# Unification of fonts encoding system of Devanagari writing in Nepali

Trishna Singh
strishna8 [at] gmail [dot] com

Santa B. Basnet
sbasnet81 [at] gmail [dot] com

**Integrated ICT, Jwagal, Lalitpur**
Nepal

**Abstract**

*We devise a generalized framework to unify the text written in single-byte TTF formats and the multi-byte Unicode format of Nepali Devanagari script. It incorporates finite state approach to understand text encodings and utilizes the character mapping table, runtime cache and the glyphs re-arrangement rules during conversion. We can seamlessly use the outcome of the system in Nepali language computation such as Spelling Correction, Information Retrieval and so on.*

**Keywords:** *font-encoding, glyph, tokenizer, state transition, cache*

## 1 Introduction

Writing system of around 95% languages in Nepal [3] uses Devanagari script which is also found in many Indian languages. Most computer applications afford people to use single-byte glyph based true type fonts (TTF) encoding technology to store computerized Nepali language text. The formation of a dependent or independent single character based on ASCII encoding relies on the composition of either single or multiple glyphs. For example: *"ण"* is a single character in Nepali which maps to the letters *"0f"* in the widely used TTF font Preeti. Some special symbols with single-byte code are mapped to represent composite characters in many TTF fonts for convenience in typing Nepali text. For example: भ्र and ज्ञ are mapped to the ASCII code *180* in Preeti and PCS Nepali fonts respectively. Many people have contributed to design the shape and the composition of glyphs for character formations using various computer tools. This has resulted in many non-standard and non-uniform true type character mapping fonts. These fonts are primarily designed with a focus on composing and writing Devanagari characters and less to represent the language itself. These irregularities are problematic for computational algorithms used in text processing.

The development of multi-byte Unicode encoding standard, which aims to cover almost all languages writing system used these days, make it straight forward to encode wide range of characters. The range includes the Devanagari alphabets. Computer algorithms can easily process Unicode encoded Devanagari script. Many institutions and most publication houses in Nepal have however developed their own glyphs for TTF fonts and Unicode. A lot of them have established themselves as industry standard with a large number of active users and an enormous volume of content. Such unmanaged diversity hinders processing and exchange of information through computers.

Most of the publications and news houses uses traditional TTF fonts to generate significant amount of Nepali text daily. These institutions also publish these materials online. The trouble is that internet browsers cannot process the traditional TTF fonts unless the font resources are in the user machine. Institutions that understand the inconvenience uses free software tools available on the internet to convert a few single-byte TTF fonts to its equivalent representation in Unicode encoding. Most of these tools are limited to few characters or work for a specific font such as Preeti. This introduces inconsistencies and errors in writing system of the language. For example, the Devanagari character "ई" can be obtained from two distinct sets of character combination in Unicode. The first is the single character denoted by the code *0x908*. The second way to render the same character is to combine two characters, "ि + ई". This is not the correct representation for the

language but nonetheless is available in the popular conversion tools. To resolve these issues, the paper puts forward a robust font unification system of the TTF family and Unicode in Devanagari script that preserves the integrity of all the glyphs conjuncts while writing Nepali text. The approach and its implementation can be useful to text processing system such as OCR, spelling correction, text archiving and searching and language parsing.

## 2 Background of font encoding and conversion

There have been earlier efforts to standardize the writing system of Nepali within computers. Nepal Codes for Information Interchange was proposed by a committee of experts from government organizations, universities and the telecommunications sector. [1] The recommendation introduces a standard that encodes Nepali Devanagari characters by including some conjuncts, partial letters, matra forms of the vowels and other diacritics all within the code range from 0 to 255. Hall et al. [2] and Hall [3] explained the necessity and the problems of achieving the unified encoding system for small languages in computer. These present the text encoding history of Nepal's languages in computer. Only three languages of Nepal have their written traditions, viz., Nepali, Newari and Limbu. They point out the coding limitation that of the Unicode which does not encodes phonological arguments, limited to certain expertise and bias towards the separation of unified text encoding among small languages. One of the main barrier is that the multiple disciplines of encoding present barriers to the development of software such as OCR and written text processing. Hardie [4] designed a set of mapping rules that converts the conjuncts created using half-form glyph-based 8-bit fonts to the equivalent conjuncts in Unicode for some South Asian languages. The effort produced a text conversion computer program named Unicodify[1] that has been used extensively in corpus building for South Asian languages.

Madan Puraskar Pustakalaya[2] has been active in developing font conversion tools, specifically non-Unicode fonts to Unicode. It has crafted a limited number of rules to convert text documents from few TTF fonts to Unicode and vice-versa. The tools are released with the name *"Conversion Tools 3.0"*. The limited rules are manually encoded in individual program modules for the respective fonts and do not share common codes among font families. This contributes limited usability with its constricted coverage and performance. UNESCO[3] has supported this work aiming to build the standard font for software professionals to develop language based utility applications such as dictionary, spell-checker in Nepali. There are many other similar small utility programs online to convert TTF to Unicode font and vice-versa. Most of them do not have the full coverage of TTF font codes and conjuncts assembled using the TTF font glyphs. These produces conversion errors and requires significant manual effort to correct them.

Raj and Prahallad [5] have developed the font encoding identification scheme by utilizing the Vector Space Model (VSM) and the term frequency – inverse document frequency (tf-idf) metrics. Glyphs are used analogous to terms and the words with its sentences as the documents. The n-gram measures (uni-glyph, bi-glyph and tri-glyph) are used in document weighting to estimate the models output of all font-types during font identification. A set of generic glyph assimilation rules are designed to convert the font's data, which uses the predefined mapping table of glyphs, from one font to another for a particular language text. The glyph assimilation rules re-arrange to the proper position and combines some character glyphs so that it forms a conjunct during font conversion.

[1] *http://www.lancaster.ac.uk/staff/hardiea/unicodify.htm*
[2] *http://www.madanpuraskar.org*
[3] *http://www.unesco.org/new/en/brasilia/about-this-office/single-view/news/unesco_supports_nepali_font_standardization/*

Lehal et al. [6] propose a statistical model to convert multiple formats of available fonts encoded in ASCII to Devanagari Unicode. Mapping characters to a single code value in Unicode rely on Tri-gram language model and uses fonts mapping tables. The font mapping is done in two stages: first, it converts all the characters of detected fonts to the intermediate code which corresponds to the glyph of a font. The intermediate set of codes are designed and trained to cover all set of available fonts and the mapping rules are designed according to the glyphs. Second, the intermediate representation of codes is mapped to Unicode and re-arranged by the transformation rules. These rules are carefully crafted to perform the Unicode transformation based on the character categories.

Our method relies on the characters categorized in the TTF fonts, the position of glyph appearances with other characters and the font mapping table. We craft generic rules for the glyph re-arrangement and their cache so that the font conversion works in both directions easily. New fonts can be added in the conversion system by providing minimum amount of font information.

## 3 The font unification system

The unification means, we analyze most of the available writing system present currently in Nepali text expressed via true type i.e. the single byte and Unicode i.e. the multi-byte fonts and build a computational processing algorithm that understands the written conjuncts of character sequences. This preserves the integrity of glyphs expressed for a particular character used among diversified fonts. For example, the word "ज्ञानझुण्ड" (meaning: knowledge cluster) can be represented with many different fonts are shown in the following table.

| *Font* | *Word* | *Bytes Representation* |
|---|---|---|
| *Unicode* | ज्ञानझुण्ड | 91C 94D 91E 93E 928 91D 941 923 94D 921 |
| *Preeti* | ज्ञानभ्फुण्ड, ज्ञानभुण्ड | 31 66 67 B4 27 30 38, 31 66 67 65 27 6D 30 38 |
| *PCS Nepal* | ज्ञान□ण्ड, ज्ञानभुण्ड | 21 66 67 B4 27 29 2A, 21 66 67 65 27 6D 29 2A |
| *Kantipur* | ज्ञानभ्फुण्ड, ज्ञानभुण्ड | 31 66 67 B4 27 30 38, 31 66 67 65 27 6D 30 38 |
| *Himalb* | **ज्ञानझुण्ड,** **ज्ञानभुण्ड** | 31 66 67 B4 27 30 38, 31 66 67 65 27 6D 30 38 |

*Table 3.1: Sample word representation in different Nepali TTF fonts and Unicode.*

As shown in table 3.1, the same word can be written in multiple forms (preserves the same meaning) in the same font as well as it varies the internal representation among different fonts. In particular भ, भ and □ shares common as well as different codes among same and different TTF font family. It also has variation length of glyphs. The character can be expressed single byte of code "**B4**" and also multi byte codes "**65 6D**". The equivalent representation of the same character in Unicode is "**91D**". In similar, there are a lot of code variations exists and that should be coped with the unification system.

Our unification system assumes that the Unicode is the standard to represent all the glyphs of font data. So, this system interacts with the following components in fig 3.1 to unify all the fonts to Unicode and vice versa.

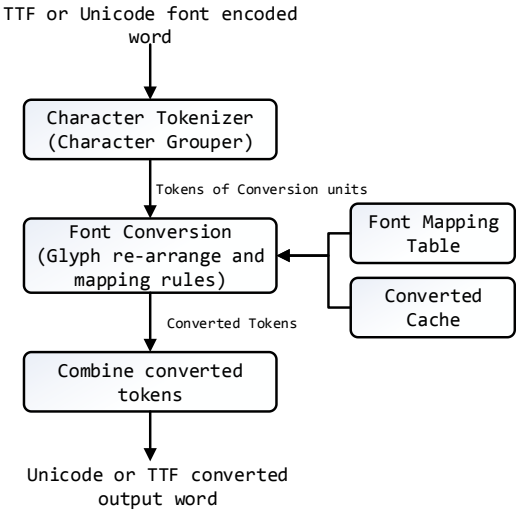



*Fig. 3.1 Font unification system components.*

Initially, we group the sequences of character glyphs in both TTF and Unicode encoded text as the minimal conversion unit by using a character tokenizer. The tokenized output then send to the conversion system which utilizes the font mapping rules and glyph re-arrange rules based the glyph category types. Finally, the converted sequences of

characters then combined to form the whole word. The individual component details are described in the following sections.

### 3.1 Character tokenizer for glyphs grouping

We manually organize the glyphs in the fonts in order to group categorized characters that could join with the others. The grouping of glyphs are based either on the position it could appear during conjunct formation, or whether they could appear alone. The assumption behind this grouping is that native speakers know which character goes where, that is, if it should be placed on the left, top, right and the bottom of the center character. The center character in this case could be whole consonant (WC) or whole vowel (WV). The same idea [6] is implemented for Telugu glyphs by assigning the position with number values. We define fifteen categories for font glyphs such as whole consonants (WC), half consonants (HFC), compound characters (CSC) and right vowels (RV). For example, RV for the Kantipur TTF font is the set {ा,ी}. All of the character groups (labelled with distinct acronyms) are given in Appendix A. We name these categories according to the order in the sequence of conjuncts. We then build a finite-state machine to capture the character transition in conjunct formation. The transition among the categories is described by a memory based state machine known as Pushdown Automata (PDA)[4]. The state machine for the TTF and Unicode character tokenizer is formally defined by five tuples, as follows:

**TTF Tokenizer ($T_1$)** = (

Set of States (Q): {1, 2, …, 20},

Set of Input Categories (Σ): {WC, WV, NUM, CSC, BV, TV, LV, RV, BDU, HLN, HFC, RHF and ELSE},

Starting state ($q_0$): 1,

Set of final states (F): {19},

Set of transitions (δ): See all transitions are shown in Appendix – C

)

*Fig 3.2 Definition for TTF text sequence tokenizer.*

**Unicode Tokenizer ($T_2$)** = (

Set of States (Q): {1, 2, 3, 4, 5, 6, 7, 8},

Set of Input Categories (Σ): {WC, WV, NUM, PCM, ZWJ, ZWNJ, JNS},

Starting state ($q_0$): 1,

Set of final states (F): {8},

Set of transitions (δ): See all transitions are shown in Appendix – B

)

*Fig 3.3 Definition for Unicode text sequence tokenizer.*

The word "राष्ट्रियताले" results from transitions that generate these tokens in sequence with given transitions among TTF fonts: [रा{WC_RV}, ष्ट्र{LV_HFC_WC_HLN}, य{WC}, ता{WC_RV}, ले{WC_TV}]. For example, a character token sequence "ष्ट्र" has LV, HFC, WC and HLN category types. These tokens are taken to be the minimal unit of conversion in order to apply the font conversion map and the glyph arrangement rules.

### 3.2 Font mapping table

The mapping table organizes both single and multiple mapping entries needed for all other fonts to Unicode and vice-versa conversion. We consider a set of ASCII codes are shared as a common code for a single glyph among all the TTF fonts. To accommodate it, we build the mapping table in two stages. First, we find intersection among all the codes in the TTF font family and select a set of commonly shared codes. We then prepare a mapping table for these entries. The key benefit of extracting common codes is that it saves a huge amount of run-time memory by avoiding redundant map entries among all the available fonts. In a system with $k$ fonts, the codes are extracted using the following relation:

$$Common\ Codes = \bigcap_{i=1}^{k} (TTF\ Font)_i \quad --- (3.1)$$

[4] *https://en.wikipedia.org/wiki/Pushdown_automaton*

In the second stage, we build the mapping table for all uncovered codes. It also has few entries of composite characters (conjuncts) formed by multiple glyphs. This is done so that some of the common arrangement could easily be mapped from the entry itself.

### 3.3 Font conversion and glyph arrangement with cache

We now perform conversion on the output of the character tokenizer. Since we consider a token of character sequences to be the minimal meaningful conversion unit for font conversion, we apply the available fonts map to either to a single character or a sequence of characters. Our system utilizes several re-arrangement rules among the glyphs to preserve the sanity of the Nepali writing system. For example:

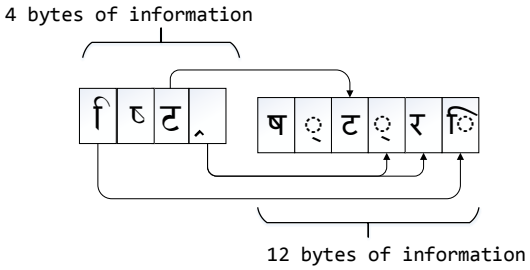



*Fig 3.4 Sample of glyphs mapping and re-arrangement during Unicode conversion.*

The four bytes of information in "ष्ट्रि{LV_HFC_WC_HLN}", shown in Fig. 3.4, is mapped to the 12 bytes of equivalent Unicode information. First re-arrangement of glyphs happens on the LV, i.e., the "ि" character. It is sent to the end to meet the sequence in Unicode writing. The special character "ॅ" is mapped to four bytes by adding half maker (◌ॅ) to the previous character and a full character "र". In another example, "औं" can have two glyphs sequences: [अ ा ◌ॅ ◌ै] and [अ ा ◌ै ◌ॅ] in Preeti font. The re-arrangement rules need to address such type-sequence variations in order to preserve the integrity of Nepali writing. We devise 15 wide-ranging glyph re-arrangement rules for the TTF and Unicode fonts to deal with all possible writing style of Nepali commonly encountered in the computer.

We build a runtime cache memory table produced in the forward conversion (i.e. from TTF to Unicode) with an aim to preserve the same set of glyphs that can be used while converting back from Unicode to TTF. There is a lot of one to many mappings from Unicode to TTF conversion. For instance, a character "झ" of Unicode can be mapped to either "□" or "झ" in PCS Nepal font, i.e., while designing a TTF font, it is provided that multiple characters of single byte code can be mapped to same character symbol in Unicode. This especially happens to the composite type of characters, made from multiple glyphs. So, in case of backward conversion, we prioritize more to the runtime cache than the offline map table.

#### 3.3.1 Forward Conversion, i.e., TTF to Unicode:

In this step, the TTF character token sequences are given as the input, and it produces the character sequences in Unicode as the output.

*Algorithm – 3.1: Conversion of TTF font to its equivalent Unicode representation.*

```
-------------------------------------
(1) Read a TTF font encoded word and
build a stream of character tokens.

(2) For all tokens:

   (a) Check if the token have single
   glyph, apply conversion map.

   (b) Check if the token have two
   glyphs, apply conversion map with
   left vowel and top vowel glyphs
   arrangement rules.

   (c) Check if the token have multiple
   glyphs, apply conversion map with
   all the crafted glyphs arrangement
   rules.

   (d) Store the applied map entry and
   rule in the runtime cache table.

(3) Concatenate all the converted
tokens to form a new equivalent
Unicode text.
-------------------------------------
```

#### 3.3.2 Backward Conversion i.e. Unicode to TTF:

In this step, the Unicode character token sequences are taken as the input and it produces the character sequences in TTF fonts. During this conversion, it utilizes cache of the character map created during forward conversion to resolve the ambiguity of one-to-many mappings from a single Unicode character to the multiple characters in TTF glyphs.

*Algorithm – 3.2: Conversion of Unicode font to its equivalent TTF representation*

```
-------------------------------------
(1) Read a Unicode encoded word and
build a stream of character tokens.

(2) For all tokens:

   a. Check if the token have single
   glyph, apply conversion map and the
   cached map.

   b. Check if the token have two
   glyphs, apply conversion map and the
   cached map with left vowel glyphs
   arrangement reverse rule.

   c. Check if the token have multiple
   glyphs, apply conversion map and the
   cached map with all the crafted
   glyphs arrangement reverse rules.

(3) Concatenate all the converted
tokens to form a new equivalent TTF
text.
-------------------------------------
```

## 4 Performance analysis

We prepare the hand labelled font converted word-list to measure the accuracy of the font unification system.

### 4.1 Test data preparation and criteria

We sampled 3000 unique words from the total of 690,000 unique words extracted from the Nepali news corpus of approximately 700,000 documents. We prepare the test word-list encoded by the selected eight Nepali TTF fonts and the equivalent list in Unicode. Hence, we have eight sets of TTF encoded and its equivalent Unicode encoded test words.

### 4.2 Correctness measure

We match the source word to the converted word as a measure of string matching in both ways in the tests. We collect all the matched words as the correct words. The system evaluation given by the accuracy measure in percentage is given by the following relation.

$$Accuracy = \frac{Correctly\ Converted\ Words}{Total\ Words} \quad ---(4.1)$$

We calculate the average accumulated accuracy score of the system among all eight selected fonts by the following relation.

$$Average\ Accuracy = \frac{\sum_{i=1}^{k} Accuracy(Font)_i}{K} \quad ---(4.2)$$

The evaluation of font unification system result is shown in the following table 4.1.

| ***Font Name*** | ***Accuracy*** |
|---|---|
| *Kantipur* | 100.00 % |
| *Preeti* | 99.96 % |
| *Himalb* | 100.00 % |
| *PCS Nepali* | 99.88 % |
| *Aalekh* | 99.73 % |
| *Aakriti* | 99.73 % |
| *Ganesh* | 100.00 % |
| *Navjeevan* | 99.92 % |
| | *Avg. Accuracy = 99.90 %* |

*Table 4.1 Summary result of accuracy evaluation*

## 5 Conclusion and Future works

The paper presented a general framework to unify all the available Nepali TTF fonts and Unicode encoded text. It enables to build computational frameworks on top of this system. The minimal data i.e. token groups and font map table can be easily extended to incorporate new font encoding for additional service. Our system makes it viable to build Nepali language tools such as spelling correction, information retrieval (searching) and morphology analysis without worrying about the font encoding used. We achieve nearly 100% accuracy on both direction of the font conversion pipelines. We can extend this framework to convert all of the writing scripts and languages of Nepal by utilizing statistical language and script detection components. The Vector-Space Model[5] with TF-IDF measure[6] can be useful in this regard. For more generalization, we can automatically learn the font-mapping table and the glyph re-

[5] *https://en.wikipedia.org/wiki/Vector_space_model*

[6] *https://en.wikipedia.org/wiki/Tf%E2%80%93idf*

arrangement rules using statistical machine learning methods. These two are our future works set to improve the unification system, make it robust and generalize so that no human involvement is required.

## 6 References


1. Nepali Font Standardisation Committee, *Nepali Font Standards*, White Paper, Version 2 (1998)

2. Pat Hall, Bal K. Bal, Sagun Dhakhwa, Bhim N. Regmi, Issues in Encoding the Writing of Nepal's Languages, *CICLing 2014*, Part I, LNCS 8403, pp. 52-67 (2014)

3. Pat Hall, Computerized writing for small languages, *The Journal of Specialised Translation*, Issue 24 (February, 2015)

4. Andrew Hardie, From legacy encodings to Unicode: the graphical and logical principles in the scripts of South Asia, *Language Resources and Evaluation*, Volume 41, Issue 1, pp 1-25 (2007)

5. Anand A. Raj, Kishore Prahallad, Identification and Conversion of Font-Data in Indian languages, *International conference on Uniersal Digital Library(ICUDL2007)*, Pittsberg, USA, (November 2007)

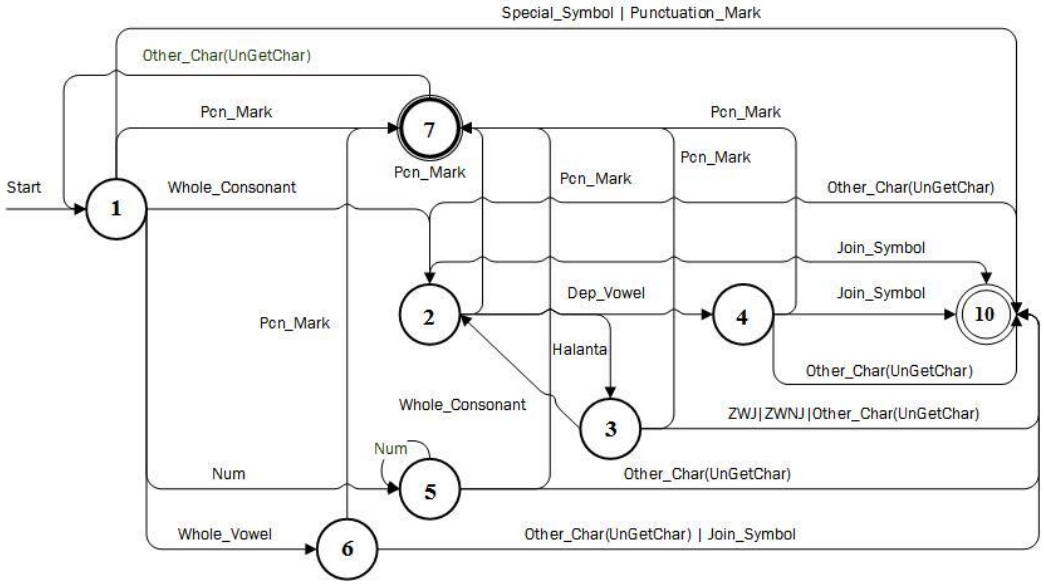
6. Gurpreet S. Lehal, Tejinder Singh, Saini P. K. Buttar, *Research in Computing Science 86*, pp. 9-23, (2014)


## 7 Appendixes

### A TTF and Unicode encoding token types list

1. Whole Consonants (WC)
2. Half Consonants (HFC)
3. Compound Characters (CSC)
4. Whole Vowels (WV)
5. Right Vowels (RV)
6. Left Vowels (LV)
7. Top Vowels (TV)
8. Bottom Vowels (BV)
9. Special Characters (SPC)
10. Special Symbols (SPS)
11. Halanta (HLN)
12. Bindu (BDU)
13. Numbers (NUM)
14. Right Half (RHF)
15. Half Maker

### B State transitions of Unicode encoded text tokenizer



## C. State transitions of TTF encoded text tokenizer