AKKHARA: A Generalized Rewrite-based Input Method Editor for Writing Systems with Medium Complexity

Chenchen Ding, Masao Utiyama, Eiichiro Sumita
Advanced Translation Technology Laboratory,
Advanced Speech Translation Research and Development Promotion Center,
National Institute of Information and Communications Technology
3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0289, Japan
{chenchen.ding, mutiyama, eiichiro.sumita}@nict.go.jp

Abstract

This paper describes the input method editor AKKHARA, which has been developed to accommodate writing systems comprising several tens to hundreds of symbols. In particular, it is designed as a general solution for the efficient input of various abugida writing systems used in South and Southeast Asia. As an engineering realization, AKKHARA accepts and applies a set of rewrite rules with priorities, so that alternation, substitution, and normalization of character strings are applied alongside the keystrokes. Compared with key-character editors, AKKHARA provides greater flexibility for Romanization-based rule edition. Compared with input methods developed for Chinese and Japanese, AKKHARA is lightweight and easy to maintain. A version of AKKHARA for Microsoft Windows has been released[1] that supports Unicode characters with customizable functions for rewrite rule edition.

1 Introduction

Various alphabet writing systems are used to record a large number of languages around the world. These alphabet systems use tens of distinguishable symbols to transcribe phonemes.[2] At the other extreme, languages such as Chinese and Japanese apply logogram systems totally or partially in their orthographies, whereby thousands of symbols are used to represent morphemes.

This work focuses on a group of syllabic systems that lies somewhere between the abovementioned extremes. They are referred to as having *medium complexity* in this paper. Compared with logogram systems, they are still phonetic-based, although a certain redundancy in spelling may be retained to distinguish morphemes; compared with an alphabet system, they assign symbols based on syllables rather than on phonemes, and may apply nonlinear combination schemes to basic/diacritic characters.

Traditionally, these syllabic systems are input by direct keystrokes treated as common alphabets. Because of the larger symbol set, the keyboard layout is crowded and one or more alternation keys are required. However, these systems are not as complex as those used in Chinese or Japanese, where language-dependent input methods are indispensable. The AKKHARA system developed in this work provides a general solution for the input of writing systems having medium complexity.

AKKHARA repeatedly applies rewrite operations to convert keystrokes into specific character strings. Therefore, compared with a keyboard layout editor, AKKHARA has more capacity and flexibility in applying string operations than simply appending new characters by keystrokes. This is especially important for writing systems having certain redundancies in their encoding systems, as some complex combined glyphs can be realized in more than one way. Thus, rewrite operations can normalize the input to generate consistently encoded textual data.

The temporary input methods for Chinese and Japanese are generally based on common Romanization systems such as *Pinyin* and *Rōmaji*. However, in many languages that do not use the Latin alphabet, there is no consistent Romanization system. In AKKHARA, the set of rewrite rules is organized in an editable textual file with a structured format, enabling users to customize the input method according to their preferences.

In Sec. 2, the linguistic background of writing systems is introduced. Related work and applications are reviewed in Sec. 3. Section 4 provides an overview of AKKHARA and Sec. 5 describes the rewrite rule edition. Section 6 provides a real example of defining a minimal input method for the Myanmar script with 133 rules. Section 7 summarizes the conclusions to this study and presents ideas for future work.

[1] https://www2.nict.go.jp/astrec-att/member/ding/my-akkhara.html

[2] Of course, the orthography is affected by many historical factors.

2 Linguistic Background

Table 1 lists the main types of writing systems used in the world, with a comparison of the number of symbols applied. Generally, an analytic phonetic writing system, e.g., a typical alphabet or abjad system, usually contains around 30 ($\approx 10^{1.5}$) symbols. This is because the average phonetic inventory among natural languages contains 22 ± 3 consonants (Maddieson, 2013a) and five or six vowels (Maddieson, 2013b). At the other extreme, a logogram system may contain thousands of symbols for daily use. For example, the *Table of General Standard Chinese Characters*[3] contains $8,105$ characters for Chinese, and the *List of jōyō kanji*[4] contains 2,136 characters for Japanese.

Figure 1 compares the character distribution across different writing systems. The parallel data from the Asian Language Treebank (Riza et al., 2016)[5] was used. The differences between an alphabet system (English), a mixed syllabic and logogram system (Japanese), and four abugida systems (Myanmar, Khmer, Thai, and Lao) are presented. Clearly, the four abugida systems with medium complexity share similar features, but differ from the alphabet/logogram systems. Generally, in these medium complexity systems, there are more commonly used symbols and a longer tail of obscure characters than in a general alphabet system. However, the complexity is still far from that of a logogram system.

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. The abugida/syllabic systems are referred to as having *medium complexity* in this paper.

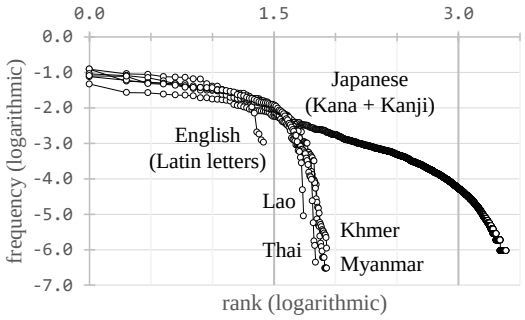

Figure 1: Character distribution on Latin letters in English, the *kana* and *kanji* in Japanese, and the four abugida systems of Myanmar, Khmer, Thai, and Lao.

3 Related Work and Applications

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

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

Ding et al. (2019) proposed an input method for the Myanmar script that can be formulated by a complex automaton. Rather than key-character mapping, the basic idea is the conversion of strings so that operations such as looped alternation, substitution for combined glyphs, and normalization for ambiguities and redundancies in the Unicode can be modeled in a uniform framework. However, this method is only applicable to the specific writing system and the overall automaton is hard-coded. Essentially, `AKKHARA` is a generalized extension of this previous approach.

Prediction based on words, phrases, or even sentences supported by large-scale data is a common function provided by many Chinese and Japanese input methods. Ding et al. (2018) investigated an auto-complete method for several abugida systems that can achieve high accuracy, even with limited training data. This is a direction for the future development of `AKKHARA` because it provides a more extensible platform than a key-character mapper.

[3] http://www.gov.cn/gzdt/att/att/site1/20130819/tygfhzb.pdf

[4] https://www.bunka.go.jp/kokugo_nihongo/sisaku/joho/joho/kijun/naikaku/pdf/joyokanjihyo_20101130.pdf

[5] https://www2.nict.go.jp/astrec-att/member/mutiyama/ALT/

[6] https://keyman.com/

[7] https://github.com/rime

4 Overview of AKKHARA

AKKHARA can be installed and used as a common input method editor under Win. It supports multiple input methods, which can be selected from a list. Figure 2 shows the launch screen of AKKHARA, while Fig. 3 shows an image of the typed and converted strings. For a selected input method, the details can be customized and viewed in the interface shown in Fig. 4.

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

`input` ||| `output` ||| `priority` ||| `status` (1)

Here, `input` and `output` are Unicode strings before and after a rewriting operation. The `priority` is generally a non-negative integer, and rules with smaller values are always applied before rules with larger values. The `priority` can also take a value of -1 for final normalization after all other operations have been conducted. The `status` takes a value of 0, 1, or -1. Values of 1 and -1 are flags for the rules that have been enabled and disabled by the customization, and 0 denotes the basic key-character mappings that are always applied. Therefore, the `customization` label (left of Fig. 4) lists all the non-zero flagged rules with their status, and the `application` label (right of Fig. 4) lists all the non-negative flagged rules that are used.

As a generalization of the hard-coded automaton in Ding et al. (2019), a set of rules in the abovementioned format actually defines the state transitions in an automaton by the `input` and `output` fields.[8] The table here provides an example of the state-transition from parts of Figs. 8 and 9 in the appendix of Ding et al. (2019). Thus, this table can be described by the following rules: `r` ||| `3C` , `g` ||| `02` , `3C r` ||| `1B` , and `1B g` ||| `4D` . The following section provides a brief overview of rule edition.

	r	g
_	3C	02
3C	1B	
1B		4D

5 Rule Edition

5.1 General Issues

An `akkhara` file can be created and edited by a general text editor such as `Notepad` under `Win`.

[8] Strictly, `priorities` are unnecessary if the rules are deliberately edited. As described later, `priorities` can group rules that are useful for edition.

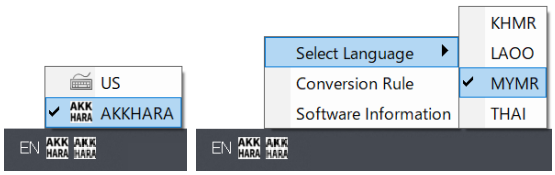

Figure 2: Interface to launch AKKHARA. Four scripts are listed: Khmer, Laos, Myanmar (selected), and Thai.

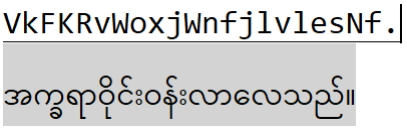

Figure 3: Input interface. Pressing the `Space` bar will input the lower converted Myanmar script. Pressing the `Enter` key will input the upper original string.

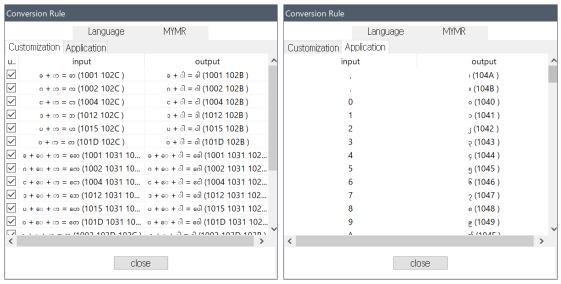

Figure 4: Interface showing details of the selected input method. The left screen shows a customization in which specific rules are enabled/disabled, while the right screen lists all of the enabled rules.

For the `input` and `output` files in a rule, ASCII characters[9] can be directly used; other characters should be represented by hexadecimal Unicode. The rules should be integrated on the basis of increasing values of `priority` from 0, ending with those that have a `priority` of -1. A reboot is required to reload an `akkhara` file after modified.

In the following subsections, typical rule patterns are described. The examples are taken from Ding et al. (2019). The `status` of the rules is omitted as this depends on user preference.

5.2 Simple Mapping

For basic key–character mappings, simple rules can be described in the following form.

`k` ||| `1000` ||| `0` ||| (2)

`h` ||| `103E` ||| `0` ||| (3)

`K` ||| `1001` ||| `0` ||| (4)

[9] AKKHARA use the US keyboard layout under Win by default. The ASCII characters mean those letters and marks can be input by direct keystrokes on such a keyboard.

One-to-many mappings can also be assigned.

```
x ||| 1004 103A ||| 0 |||                    (5)
```

For these basic mappings that directly use ASCII characters, the `priority` is set to 0, as no other prior rules will be applied.

5.3 Simple Conversion

Given Rules (2) – (4), adding Rule (6) allows the input of the character `1001` by two keystrokes, `k` and `h`, as well as by the uppercase `K`.

```
1000 103E ||| 1001 ||| 1 |||                 (6)
```

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

```
q ||| 200B ||| 0 |||                         (7)
200B 103E ||| 103E ||| -1 |||                (8)
```

Here, `q` is set as an invisible separator by Rule (7) and the zero width space `200B` is deleted by Rule (8) as a final normalization with a `priority` of -1. The keystrokes `k q h` then forcibly input the sequence `1000 103E`.[10] Section 5.7 discusses further details about the normalization processing.

5.4 Conversion by Multiple Keystrokes

In many abugida systems, symbols have dependent and independent forms. For example, the `103E` given by Rule (3) is a dependent form — the corresponding independent form is `101F`. By adding the following rules, a double stroke of `h` will input the independent form as well as an uppercase `H`.

```
H ||| 101F ||| 0 |||                         (9)
103E 103E ||| 101F ||| 1 |||                 (10)
```

This is similar to the case of Sec. 5.3, but the conversion is implemented by tapping an identical key. This is suitable for symbols with identical Latin transcriptions in which one of them is overwhelmingly used in the orthography.

[10] The sequence `1000 103E` violates the common orthography of Myanmar, which states that it should not appear in normal texts. It is only mentioned here as an example.

Combined with the case of Sec. 5.3, a key for conversion can be used multiple times to input obscure symbols, as shown by the following rules.

```
n ||| 1014 ||| 0 |||                        (11)
g ||| 1002 ||| 0 |||                        (12)
1014 1002 ||| 1004 ||| 1 |||                (13)
1004 1002 ||| 100F ||| 1 |||                (14)
100F 1002 ||| 104C ||| 1 |||                (15)
```

From Rules (11) – (15), a keystroke of `n` inputs `1014`, and then multiple keystrokes of `g` successively convert `1004`, `100F`, and `104C`. Note that the `priority` values of Rules (13) – (15) can be all set to 1. Because the rewrite is implemented after each keystroke, there is no need to distinguish the order of these rules. When one of them is applied, the previous rewrite has already been completed.

Similar to Rule (8), Rules (16) and (7) prevent a keystroke of `q` from being converted if the very character `1002` input by `g` is expected.

```
200B 1002 ||| 1002 ||| -1 |||                (16)
```

5.5 Looped Conversion

This kind of conversion is applied to groups of diacritic symbols. Here, the characters `102D` and `102E` in Myanmar script are taken as examples. The two diacritic symbols are generally Romanized as `i`, but with different tones.

```
i ||| 102D ||| 0 |||                         (17)
I ||| 102E ||| 0 |||                         (18)
102D 102D ||| 102E ||| 1 |||                 (19)
102E 102D ||| 102D ||| 1 |||                 (20)
```

Rules (17) – (20) input the two symbols by keystrokes of `i`, where `102D` is input by an odd number of times and `102E` is input by an even number of times. As a shortcut, `102E` can also be input by an uppercase `I`.

5.6 Normalization of Variants

Context-dependent variants of specific symbols are used in some writing systems.[11] The following rules provide a relatively complex example.

```
p ||| 1015 ||| 1 |||                        (21)
v ||| 102C ||| 1 |||                        (22)
1015 102C ||| 1015 102B ||| 2 |||           (23)
1015 1039 1015 102B |||
1015 1039 1015 102C ||| 3 |||                (24)
```

[11] Such as the abandoned Latin letter *long s*.

The characters 102B (*tall aa*) and 102C (*aa*) are variants of one diacritic mark. Typically, 102C is used by default, while the tall variant 102B is easier to read in specific combinations. Rule (23) alternates the tall variant after the character 1015 input by p. However, in the geminated case whereby two 1015 inputs are stacked by the operator character 1039, the default variant will be used, which is realized by Rule (24). Consequently, the `priority` of Rule (24) must be greater than that of Rule (23).[12]

In orthographies that preserve historical features, certain obscure variants may only appear in a few specific words, and these can be organized in the rules. As in Rules (23) and (24), specific conversions requiring longer contexts should have a larger `priority` value than those general ones requiring less contextual information.

5.7 Normalization of Encoding

Rules (8) and (16) have been introduced in previous subsections to avoid potential ambiguities. More generally, rules with a `priority` of -1 are designed for some non-intuitive combinations/orders in the encoding system (i.e., Unicode here). Unlike Sec. 5.6, the normalization in this subsection may be unnoticed by users.

A crucial case is to use the `Backspace` key for deletion during inputting. Because the appending and converting operations are uniform in AKKHARA, a keystroke of `Backspace` actually cancels the previous rule application, regardless of the specific operation. Figures 5 (a) and 5 (b) provide examples on the deletion of appending and converting operations, respectively. As an example of this kind of normalization, the order of Myanmar characters 103A and 1037 is not strictly determined. They are usually normalized into 1037 103A by an increasing order of the code, although intuitively, 103A is considered to be part of the preceding character. By Rule (5) and the following Rule (25), the input can be realized as in Fig. 5 (c), which is in accord with our intuition. Rule (26) performs the final normalization when escaping the input mode.

$$\texttt{J} \;|||\; \texttt{1037} \;|||\; \texttt{0} \;|||\; \tag{25}$$

$$\texttt{103A 1037} \;|||\; \texttt{1037 103A} \;|||\; -\texttt{1} \;|||\; \tag{26}$$

[12]For clarity, the `priority` of Rule (23) is set to 2 here to emphasize that it will be conducted after all common rewrites.

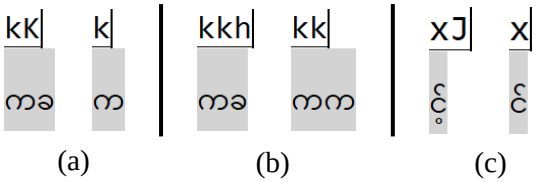

Figure 5: (a) Using k and K to input 1000 and 1001, with a `Backspace` keystroke deleting the character 1001. (b) Using k and kh to input two characters and the `Backspace` keystroke to cancel the conversion caused by h. (c) During the input process, 1037 (the lower circle) and 103A (the upper arc) are not normalized, so that a `Backspace` keystroke deletes the 1037.

6 Example

In this section, we provide an example of an `akkhara` file that realizes the function described by Ding et al. (2019). For a clear presentation, the rules are organized in Tables 2, 3, 4, 5, and 6, according to their `priority` of 0, 1, 2, 3, and -1, respectively. By organizing all the rules in the format of Rule (1) and placing them sequentially line-by-line in an `akkhara` file, the input method is defined and facilitated by AKKHARA.

Table 2 includes the basic mapping described in Sec. 5.2 for two punctuation marks, 10 digits,[13] and the 26 lower/uppercase Latin letters. The three letters a, o, and x have a one-to-many mapping.

Table 3 includes further conversions. The table is divided into three blocks from top to bottom and from left to right. The first block contains simple conversions (mainly by 103E and 1002, and also by 103B and 103C), as described in Sec. 5.3; the second block contains five conversions implemented by double keystrokes, as described in Sec. 5.4; the third group contains looped conversions for five pairs (102D/102E, 102F/1030, 1031/1032, 1038/1037, and 103A/1039), as described in Sec. 5.5. Tables 2 and 3 cover the scheme proposed in Fig. 4 of Ding et al. (2019).

Tables 4 and 5 are related to normalization for variants of 102C/102B. Character sequences requiring alternation are listed. This issue was not explicitly mentioned by Ding et al. (2019); in this paper, it has been discussed in Sec. 5.6.

Table 6 contains the final normalization rules. The upper two rules related to 200B have been mentioned in Sec. 5.3 in terms of cleaning up the invisible separator to avoid ambiguities. The lower three rules are related to the Unicode scheme for generating the recommended encoding manner.

[13]They were not included in Ding et al. (2019).

in.	out.	in.	out.	in.	out.
,	104A	j	1038	F	1039
.	104B	k	1000	G	1003
0	1040	l	101C	H	101F
1	1041	m	1019	I	102E
2	1042	n	1014	J	1037
3	1043	p	1015	K	1001
4	1044	q	200B	L	1020
5	1045	r	103C	M	1036
6	1046	s	101E	N	100A
7	1047	t	1010	O	1029
8	1048	u	102F	P	1016
9	1049	v	102C	Q	1002
_____		w	103D	R	100B
b	1017	y	103B	S	1026
c	1005	z	1007	T	1011
d	1012	_____		U	1030
e	1031	A	104F	V	1021
f	103A	B	1018	W	101D
g	1002	C	1006	X	1004
h	103E	D	1013	Y	101A
i	102D	E	1032	Z	1008

in.	out.
a	1031 102C
o	102D 102F
x	1004 103A

Table 2: 64 rules of `in.` ||| `out.` ||| `0` ||| `0` .

In this example, an input method for 75 Unicode characters[14] are defined by 133 rules. More rules can be added to accommodate users' preferences. Generally, the number of rules can be two or three times the number of symbols. In practice, a scale of $10^2 - 10^3$ rules is suggested for an `akkhara` file to ensure easy edition and maintenance.

7 Conclusion and Future Work

This paper has described the AKKHARA, a general input method editor that can accommodate writing systems of medium complexity. AKKHARA is more powerful than a key-character mapper, but is simpler than input methods for Chinese and Japanese. Therefore, AKKHARA is easy to maintain and suitable for realizing customized input methods.

Future developments include: 1) developing AKKHARA for other OSs besides `Win`, 2) improving the interface of rule edition, and 3) integrating corpora for prediction functions.

[14] From `1000` to `104F`, without `1022`, `1028`, `1033`, `1034`, and `1035`.

in.	out.	in.	out.
1000 103E	1001	102D 1002	1023
1002 103E	1003	102E 1002	1024
1005 103E	1006	102F 1002	1025
1007 103E	1008	1030 1002	1026
1010 103E	1011	1031 1002	1027
1012 103E	1013	1014 103B	100A
1015 103E	1016	101E 103C	1029
1017 103E	1018	_____	
1004 1002	100F	102C 102C	1021
100A 1002	1009	103B 103B	101A
100F 1002	104C	103C 103C	101B
1010 1002	100B	103D 103D	101D
1011 1002	100C	103E 103E	101F
1012 1002	100D	_____	
1013 1002	100E	102D 102D	102E
1014 1002	1004	102E 102D	102D
1019 1002	1036	102F 102F	1030
101B 1002	104D	1030 102F	102F
101C 1002	1020	1031 1031	1032
101E 1002	103F	1032 1031	1031
1020 1002	104E	1038 1038	1037
1027 1002	104F	1037 1038	1038
1029 1002	102A	103A 103A	1039
102C 1002	102B	1039 103A	103A

Table 3: 46 rules of `in.` ||| `out.` ||| `1` ||| `0` .

*			
1001	1001 1031	1002	1002 1031
1004	1004 1031	1012	1012 1031
1015	1015 1031	101D	101D 1031
1002 103D		1002 103D 1031	
1012 103D		1012 103D 1031	

Table 4: 16 rules of `*` `102C` ||| `*` `102B` ||| `2` ||| `1` .

*	
1015 1039 1015	1015 1039 1015 1031

Table 5: Two rules of `*` `102B` ||| `*` `102C` ||| `3` ||| `1` .

in.	out.	in.	out.
200B 103E	103E	200B 1002	1002

in.	out.
1025 102E	1026
1029 1031 102C 103A	102A
103A 1037	1037 103A

Table 6: Five rules of `in.` ||| `out.` ||| `-1` ||| `0` .

References

Chenchen Ding, Masao Utiyama, and Eiichiro Sumita. 2018. Simplified abugidas. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 491–495, Melbourne, Australia. Association for Computational Linguistics.

Chenchen Ding, Masao Utiyama, and Eiichiro Sumita. 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*, pages 157–162, Hong Kong, China. Association for Computational Linguistics.

Ian Maddieson. 2013a. Consonant inventories. In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Ian Maddieson. 2013b. Vowel quality inventories. In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Hammam Riza, Michael Purwoadi, Teduh Uliniansyah, Aw Ai Ti, Sharifah Mahani Aljunied, Luong Chi Mai, Vu Tat Thang, Nguyen Phuong Thai, Rapid Sun, Vichet Chea, Khin Mar Soe, Khin Thandar Nwet, Masao Utiyama, and Chenchen Ding. 2016. Introduction of the asian language treebank. In *Proc. of O-COCOSDA*, pages 1–6.