `2` to distinguish them from general input and conversion operations. The longer the context, the larger the `priority` assigned. In the most extreme case, certain obscure variants may appear only in a few specific words. For such rules, a large constant of `priority` can be set for convenience.

5 Case Study

5.1 General Issues and a Myanmar Input Method

In general, the following issues are required to design a minimum scheme to input a writing system with medium complexity.

- The arrangement of some common characters. These characters should be input by simple keystrokes.
- The arrangement of other characters that require alternation keys. The alternation key is primarily the `shift`-key, or a second alternation key is required if characters remain.

The original work of Ding et al. (2019) focused primarily on these two issues, thus providing a basic interface for inputting the specific set of characters. Their work extended the use of alternation key(s) for more efficient conversion operations. The instruction of the input method can be printed by users on `A4` paper (Fig. 5, i.e., Fig. 4 in the original paper). The conventionally accepted Romanization, frequency of characters, and keystroke distribution on a `QWERTY` keyboard are considered in the design for efficiency.

In brief, the Myanmar Language Commission (MLC) Transcription System (Department of the Myanmar Language Commission 2014) was applied. Because the MLC system is rigid and academic, it is simplified for input purposes. For example, aspirated consonants are denoted by a preceding `h` in the MLC system such as `hk` or `hp`, for historical reasons. This is adjusted to succeeding `h`, such as `kh` and `ph` to input the corresponding letters. Compared with the MLC system, this design is based on general use of Latin letters to address the conversion function using `h` in the aforementioned example. In addition, vowel systems are simplified such that more than one letters with similar phonetic values are arranged on one key, and obscure letters can be input using multiple keystrokes of the same key. For other diacritic marks, those frequently used are arranged in the middle area of the keyboard (e.g., on the `f` and `j` keys) for easy keystrokes. More details about the design, as well as about the keystroke distribution compared with traditional keyboard layout, can be found in the original paper (Ding et al. 2019).

For further convenience, an input method should consider orthographic and encoding issues because:

- Some redundant characters are context-dependent. In some cases, they are encoded as one character by Unicode and the variants are accommodated by font design. In other cases, they are treated as different characters that require manual selection.
- The combination of some complex glyphs can be realized in more than one manner; the order of component characters is not strictly defined. Similarly, specific fonts may be sensitive or insensitive to diversity.

These issues require detailed normalization operations, which require users to have a good understanding of the encoding and fonts when everything is performed manually. The original Myanmar input method was formulated using a finite-state automaton (Hopcroft et al. 2013) and hard-coded to satisfy the minimum input requirements. More sophisticated operations are not explicitly mentioned as these functions are, although possible, difficult to realize using a vanilla automaton. Therefore, the original implementation is more illustrative than practical.

The following subsection discusses the realization of the input method using a generalized `AKKHARA` with basic extended normalization operations.

Fig. 5 Each cell provides the Unicode, Myanmar character, and input manner top to bottom. For Myanmar characters with more than one input method, a vertical bar is used to separate the different input manners.

5.2 Realization Using AKKHARA

An `akkhara` file that realizes the aforementioned input method was released with the `AKKHARA` binary files. The Appendix lists rules organized in Tables 2, 3, 4, 5, and 6, according to their `priority` of `0`, `1`, `2`, `3`, and `−1`, respectively. The leading number is the line number in the released file.

Table 2 includes the basic mapping described in Sec. 4.3 for two punctuation marks, 10 digits,[11] and 26 lower/uppercase Latin letters. The three letters `a`, `o`, and `x` have a one-to-many mapping. Table 3 lists further conversions. The table is divided into three blocks from top to bottom and from left to right. The first block (lines 65–95) contains simple conversions (mainly by `103E` and `1002`, and by `103B` and `103C`), as described in Sec. 4.4; the second block (lines 96–105) contains looped conversions for five pairs (`102D`/`102E`, `102F`/`1030`, `1031`/`1032`, `1038`/`1037`, and `103A`/`1039`), as described in Sec. 4.6; and the third block (lines 106–110) contains five conversions implemented using multiple keystrokes, as described in Sec. 4.5. Tables 2 and 3 cover the scheme in Fig. 5, which realizes the minimum requirements for basic character conversion.

From Table 4, the normalization functions in Sec. 4.7 are applied. Tables 4 and 5 concern the normalization of the variants of `102C`/`102B`. Character sequences that require alternation are listed. Some rules, such as lines 125 and 126, are prepared for obscure cases that require a context of up to three characters to decide the variants. This is inconveniently to be represented by an automaton, where the status of the tri-grams should be prepared.

Table 6 lists the final normalization rules. Sec. 4.4 mentions the upper two rules related to `200B` in terms of cleaning the invisible separator to avoid ambiguities. The lower three rules are concern the Unicode scheme for the recommended encoding. An issue related to the final normalization is the use of the `backspace` key for deletion while inputting. Because the appending and converting operations are uniform in `AKKHARA`, a `backspace` keystroke actually cancels the previous rule application, regardless of the specific operation. Figure 6 shows examples of canceling appending and converting operations. To hide such encoding details from users, normalization substitutions are conducted after all other conversions are performed. This operation exceeds the capability of an automaton, but is a general rewriting regardless of the previous status.

These 133 rules realize a basic Myanmar input method that can generate well formed Unicode character sequences. Users can add more customized rules. As mentioned, `priority` is convenient to group rules, and more sophisticated context dependent conversions can be assigned a `priority` from `4`. These black-boxes are the core functions of the input method, and all user-added rules

[11] Ding et al. (2019) did not include these.

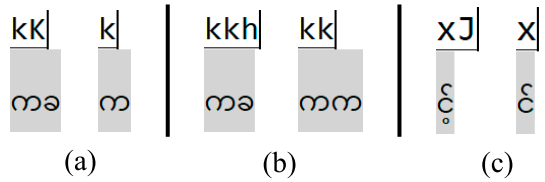

Fig. 6 (a) Using `k` and `K` to input `1000` and `1001`, with a `backspace` keystroke deleting character `1001`. (b) Using `k` and `kh` to input the two characters and a `backspace` keystroke to cancel the conversion caused by `h`. (c) During the input process, `1037` (the lower circle, which is input by an upper case `J`) and `103A` (the upper arc, which is the end of the string input by `x`) are not normalized (by line 133 in Table 6) such that a `backspace` keystroke deletes `1037`.

are applied after the basic operations.

6 Conclusion and Future Work

This paper has described the `AKKHARA`, a general input method editor that can accommodate writing systems of medium complexity. A realization of a Romanization-based Myanmar input method is illustrated as an application case for `AKKHARA`. `AKKHARA` is more powerful than a key-character mapper, but simpler than the input methods for Chinese and Japanese. Therefore, `AKKHARA` is easy to maintain and suitable for realizing customized input methods.

Future engineering developments include: 1) developing `AKKHARA` for other OSs in addtion to `Windows`, 2) improving the rule editing interface, and 3) integrating corpora for prediction functions. The input methods of more abugida and syllablic systems will be further investigated and edited using `AKKHARA` for efficient input and normalized digitization.

References

Department of the Myanmar Language Commission (2014). *Myanmar-English Dictionary (Myanma-anggalip abidan)* (12th edition). Ministry of Education, the Republic of the Union of Myanmar.

Ding, C., Utiyama, M., and Sumita, E. (2018). "Simplified Abugidas." In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 491–495, Melbourne, Australia. Association for Computational Linguistics.

Ding, C., Utiyama, M., and Sumita, E. (2019). "MY-AKKHARA: A Romanization-based

Burmese (Myanmar) Input Method." In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pp. 157–162, Hong Kong, China. Association for Computational Linguistics.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2013). *Introduction to Automata Theory, Languages, and Computation* (3rd edition). Pearson.

Maddieson, I. (2013a). "Consonant Inventories." In Dryer, M. S. and Haspelmath, M. (Eds.), *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Maddieson, I. (2013b). "Vowel Quality Inventories." In Dryer, M. S. and Haspelmath, M. (Eds.), *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Riza, H., Purwoadi, M., Uliniansyah, T., Ti, A. A., Aljunied, S. M., Mai, L. C., Thang, V. T., Thai, N. P., Sun, R., Chea, V., Khin Mar Soe, Khin Thandar Nwet, Utiyama, M., and Ding, C. (2016). "Introduction of the asian language treebank." In *Proc. of O-COCOSDA*, pp. 1–6.

Appendix

1	, ||| 104A ||| 0 ||| 0	33	U ||| 1030 ||| 0 ||| 0
2	. ||| 104B ||| 0 ||| 0	34	V ||| 1021 ||| 0 ||| 0
3	0 ||| 1040 ||| 0 ||| 0	35	W ||| 101D ||| 0 ||| 0
4	1 ||| 1041 ||| 0 ||| 0	36	X ||| 1004 ||| 0 ||| 0
5	2 ||| 1042 ||| 0 ||| 0	37	Y ||| 101A ||| 0 ||| 0
6	3 ||| 1043 ||| 0 ||| 0	38	Z ||| 1008 ||| 0 ||| 0
7	4 ||| 1044 ||| 0 ||| 0	39	a ||| 1031 102C ||| 0 ||| 0
8	5 ||| 1045 ||| 0 ||| 0	40	b ||| 1017 ||| 0 ||| 0
9	6 ||| 1046 ||| 0 ||| 0	41	c ||| 1005 ||| 0 ||| 0
10	7 ||| 1047 ||| 0 ||| 0	42	d ||| 1012 ||| 0 ||| 0
11	8 ||| 1048 ||| 0 ||| 0	43	e ||| 1031 ||| 0 ||| 0
12	9 ||| 1049 ||| 0 ||| 0	44	f ||| 103A ||| 0 ||| 0
13	A ||| 104F ||| 0 ||| 0	45	g ||| 1002 ||| 0 ||| 0
14	B ||| 1018 ||| 0 ||| 0	46	h ||| 103E ||| 0 ||| 0
15	C ||| 1006 ||| 0 ||| 0	47	i ||| 102D ||| 0 ||| 0
16	D ||| 1013 ||| 0 ||| 0	48	j ||| 1038 ||| 0 ||| 0
17	E ||| 1032 ||| 0 ||| 0	49	k ||| 1000 ||| 0 ||| 0
18	F ||| 1039 ||| 0 ||| 0	50	l ||| 101C ||| 0 ||| 0
19	G ||| 1003 ||| 0 ||| 0	51	m ||| 1019 ||| 0 ||| 0
20	H ||| 101F ||| 0 ||| 0	52	n ||| 1014 ||| 0 ||| 0
21	I ||| 102E ||| 0 ||| 0	53	o ||| 102D 102F ||| 0 ||| 0
22	J ||| 1037 ||| 0 ||| 0	54	p ||| 1015 ||| 0 ||| 0
23	K ||| 1001 ||| 0 ||| 0	55	q ||| 200B ||| 0 ||| 0
24	L ||| 1020 ||| 0 ||| 0	56	r ||| 103C ||| 0 ||| 0
25	M ||| 1036 ||| 0 ||| 0	57	s ||| 101E ||| 0 ||| 0
26	N ||| 100A ||| 0 ||| 0	58	t ||| 1010 ||| 0 ||| 0
27	O ||| 1029 ||| 0 ||| 0	59	u ||| 102F ||| 0 ||| 0
28	P ||| 1016 ||| 0 ||| 0	60	v ||| 102C ||| 0 ||| 0
29	Q ||| 1002 ||| 0 ||| 0	61	w ||| 103D ||| 0 ||| 0
30	R ||| 101B ||| 0 ||| 0	62	x ||| 1004 103A ||| 0 ||| 0
31	S ||| 1026 ||| 0 ||| 0	63	y ||| 103B ||| 0 ||| 0
32	T ||| 1011 ||| 0 ||| 0	64	z ||| 1007 ||| 0 ||| 0

Table 2 Sixty-four rules with a `priority` of `0`.

65	`1000 103E ||| 1001 ||| 1 ||| 0`
66	`1002 103E ||| 1003 ||| 1 ||| 0`
67	`1005 103E ||| 1006 ||| 1 ||| 0`
68	`1007 103E ||| 1008 ||| 1 ||| 0`
69	`1010 103E ||| 1011 ||| 1 ||| 0`
70	`1012 103E ||| 1013 ||| 1 ||| 0`
71	`1015 103E ||| 1016 ||| 1 ||| 0`
72	`1017 103E ||| 1018 ||| 1 ||| 0`
73	`1004 1002 ||| 100F ||| 1 ||| 0`
74	`100A 1002 ||| 1009 ||| 1 ||| 0`
75	`100F 1002 ||| 104C ||| 1 ||| 0`
76	`1010 1002 ||| 100B ||| 1 ||| 0`
77	`1011 1002 ||| 100C ||| 1 ||| 0`
78	`1012 1002 ||| 100D ||| 1 ||| 0`
79	`1013 1002 ||| 100E ||| 1 ||| 0`
80	`1014 1002 ||| 1004 ||| 1 ||| 0`
81	`1019 1002 ||| 1036 ||| 1 ||| 0`
82	`101B 1002 ||| 104D ||| 1 ||| 0`
83	`101C 1002 ||| 1020 ||| 1 ||| 0`
84	`101E 1002 ||| 103F ||| 1 ||| 0`
85	`1020 1002 ||| 104E ||| 1 ||| 0`
86	`1027 1002 ||| 104F ||| 1 ||| 0`
87	`1029 1002 ||| 102A ||| 1 ||| 0`
88	`102C 1002 ||| 102B ||| 1 ||| 0`
89	`102D 1002 ||| 1023 ||| 1 ||| 0`
90	`102E 1002 ||| 1024 ||| 1 ||| 0`
91	`102F 1002 ||| 1025 ||| 1 ||| 0`
92	`1030 1002 ||| 1026 ||| 1 ||| 0`
93	`1031 1002 ||| 1027 ||| 1 ||| 0`
94	`1014 103B ||| 100A ||| 1 ||| 0`
95	`101E 103C ||| 1029 ||| 1 ||| 0`
96	`102D 102D ||| 102E ||| 1 ||| 0`
97	`102E 102D ||| 102D ||| 1 ||| 0`
98	`102F 102F ||| 1030 ||| 1 ||| 0`
99	`1030 102F ||| 102F ||| 1 ||| 0`
100	`1031 1031 ||| 1032 ||| 1 ||| 0`
101	`1032 1031 ||| 1031 ||| 1 ||| 0`
102	`1038 1038 ||| 1037 ||| 1 ||| 0`
103	`1037 1038 ||| 1038 ||| 1 ||| 0`
104	`103A 103A ||| 1039 ||| 1 ||| 0`
105	`1039 103A ||| 103A ||| 1 ||| 0`
106	`102C 102C ||| 1021 ||| 1 ||| 0`
107	`103B 103B ||| 101A ||| 1 ||| 0`
108	`103C 103C ||| 101B ||| 1 ||| 0`
109	`103D 103D ||| 101D ||| 1 ||| 0`
110	`103E 103E ||| 101F ||| 1 ||| 0`

Table 3 Forty-six rules with a `priority` of `1`.

111	`1001 102C ||| 1001 102B ||| 2 ||| 1`
112	`1002 102C ||| 1002 102B ||| 2 ||| 1`
113	`1004 102C ||| 1004 102B ||| 2 ||| 1`
114	`1012 102C ||| 1012 102B ||| 2 ||| 1`
115	`1015 102C ||| 1015 102B ||| 2 ||| 1`
116	`101D 102C ||| 101D 102B ||| 2 ||| 1`
117	`1001 1031 102C ||| 1001 1031 102B ||| 2 ||| 1`
118	`1002 1031 102C ||| 1002 1031 102B ||| 2 ||| 1`
119	`1004 1031 102C ||| 1004 1031 102B ||| 2 ||| 1`
120	`1012 1031 102C ||| 1012 1031 102B ||| 2 ||| 1`
121	`1015 1031 102C ||| 1015 1031 102B ||| 2 ||| 1`
122	`101D 1031 102C ||| 101D 1031 102B ||| 2 ||| 1`
123	`1002 103D 102C ||| 1002 103D 102B ||| 2 ||| 1`
124	`1012 103D 102C ||| 1012 103D 102B ||| 2 ||| 1`
125	`1002 103D 1031 102C ||| 1002 103D 1031 102B ||| 2 ||| 1`
126	`1012 103D 1031 102C ||| 1012 103D 1031 102B ||| 2 ||| 1`

Table 4 Sixteen rules with a `priority` of `2`.

127	1015 1039 1015 102B ||| 1015 1039 1015 102C ||| 3 ||| 1
128	1015 1039 1015 1031 102B ||| 1015 1039 1015 1031 102C ||| 3 ||| 1

Table 5 Two rules with a `priority` of 3.

129	200B 103E ||| 103E ||| -1 ||| 0
130	200B 1002 ||| 1002 ||| -1 ||| 0
131	1025 102E ||| 1026 ||| -1 ||| 0
132	1029 1031 102C 103A ||| 102A ||| -1 ||| 0
133	103A 1037 ||| 1037 103A ||| -1 ||| 0

Table 6 Five rules with a `priority` of -1.

Chenchen Ding: received an M.E. degree in computer science from the University of Tsukuba, Japan, in 2012, and a Ph.D. degree in engineering from the University of Tsukuba, Japan, in 2015. He is currently a senior researcher with the National Institute of Information and Communications Technology, Japan. His research interests include computational linguistics and natural language processing.

Masao Utiyama: received a B.S. in Computer Science from the University of Tsukuba, Japan, in 1992, an M.S. in Computer Science from the University of Tsukuba, Japan, in 1994, a Ph.D. in Engineering from the University of Tsukuba, Japan, in 1997. From 1997 to 1999, he was a Research Associate at Shinshu University, Japan. He has been a member of the National Institute of Information and Communications Technology, Japan, since 1999, and is currently an Executive Researcher at NICT.

Eiichiro Sumita: received his Ph.D. in Engineering from Kyoto University in 1999, and a master's and bachelor's in Computer Science from the University of Electro-Communications in 1982 and 1980, respectively. He is now in the National Institute of Information and Communication Technology (NICT), its fellow and the associate director-general of Advanced Speech Translation Research and Development Promotion Center (ASTREC). His research interests cover machine translation and e-learning. He is a co-recipient of the Maejima Hisoka Prize in 2013, the Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology, Prizes for Science and Technology in 2010, and the AAMT Nagao Award in 2007.

(Received April 25, 2022)
(Revised July 22, 2022)
(Accepted September 1, 2022)