GFUC: Gurmukhi Font and Unicode Converter

Gurjot Singh Mahi
Information Technology Lab,
Rajiv Gandhi National Institute of Youth Development, Regional Centre,
Chandigarh, India

Amandeep Verma
Department of Computer Science,
Punjabi University Regional Centre for Information Technology and Management,
Mohali, India

ABSTRACT
Growth of information technology has played a great role in connecting the world together. The to and fro of information is common in this world. Fonts play a key major role in this communication process in digital domain. Common encoding scheme for one language helps in loss-less digital communication. Indian fonts lacks in this zone, as no Indian font has standard encoding format for mapping characters. Numerous indic fonts were created with diverse mapping schemes. Gurmukhi as one of the prominent Indian script also suffered from this negligence. This study investigates the Gurmukhi font and Unicode converter, which works for font to font substitution and font to Unicode substitution using an algorithmic process taking intake of Gurmukhi text written in OpenXML document. This converter works for 5 ASCII based legacy Gurmukhi fonts and tries to handle the diverse mapping scheme of these fonts. It gives the hustle free substitution mechanism for both inter-font and Unicode conversion. The performance of GFUC is measure of various well-defined norms and gives 100% accuracy during conversion.

General Terms
Natural Language Processing, Information Science, Substitution Algorithm, Unicode, Algorithm, Rendering

Keywords
ASCII, Unicode, Gurmukhi fonts, Gurmukhi Font Substitution, Conversion System.

1. INTRODUCTION
Technical development of any language depends upon the availability of literature in digital format so that it can be used in various natural language processing technologies. This literature is kept in digital format using a scheme known as encoding in various legacy documents. Encoding provides a special byte code to each character in a particular language. Indian fonts were also designed following this criterion. As due to non-standardization of character mapping schemes, many Indian fonts were encoded in miscellaneous manner using ASCII [1] encoding format which was intended only for English language. This can cause a problem in case of Indian languages, as with the due course of time large amount of literature was written using these ASCII legacy fonts which do not contain any sematic meaning in digital domain and was usually prone to information loss during substitution. The studies like [2] and [3] has proven that information loss occurs during substitution between various fonts of particular language.

Digital Gurmukhi fonts are also suffering from this trouble prevailing in digital domain, which prevents the growth and development of natural language applications in Gurmukhi language. To tackle this problem for Gurmukhi script the present study was intended to develop an application – Gurmukhi Font and Unicode Converter (GFUC). The GFUC is designed to deal with two problem areas – "Publishing" and "Machine Readability". Book publishers tends to use the ASCII Gurmukhi fonts in books publishing and from the term machine readability, it is an attempt to make ASCII fonts machine readable by performing substitution of one Gurmukhi ASCII font into equivalent Unicode [4] format. GFUC uses an algorithm process to address the problem for 5 Gurmukhi fonts – "*Joy", "Gurbani Akhar", "Anandpur Sahib", "Akhar"* and *"Sukhmani"*. GFUC design is focused on two key areas – "Font to Font Substitution" and "Font to Unicode Substitution". In "Font to Font Substitution", one Legacy Gurmukhi font is substituted with another Gurmukhi font, which resolves the problem in publishing domain and in "Font to Unicode Substitution", one ASCII Gurmukhi font is substituted with its equivalent Unicode byte code depending upon the type of font user choose, giving semantic meaning to Gurmukhi ASCII bases text. GFUC is capable of giving 100% Substitution accuracy for both Font to Font conversion and Font to Unicode substitution. It is an application for NLP domain which provides both type of conversions with complete accuracy for mentioned 5 legacy Gurmukhi ASCII fonts.

2. RELATED LITERATURE
There are various ideas closed to the said proposed work, some of them are converters developed using various efficient algorithmic techniques and some by using graph assimilation process. It began when first Devanagari font converter was published using an algorithmic approach in [5] in context of Indian fonts. [6] developed an Intelligent Bengali Unicode Converter (IBUC) in which authors have proposed an algorithm for efficient conversion of Bengali ASCII based fonts to Unicode. IBUC gives 100% accuracy rate as compared to other Bengali font converters like Acro and Nikosh, and is successful in converting the other Bengali fonts like "*AdarshaLipi*" and "*MoinaExpanded*" which was not considered in other font converters.

A language based font converter was developed by [7]. New TF-IDF based approach was designed, in which a glyph assimilation process was used for identification and conversion of fonts. The proposed work has reported an accuracy of 99% for 10 Indian languages. An omni font convertor has been designed by [8] for Gurmukhi to Shahmukhi transliteration purpose. This proposed work identifies the Punjabi font using the character level trigram language model. The trigram probability is calculated at word level for conversion of Punjabi font into the Unicode format. The system has achieved 99.75% ASCII to Unicode conversion accuracy at word level.

3. PROBLEM COMPLEXITY

Earlier studies, [3] and [9] clearly demonstrated that information loss occurred during the substitution of one Gurmukhi legacy font with other Gurmukhi legacy font. The core basic reason for the design of GFUC is non-standardized design of these Gurmukhi fonts. The other major problem areas which were channelized during the design of GFUC are discussed as follows:

3.1 Non Availability of Well-Defined Code Points

Gurmukhi keys are mapped on different code points using ASCII font encoding format. Due to different mapping schemes, it become hard to design a natural language processing application, as much of the text used for training purposes in various learning algorithms is done using these Gurmukhi fonts written using numerous fonts mapping schemes. For example, the character "ਓ" is mapped on binary value 01010100 in *"joy"* Gurmukhi font, on 01010000 in *"AnandpurSahib"* font and on 01100001 in *"GurbaniAkhar"* font. This is just one case of different code mappings for similar character in different fonts, but same is the case with available 255 Gurmukhi fonts. It cause a big hoax in the field of language processing and confines the practice of Gurmukhi script in language technology development. The decimal code point for the keyword "ਪੰਜਾਬੀ" is represented in Table 1 [8].

Table 1 shows the Gurmukhi script character "ਪ" is mapped on 0067, 0066, 0070, 0050 and on 00EA decimal code points in *"Asees"*, *"Gold"*, *"Satluj"*, *"Sukhmani"* and *"P-RUBY"* Gurmukhi fonts respectively. This rigid and problematic property of ASCII Gurmukhi fonts makes them hard to be used by researchers for simple NLP tasks and for publishing by various publishing houses.

Table 1. Decimal representation of "ਪੰਜਾਬੀ" in various Gurmukhi fonts

Gurmukhi Font	Decimal Representation of "ਪੰਜਾਬੀ"
Asees	0067+007A+0069+006B+0070+0068
Gold	0066+002E+0075+006A+0057+0067
Satluj	0070+00B5+006A+003B+0062+0049
Sukhmani	0050+005E+004A+0041+0042+0049
P-RUBY	00EA+00B3+00DC+00C5+00EC+00C6

3.2 Typing Complexity

Due to different code points for more than 255 Gurmukhi fonts and use of dissimilar keyboard formats like Inscript and Phonetic for plotting Gurmukhi characters on keyboard keys, make it challenging for typewriters to type these fonts using keyboard. As for example, Fig. 1 demonstrates the working of various Gurmukhi keyboard formats. If we want to type "ਸਮਾਜਿਕ" in *"joy"* font which is particularly designed using Inscript keyboard design, then we have to press "; + w + k + f + I + e" keys and similarly, in *"Sukhmani"* font which is designed using phonetic keyboard, we have to press "S + M + A + E + J + K" keys. This uneven distribution of keys in these fonts makes the Gurmukhi typing more difficult for typewriter to type using one font which is not according to their learned format of keyboard.

; + w + k + f + I + e(Joy)
S + M + A + E + J + K (Sukhmani)

Fig. 1. Keyboard keys combination representation in two fonts for typing "ਸਮਾਜਿਕ"

3.3 Unicode Rendering Problem

Although it is possible to convert the legacy Gurmukhi font into Unicode standard but sometime this result in incorrect semantic value of word, due to inappropriate rendering of characters in some case. This indifference between rendering of characters in ASCII and Unicode creates problem, as if we want to type "ਕਿ" in Gurmukhi ASCII font, then we place Gurmukhi vowel sign "I" and then Gurmukhi letter "KA". While if the same process is followed in Unicode, this result in incorrect word with no sematic meaning in Gurmukhi script as shown in Fig. 2.

ਿ+ ਕ → ਕਿ (Rendering in ASCII font)
ਿ + ਕ → ਿਕ (Rendering in Unicode)

Fig. 2. Rendering mechanism in Gurmukhi ASCII font and Unicode

3.4 Handling Gurmukhi Special Vowels

Typically in legacy Gurmukhi ASCII fonts, long vowels like KHHA(ਖ਼), GHHA(ਗ਼), ZA(ਜ਼), SHA(ਸ਼), LLA(ਲ਼) and FA(ਫ਼) are typed with the help of two characters. Whereas, in Unicode contains unique code values for these long vowels. For example, if we want to type ਲ਼ in *"joy"* font, it is typed by using the keys a + b, but in Unicode it is mapped special code value - 0A33. This uneven arrangement between two different standards of font creates problem in handling these long vowels w.r.t. Gurmukhi script.

4 SYSTEM FRAMEWORK

The proposed application shown in Fig. 3 was designed to enable the user to convert legacy Gurmukhi fonts efficiently and without error. This entire system was designed on the PC (Pentium(R) Dual-Core CPU T440 @ 2.20 GHz, 4GB RAM, Windows 8 and Ubuntu Platform, Python). The time complexity of this system is measured to be O(n).

The open XML file is used as the target file format to perform substitution. Whole system design is divided in 4 stages:

1. Parsing OpenXML file document.
2. Design of mapping dictionaries.
3. Designing of substituted algorithm.
4. Assembling all Parts in GFUC System Framework.

In the first step, parsing of OpenXML document takes place and text is extracted. In second step, dictionaries were created for GFUC system application in which manual font mappings were developed to make the finale processed document error free. Third step consist of designing of substitution algorithm for extracted text. Finally, the modules were assembled to make one Gurmukhi Font and Unicode Converter.

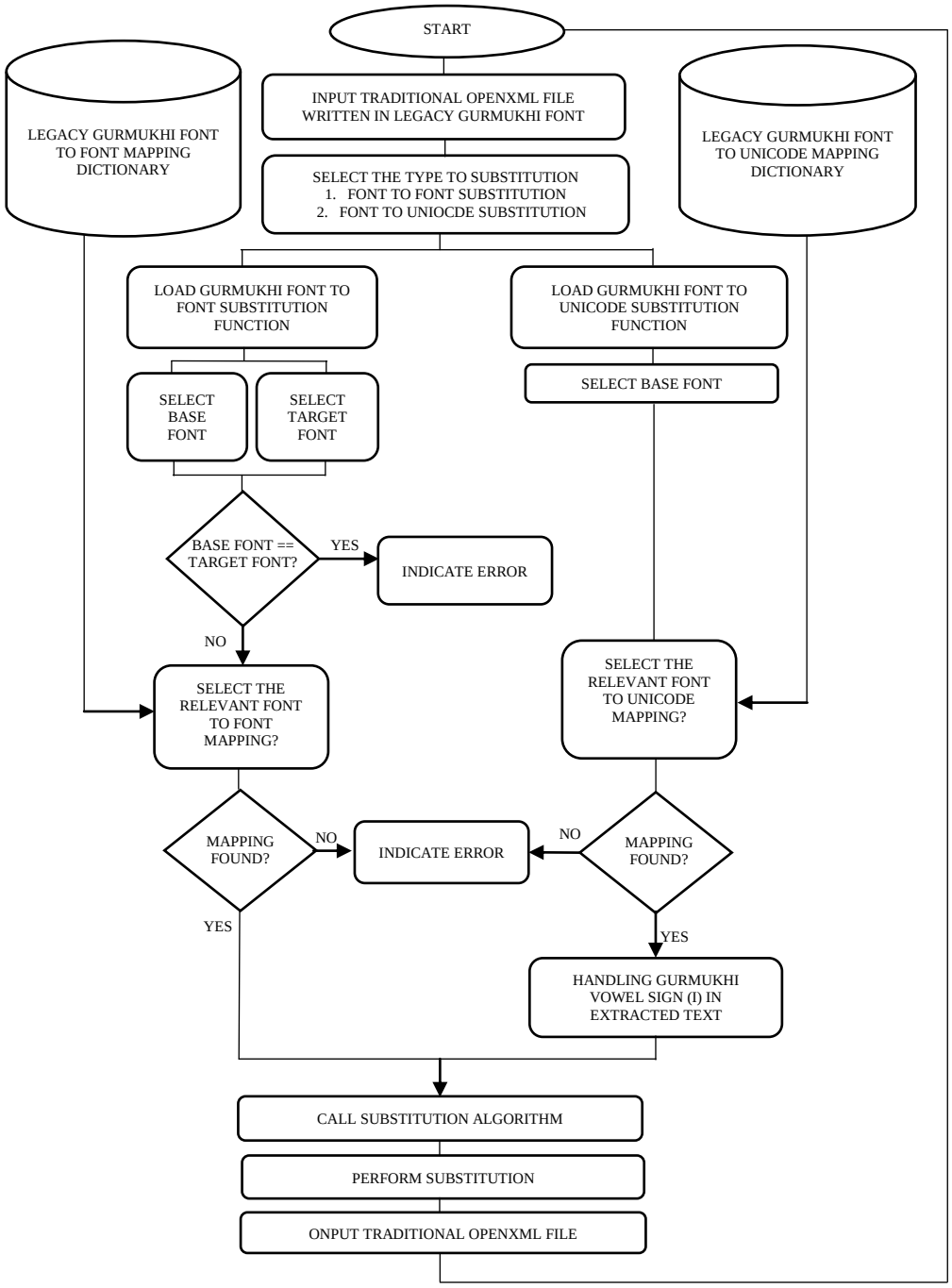

Fig. 3. System framework of GFUC

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?mso-application progid="Word.Document"?>
<pkg:package xmlns:pkg="http://schemas.microsoft.com/office/2006/xmlPackage">
.
.
.
.
.
<w:body>.....<w:rFonts w:ascii="Joy" w:hAnsi="Joy"/>.....<w:t>gzikph fposKs dh ouBkFftT[As ns/ f;oiDekoh ftu ;wkfie fBnK ns/
wkBtFw[esh dh ;zebgkswesk e/doh d/ o{g ftu rsh ojh. </w:t>
.
.
.
.
</pkg:package>
```

Fig. 4. Internal representation of Gurmukhi OpenXML document

4.1 Parsing OpenXML File Document

The initial step of GFUC mechanism was to extract the Gurmukhi text from the traditional Microsoft word file which is saved using .doc or .docx extension. The beauty of these format is that Microsoft document at back-end is saved in OpenXML format as shown in Fig. 4, which is usually said as original ECMA-376[10] standard, which is now represented under ISO as ISO/IEC 29500-1:2008 standard. This standard defines the XML set of vocabularies to represent the word-processing document [11]. We have used Textract 1.2.0 [12] to parse and extract the text from OpenXML word document. This structured Gurmukhi text is the data for which substitution function will be executed in upcoming steps.

4.2 Design of Mapping Dictionaries

Dictionaries, known as the abstract data types are chosen as the standard data-type for database creation. The basic aim of taking dictionaries as a standard data-type for our system was that dictionaries are accessed by their keys and not via its position [13], as in case of using arrays and linked lists as a data type for storage medium that could put unnecessary burden on the system design. Mappings were created for total 61 characters in Font to Font substitution and 70 characters in Font to Unicode substitution. This leads us to creation of total 25 dictionaries for substitution purpose, in which 20 dictionaries were created for font to font substitution purpose and 5 for font to Unicode substitution. Each Gurmukhi font key in dictionary is mapped to its relevant key in another Gurmukhi font and vice-versa. In this way Gurmukhi character keys were manually mapped to each other for 5 Gurmukhi fonts and same thing was achieved for Unicode Conversion. The example of Joy to GurbaniAkhar and Joy to Unicode dictionary is showcased in Fig. 5 and Fig. 6.

4.3 Text Substitution Algorithm

To make the Substitution work, we came up with a Gurmukhi text substitution algorithm. It is a 5 step algorithm. We have used five python inbuilt functions to create our own Gurmukhi font replacement module. KWARGS, join, enumerate, idx and get are five inbuilt functions used. SUBSTITUTION algorithm was created as an internal part of SUBSTITUTED_TEXT algorithm. It is designed to replace each character extracted from the Gurmukhi OpenXML document file from base font to target font selected in SUBSTITUTED_TEXT algorithm. The text extracted from the OpenXML document and dictionary chosen in the SUBSTITUTED_TEXT algorithm is passed to SUBSTITUTION algorithm by using OPENXMLDOCUMENTTEXT and KWARGS keyword.

Selected Gurmukhi key dictionary is created using a comma separated list of 'key':'value' pairs within curly braces, an example is shown in Fig. 5 and Fig. 6. As said earlier, the selected dictionary is passed to SUBSTITUTION function using KWARGS keyword. KWARGS permits SUBSTITUTION function to pass arbitrary number of keyword arguments from SELECTED_DICTIONARY. All unique dictionary character keys are loaded in the All_Characters in step 1 using KWARGS.keys(), in which keys module return the list of each available key to All_Characters. All_Characters now holds the entire list of unique keys in dictionary, like [[T],[n],[J],[;],[j],[e],[y],……].

In step 2, step 3 is repeated to compute the index value (idx) and unique keyword (k) in OPENXMLDOCUMENTTEXT using enumerate keyword which iterates the Gurmukhi text keywords one by one.

In step 3, step 4 is repeated for each key variable in All_Characters. In step 4, if unique keyword k is present in enumerated OPENXMLDOCUMENTTEXT text, which is performed using ".join(key) then that unique key k is replaced by its index (idx) position in enumerated OPENXMLDOCUMENTTEXT using:

OPENXMLDOCUMENTTEXT[idx]=
KWARGS.get(".join(key))

where .get function encapsulate the new value for unique ".join(key) value against old key at idx. Step 5 joins the replace key k in OPENXMLDOCUMENTTEXT using ".join(OPENXMLDOCUMENTTEXT) and returns the substituted text.

We now formally state the substitution algorithm in Fig. 7.

4.4 Assembling all Parts in GFUC System Framework

In the first step, test was extracted/parse from the OpenXML document, in second step mappings were manually designed and in third step text substitution algorithm was proposed. In the last step, user defined function is implemented in the form of Gurmukhi font and Unicode Converter (GFUC). The user defined function is further divided in two categories:

```
joy2gurbaniakhar = { 'T':'a', 'n':'A', 'J':'e', ';':'s', 'j':'h', 'e':'k', 'y':'K', 'r':'g', 'x':'G',
'C':'|', 'u':'c', 'S':'C',     'i':'j', 'M':'J', 'R':'\\', 'N':'t', 'm':'T', 'v':'f', 'Y':'F', 'D':'x',
's':'q', 'E':'Q', 'd':'d', 'X':'D', 'B':'n', 'g':'p', 'c':'P', 'p':'b', 'G':'B', 'w':'m', ':':'X',
'o':'r', 'b':'l', 't':'v', 'V':'V', 'ô':'S', 'õ':'^', 'ö':'Z', '÷':'z', 'ø':'&', 'ÿ':'L',    ']':'IN',
'A':'N', '/':'y', 'k':'w', 'f':'i', 'h':'I', '?':'Y', '[':'u', '{':'U', '\'':'o', '\"':'O', 'U':'E',
'K':'W', 'Q':'H',   'P':'H', 'q':'R', 'z':'M', 'Z':'~', 'L':':', '.':'[', 'F':'-', 'H':'.', 'W':'hY'}
joy2unicode = {'T':'ਉ', 'n':'ਅ', 'J':'ਏ', ';':'ਸ', 'j':'ਹ', 'e':'ਕ', 'y':'ਖ', 'r':'ਗ', 'x':'ਘ', 'C':'ਛ',
```

Fig. 5. Implemented Joy to Gurbani Akhar key mapping dictionary

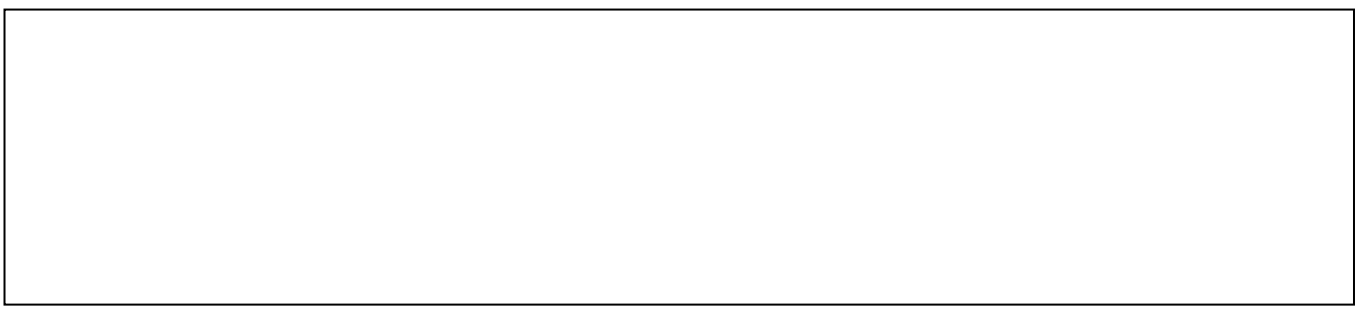

Fig. 6. Implemented Joy to Unicode key mapping dictionary

```
SUBSTITUTION(OPENXMLDOCUMENTTEXT, KWARGS)
OPENXMLDOCUMENTTEXT: Text extracted by parsing Gurmukhi OpenXML document
KWARGS: Selected dictionary in SUBSTITUTED_TEXT Algorithm

1. Set All_Characters <- [[ab for ab in k] for k in KWARGS.keys()]
2. Repeat 3, for idx, k in enumerate(OPENXMLDOCUMENTTEXT):
3.      Repeat 4, for key in All_Characters:
4.              if k in ".join(key), then:
                        OPENXMLDOCUMENTTEXT[idx] = KWARGS.get(".join(key))
                [end of if Statement]
        [End of inner loop]
   [End of outer loop]
5. Return ".join(OPENXMLDOCUMENTTEXT)
```

Fig. 7. Substitution algorithm

1. Font to font conversion.
2. Font to Unicode conversion.

Both the functionalities are user defined. From the user-defined we mean to say that the user has been given power to choose base font in which the legacy OpenXML document has been written and also the target font is selected, in which user want to convert the base font. One cannot select the same Gurmukhi font as the base font and target font. A constraint has been added in which system will raise an error in this type of scenario and to restrict this action by the user. If the user selected the different base font and target font, the relevant font mapping dictionary will be selected from the database which is mentioned in step 1 of this system. Also is the dictionary is not available in the database, the constraint has been added to the system which show error in this case. To make the conversion happen the substitution function was created and implemented. If the dictionary is found, the extracted text is sent to this substitution function and substituted text OpenXML file is taken as output. In the same way, second functionality includes all the steps but instead of selection of both base and target font, only base font in selected and equivalent font mapping dictionary is extracted from database. If the mapping is found the substitution is performed and resultant substituted OpenXML file with Unicode encoding is taken as output at the end

The proposed is intended to extract text from the Open XML document and perform the conversion from ASCII Base Gurmukhi font to any other ASCII target Gurmukhi font or into Unicode. To perform this conversion, we came up with an algorithm –Font Selection and Conversion Algorithm showcased in Fig. 8.

5. FONT SELECTION AND CONVERSION ALGORITHM

Initially, in step 1, the algorithm takes the user based type of font substitution i.e. font to font substitution (FONT2FONT) or font to Unicode substitution (FONT2UNICODE) and this user based selection is passed to CHOICE variable. Eventually after the selection of user based font conversion, the target conversion is obtained as follows:

In step 2, If CHOICE == FONT2FONT, then the user is asked to select BASEFONT and TARGETFONT, where the BASEFONT is the basic Gurmukhi font in which the OPENXMLDOCUMENTTEXT is written and TARGETFONT is the font in which user want to convert the BASEFONT. Eventually after the selection of Gurmukhi fonts, the target conversion is obtained as follows:

(a) If BASEFONT == TARGETFONT, then we obtain an ERROR, and begin Font selection again. Else, FONTTOFONTDATABASE is searched for relevant Gurmukhi DICTIONARY for the selected BASEFONT and TARGETFONT.
(b) If DICTIONARY is not found in FONTTOFONTDATABASE, then we obtain an ERROR and algorithm terminates its functionality. Else, after successful search the relevant Gurmukhi font DICTIONARY is assigned to SELECTED_DICTIONARY variable. Now, the parsed OPENXMLDOCUMENTTEXT and SELECTED_DICTIONARY are passed to SUBSTITUTION algorithm to perform substitution.

The SUBSTITUTION algorithm returns the converted text to OUTPUT_TEXT.
In step 3, Else if CHOICE == FONT2UNICODE, the user is asked to select BASEFONT. Again, the FONTTOUNICODEDATABASE is searched for relevant Gurmukhi Unicode DICTIONARY for the selected BASEFONT. After the search is performed the Unicode conversion worked as follows:

(a) If, DICTIONARY is not found FONTTOUNICODEDATABASE, then we obtain an ERROR and algorithm terminates its functionality.

(b) Else, the selected Gurmukhi font Unicode DICTIONARY is assigned to SELECTED_DICTIONARY variable. Regular expressions has been used to handle the rendering problem of Gurmukhi vowel Sign I (ਿ), located at 0A3F code value in [14]. We have used four inbuilt regular expression functions to solve this problem. RE, COMPILE, SUB, LAMBDA and GROUP are four regular expression functions used. For the relevant BASEFONT, Gurmukhi vowel Sign I is selected from the FONTTOUNICODEDATABASE and initiated to I_key_OF_BASE_FONT variable. A regular expression search function is created as:

RegularExpression <-
RE.COMPILE(r'(I_key_OF_BASE_FONT)(\S)')

```
SUBSTITUTED_TEXT(FONT2FONT, FONT2UNICODE, SUBSTITUION_FUNCTION,
FONTTOFONTDATABASE, FONTTOUNICODEDATABASE, OPENXMLDOCUMENTTEXT,OUTPUT_TEXT)
FONT2FONT: Font to Font Convesion
FONT2UNICODE: Font to Unicode Conversion
SUBSTITUION: Gurmukhi text Substitution Algorithm
FONTTOFONTDATABASE: Gurmukhi Font to Font Key mappping Dictionaries
FONTTOUNICODEDATABASE: Gurmukhi Font to Unicode Key mappping Dictionaries
OPENXMLDOCUMENTTEXT: Text extracted by parsing Gurmukhi OpenXML document
OUTPUT_TEXT: Substituted Gurmukhi Text (Output)

1. Set CHOICE <- {Enter user based selection - FONT2FONT or FONT2UNICODE}
2. if CHOICE == FONT2FONT, then:
        Set BASEFONT <- Select Base Font
        Set TARGETFONT <- Select Target Font
                if BASEFONT == TARGETFONT, then:
                        RAISE ERROR
                        Print "Select different Target Font"
                        Go to Step 2
                Else:
                        Search for DICTIONARY in FONTTOFONTDATABASE for
selected Base and Target font
                                If DICTIONARY is not found, then:
                                        RAISE ERROR
                                        Print "Dictionary not Found" and Exit.
                                Else:
                                        SELECTED_DICTIONARY <- Selected
Dictionary
                                        OUTPUT_TEXT <- Set
SUBSTITUTION(OPENXMLDOCUMENTTEXT, SELECTED_DICTIONARY)
                                [End of inner if Statement]
                [End of middle if Statement]
   [End of outer if Statement]
3. Elseif CHOICE == FONT2UNICODE, then:
        Set BASEFONT <- Select Base Font
        Search for DICTIONARY in FONTTOUNICODEDATABASE for selected Base font
                If DICTIONARY is not found, then:
                        RAISE ERROR
                        Print "Dictionary not Found" and Exit.
                Else:
                        SELECTED_DICTIONARY <- Selected Dictionary
                        I_key_OF_BASE_FONT <- Select the Long Vowel I from
selected DICTIONARY
                        RegularExpression <-
RE.COMPILE(r'(I_key_OF_BASE_FONT)(\S)')
                        I_HANDLED_TEXT <- RegularExpression.SUB(LAMBDA(j):
j.GROUP(2)+j.GROUP(1), OPENXMLDOCUMENTTEXT)
                        OUTPUT_TEXT <- set SUBSTITUTION(I_HANDLED_TEXT,
SELECTED_DICTIONARY)
                [End of inner if Statement]
   [End of outer if Statement]
4. Else:
        Print "You entered wrong Choice" and Exit.
5. Exit
```

Fig. 8. Font selection and conversion algorithm

`RE.COMPILE(r'(I_key_OF_BASE_FONT)(\S)')` search for the I (ਿ) key with its following next character.

This regular expression working is divided in two working groups as shown in Fig. 9. Developed regular expression functioning is also demonstrated using finite automata given in Fig. 10. The whole process of searching and swapping the key values is divided in two steps - Glyph Segregation and Glyph Interchange. In the glyph segregation process, I key and the following next character is examined and in glyph interchange phase the keys segregated is swapped with respect to its position as shown in Fig. 11.

(I_key_OF_BASE_FONT)(\S)

- 1st Capturing group (`I_key_OF_BASE_FONT`)
  - `I_key_OF_BASE_FONT` matches the Gurmukhi vowel Sign I (ਿ) of base font (case sensitive).
- 2nd Capturing group (**\S**)
  - \S match any non-white space character [^\r\n\t\f] next to Gurmukhi vowel Sign I.

Fig. 9. Regular expression working group explanation

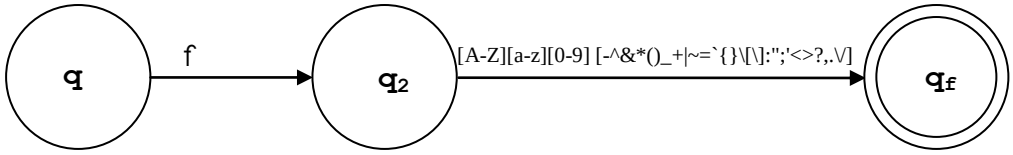

Fig. 10. Finite state representation of regular expression

Glyph interchange process is carried out by using the following function:

```
RegularExpression.SUB(LAMBDA(j):j.GROUP(2
)+j.GROUP(1), OPENXMLDOCUMENTTEXT)
```

This function, `SUB` function is used to substitute the positions of I key and following next character key, which was put on `GROUP(1)` and `GROUP(2)` respectively using `RE.COMPILE(r'(I_key_OF_BASE_FONT)(\S)')` function. The `SUB` function substitute the pattern passed in `RegularExpression` variable, using `LAMBDA()` function with `j` variable, which act like a syntactic sugar for declaring the normal function definitions. It is an iteration function which perform replacement switch using `j.GROUP(2)+j.GROUP(1)` function to concatenate the two characters using position values. This function is performed for each and every pattern matches for I Gurmukhi key and the result is saved to `I_HANDLED_TEXT`, which contains glyph assimilated document text. The substituted `I_HANDLED_TEXT` and `SELECTED_DICTIONARY` are passed to `SUBSTITUTION` algorithm to perform the Unicode substitution for Gurmukhi characters. The `SUBSTITUTION` algorithm again returns the converted text to `OUTPUT_TEXT`.

Else, in Step 4, if the user selects wrong value for `CHOICE` variable, the algorithm shows an error. This step 4 condition signals the `CHOICE` selection as incorrect and program gets terminated in Step 5.

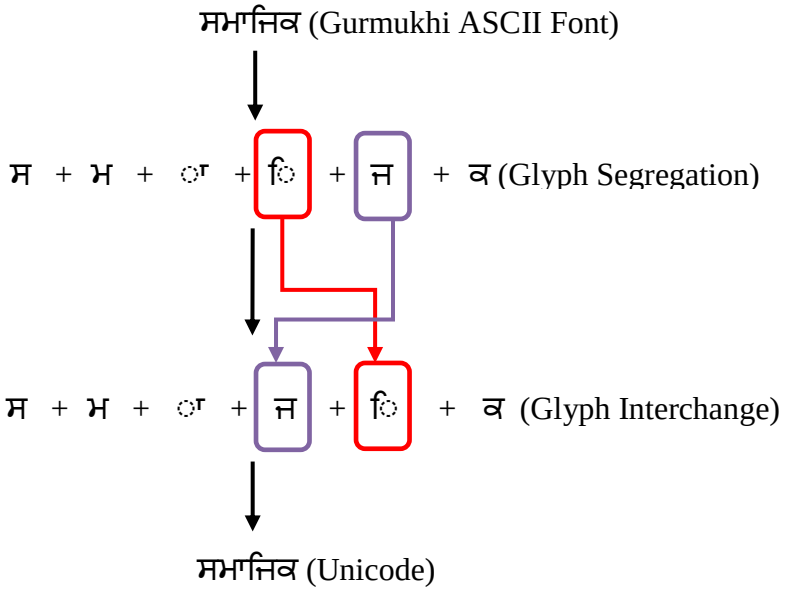

Fig. 11. Regular expression swapping mechanism for Gurmukhi vowel "ਿ"

6. ACCURACY ASSESSMENT CRITERION

We followed some standard set of procedures character level substitution and word level substitution. The set of procedures followed in the assessment are as follows:

6.1 Percentage of Accuracy

In the percentage of Accuracy, the GFUC is analysed for its substitution accuracy mechanism. For this we came up with equation 1. In this equation, Matched Characters or Words are the number of characters or words in Gurmukhi substituted font OpenXML document file, which match with the original Gurmukhi OpenXML document. Whereas, Total Characters or Words represents the total number of characters in the base OpenXML document.

$$\text{Percentage of Accuracy} = \frac{\text{Matched Characters/Words}}{\text{Total Characters/Words}} \times 100 \quad (1)$$

6.2 Overall accuracy

To calculate the Overall accuracy of GFUC application, we have used equation 2 and 3 [15]. These equations comprise to tell the overall accuracy of substitution mechanism installed in GFUC. ω represents the overall accuracy to be computed, Off diagonal components in the matrix is represented by using the NT, Major diagonal component sums up in e_{ii} and total number of columns are represented using nc.

$$\omega = \sum_{i=1}^{nc} e_{ii} / NT \quad (2)$$

$$\text{NT} = \sum_{i=1}^{nc} \sum_{j=1}^{nc} e_{ij} \quad (3)$$

6.3 Kappa Coefficient

We have used Kappa coefficient to measure the inter-rater reliability of the GFUC results. Kappa takes all columns and rows in account while computing the confusion matrix whereas overall accuracy is computed only for the major diagonal [16]. Kappa $\hat{k}$ is used to find the complete trustworthiness of GFUC. Landis and Koch (1977) standard of agreement was used to see the result $\hat{k}$ (≤ 0 = poor, .01-.20 = slight, .21 - .40 = fair, .41 - .60 = moderate, .61 - .80 = substantial, .81 – 1 = almost perfect)[17]. Total number of cells in confusion matrix is represented using N, nc represent the total number of columns in confusion matrix, sum of column i is given using i+, sum of row i is given using +i, and X_{ii} signifies the total number of correct cells in confusion matrix.

$$\hat{k} = N\sum_{i=1}^{nc} X_{ii} - \sum_{i=1}^{nc}(X_{i+} \times X_{+i})/N^2 - \sum_{i=1}^{nc}(X_{i+} \times X_{+i}) \quad (4)$$

7. RESULTS

The GFUC substitution accuracy was analysed on two criteria's – character level and word level. At character level substitution, Gurmukhi characters were analysed for substitution accuracy following the same set of standard procedures described in [3] and established of procedures mentioned in [6] is used to calculate the accuracy at word level.

It is not feasible to calculate the number of correctly substituted characters or words manually as data is large in number. To handle this problem for analysis phase, we develop two small applications to calculate our accuracy at character and word level using the above mention accuracy assessment criterion as shown in Fig. 12 and Fig. 13. This application was deigned on the same configuration mentioned in system design with the use of additional tool known as NLTK.

Fig. 11 shows the application made to record the results of font to font substitution mechanism and Fig. 13 showcases the application made to record the results of font to Unicode substitution of GFUC. These applications contain two phases – Data Arrangement and Analysis phase. In the data arrangement phase of Fig. 12 one OpenXML document which is written using Base font is substituted using GFUC font to font substitution and termed as Target font OpenXML Document File. Now these both files are further sent to GFUC font to Unicode mode to make the written data machine readable in these both files. These files are further sent to analysis phase to identify the performance of our installed mechanism in the form of GFUC. In analysis phase, the files are tokenized and application sees that if characters or words are equal in number. If they are not equal then it displays error otherwise, it is further sent to text matcher. After analysis phase we get our results. The Data Arrangement phase of Fig. 13 is slightly different from Fig. 12. As, it is used to analyse the results of GFUC font to Unicode conversion so, we haven't installed GFUC Unicode conversion for Target OpenXML file as it is initially in the Unicode form. The subsequent steps are similar to Fig. 12.

7.1 Character Level Analysis

To carry out character level assessment for GFUC font to font substitution module, 61 diverse Gurmukhi characters were taken in an OpenXML document and passed through our application designed to perform our experiment as showcased in Fig. 12. The output results of the experiment are displayed in the form of confusion matrix given in Table 2. In this major diagonal contain the number of characters matched during the experiment and off diagonal on both side of major diagonal contain the number of characters not matched in experiment.

Table 2 is analysed on the given accuracy assessment criterion mentioned in Section 6 of this paper. The overall accuracy is calculated using equation 2 and 3. The overall accuracy results are shown in Table 3. Using equation 4 on Table 2, the kappa coefficient was calculated which is represented in Table 4.

Using experimental setup demonstrated in Fig. 13, we started to analyse the GFUC Font to Unicode module by taking the dataset which comprises of total 70 Gurmukhi characters, containing the additive form of Gurmukhi special and long vowels. The result of performed experiment is represented in Table 5. Equation 1 was used to calculate the amount of accuracy for the demonstrated results at character level, which are shown subsequently with correctly recognized characters in Table 5.

7.2 Word Level Analysis

For character level analysis, we took a 5 OpenXML document files containing different amount of words in 5 different fonts. The amount of words taken for analysis each font is shown in Table 6. Same set of procedure was followed as given in Fig. 12 to calculate the accuracy at word level for Font to Font substitution mechanism of GFUC. The files containing different amount of words is passed to Fig. 12 taking account of result at the end of experiment. The results taken down during the course of experiment are shown in Table 7.

Using equation 1, the amount of accuracy was checked for the acquired number of words converted in this module. The accuracy is also given in Table 7.

At the last to know about the accuracy at word level in Unicode module of GFUC, the same set of document file containing different amount of words is given to Fig. 13 application. The results and discovered accuracy is given in Table 8. For this purpose also, equation 1 was used in which the total number of words correctly converted was checked to give the output results.

8. DISCUSSION

The results got after performing the analysis on various parameters are discussed in this section. The results were analysed by making the comparative study which includes, percentage of accuracy, overall accuracy and kappa statistics, which showcase us the overall reliability of legacy substitution mechanism in documents and Gurmukhi Font and Unicode Converter at character level and word level. For Font to Font substitution mechanism of GFUC, overall accuracy in Table 3 that was got after examination comes out to be 100%, which is excellent at character level. The kappa statistics ($\hat{k}$) in Table 4 come out to be 1, which according to the Landis and Koch (1977) standard of agreement is perfect. The Table 5 shows us 100% loss-less font substitution for font to Unicode mechanism of GFUC at character level. Table 7 tells us that about the word level analysis for GFUC Font to Font substitution. It shows us the 100% accuracy at word level for 5 Gurmukhi fonts. Also, Table 8 shows 100% accuracy at Unicode level without any information-loss of Gurmukhi character "I".

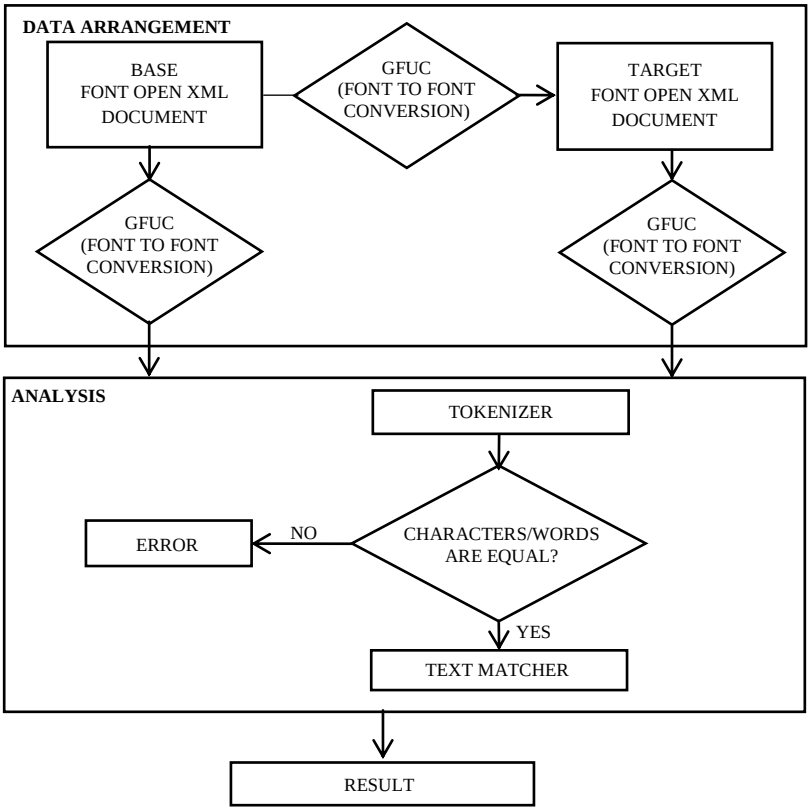

Fig. 4. Application Framework for analysis for Font to Font mechanism of GFUC

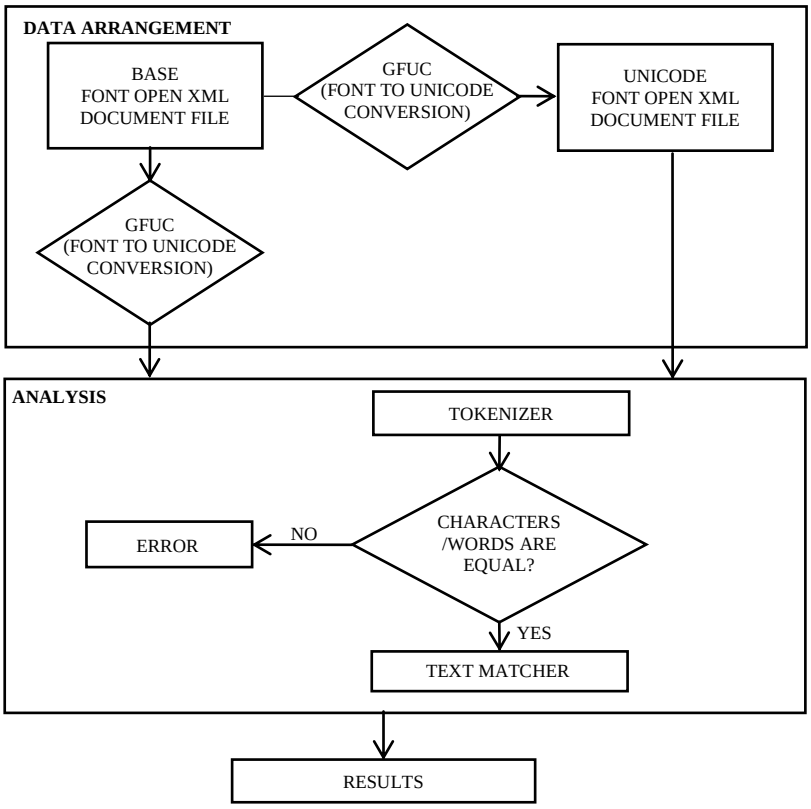

Fig. 13. Application Framework for analysis for Font to Unicode mechanism of GFUC

This result demonstrate us that by using the GFUC, the information loss was handled at bigger scale. The indicators like 100% accuracy at 2 levels of assessment, provides us with base of saying that GFUC is capable of handling the information loss for Gurmukhi script for 5 legacy Gurmukhi fonts and provide us with full document conversion without any knowledge of font key-mappings.

Table 2. GFUC Font to Font Substitution Accuracy Assessment

Base Font	Substituted Font				
	Joy	**Gurbani Akhar**	**Anandpur Sahib**	**Akhar**	**Sukhmani**
Joy	61	0	0	0	0
Gurbani Akhar	0	61	0	0	0
Anandpur Sahib	0	0	61	0	0
Akhar	0	0	0	61	0
Sukhmani	0	0	0	0	61

Table 3. Overall accuracy

Overall Accuracy	Calculated Value
ω	100%

Table 4. Kappa coefficient

Kappa coefficient	Calculated Value
$\hat{k}$	1

Table 5. GFUC Font to Unicode Substitution Accuracy Assessment

Base Font	Unicode		
	Total Character in Base Font	**Characters Correctly Substituted to Unicode**	**Accuracy at Character Level**
Joy	70	70	100%
Gurbani Akhar	70	70	100%
Anandpur Sahib	70	70	100%
Akhar	70	70	100%
Sukhmani	70	70	100%

Table 6. Number of words for testing in 5 diverse Gurmukhi fonts

Gurmukhi Font	Number of Words for Testng
Joy	3488
Anandpur Sahib	4026
Gurbani Akhar	2156
Akhar	2390
Sukhmani	2491

Table 7. GFUC Font to Font Substitution Accuracy Assessment at Word Level

Document File (Base Font - Target Font)	Words	Number of Correctly Converted Words	Accuracy
Joy - Anandpur Sahib	3488	3488	100%
Joy - Gurbani Akhar	3488	3488	100%
Joy - Akhar	3488	3488	100%
Joy - Sukhmani	3488	3488	100%
Anandpur Sahib - Joy	4026	4026	100%
Anandpur Sahib - Gurbani Akhar	4026	4026	100%
Anandpur Sahib - Akhar	4026	4026	100%
Anandpur Sahib - Sukhmani	4026	4026	100%
Gurbani Akhar - Joy	2156	2156	100%
Gurbani Akhar - Anandpur Sahib	2156	2156	100%
Gurbani Akhar - Akhar	2156	2156	100%
Gurbani Akhar - Sukhmani	2156	2156	100%
Akhar - Joy	2390	2390	100%
Akhar - Anandpur Sahib	2390	2390	100%
Akhar - Gurbani Akhar	2390	2390	100%
Akhar - Sukhmani	2390	2390	100%
Sukhmani - Joy	2491	2491	100%
Sukhmani - Anandpur Sahib	2491	2491	100%
Sukhmani - Gurbani Akhar	2491	2491	100%
Sukhmani - Akhar	2491	2491	100%

Table 8. GFUC Font to Unicode Substitution Accuracy Assessment at Word Level

Document File (Base Font - Unicode)	Words	Number of Correctly Converted Words	Accuracy
Joy	3488	3488	100%
Anandpur Sahib	4026	4026	100%
Gurbani Akhar	2156	2156	100%
Akhar	2390	2390	100%
Sukhmani	2491	2491	100%

9. CONCLUSION AND FUTURE WORK

Gurmukhi Font and Unicode Converter (GFUC) is proposed in this research. Non-Standardization is stopping the automation of Gurmukhi scripts in digital domain in digital documents. To handle this information loss GFUC was designed. GFUC is capable of giving inter font and Unicode conversion without rendering problems. Inter-font conversion was made for 5 Gurmukhi fonts and it is capable of giving loss-less font conversion between these Gurmukhi fonts. Also, our GFUC is capable of giving Font to Unicode conversion without any rendering problem in Gurmukhi glyphs. GFUC performance was analysed and checked on various parameters like percentage of Accuracy, Overall Accuracy and Kappa Coefficient matrices. GFUC gives 100% accuracy in both domains of substitution mechanism and handled information loss efficiently.

Gurmukhi font conversion system application is realized in this study. However, this system application can further be extended. In this study, 5 Gurmukhi fonts were taken into consideration to make inter-font conversion application and to design a font to Unicode conversion system. An extension to this research work can be done is following manner:

- Graphical User Interface (GUI) will be created to make the GFUC available to mass users.
- More Gurmukhi fonts can be added to make it useful in wide range.
- Study can also be extended to another available Indian scripts.

10. REFERENCES

[1] "ANSI X3.4: Coded Character Set-7-Bit American National Standard Code for Information Interchange," New York, 1986.

[2] G. Brown and K. Woods, "Born Broken: Fonts and Information Loss in Legacy Digital Documents," *Int. J. Digit. Curation*, vol. 6, no. 1, pp. 5–19, 2011 [Online]. Available: http://www.ijdc.net/index.php/ijdc/article/view/159. [Accessed: 31-Jan-2015]

[3] G. S. Mahi, A. Verma, K. S. Bajwa, and G. Singh, "Information Loss in Digital Documents," in *IEEE international symposium on emerging trends and technologies in libraries and information services*, 2015, pp. 118–121.

[4] "The Unicode Consortium." [Online]. Available: www.unicode.org

[5] A. Bharati, N. Sangal, V. Chaitanya, R. Sangal, and G. U. M. Rao, "Generating Converters between Fonts Semi-automatically," in *SAARC conference on Multi-lingual and Multi-media Information Technology*, 1998.

[6] S. S. Rahaman, M. R. Islam, and M. a. H. Akhand, "Design and development of a Bengali unicode font converter," in *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, 2013, pp. 1–4 [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6572667

[7] R. Arokia, A. Anand, and K. Prahallad, "Identification and Conversion on Font-Data in Indian Languages," in *ICUDL*, 2007.

[8] G. S. Lehal, T. S. Saini, and S. K. Chowdhary, "An Omni-font Gurmukhi to Shahmukhi Transliteration System," in *COLING*, 2012, vol. 3, no. December 2012, pp. 313–320.

[9] G. S. Mahi and A. Verma, "Wrecked Indian Fonts: A Problem for Digitalization of Indic Documents," in *60th International Conference on Embedded Librarianship and Technological challenges in Digital Age*, 2015, pp. 905–914.

[10] "ECMA-376," 2012. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-376.htm

[11] "ISO/IEC 29500-1:2008," 2008. [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=51463

[12] "Textract 1.2.0." [Online]. Available: https://textract.readthedocs.org/en/latest/

[13] "Python Dictionaries." [Online]. Available: www.python-course.eu/python3_dictionaries.php

[14] "Gurmukhi Unicode Standard, Version 6.3," 2013 [Online]. Available: unicode.org/charts/PDF/U0A00.pdf

[15] R. G. Congalton, "A review of assessing the accuracy of classification of remotely sensed data," *Remote Sens. Environ.*, vol. 37, pp. 35–46, 1991.

[16] J. Cohen, "A coefficient of agreement of nominal scales," *Educ. Psychol. Meas.*, vol. 20, pp. 37–46, 1960 [Online]. Available: http://epm.sagepub.com/cgi/doi/10.1177/001316446002000104

[17] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data.," *Biometrics*, vol. 33, pp. 159–174, 1977.