Developing a Vedic Word Processor

Abstract

Our sages have enunciated the knowledge of truth in Chaturdasha Vidyastanas [1]. But several parts of them are lost to antiquity for manifold reasons. The consolidation of such knowledge has been particularly difficult due to lack of appropriate digitization and processing capabilities. Thus, researchers in this field are at a disadvantage in their abilities to access or search authentic data online. This paper illustrates the effort to solve this problem with a specially designed platform-independent Vedic word processor with an easy and efficient way of handling Vedic texts and conjuncts, incorporating the Vdbw-ttswara font for rendering text and using the ISCII standard for processing text at the intermediate level. The first part of the paper illustrates the development of a converter for handling Vedic text. The second part of the paper illustrates interfacing of the converter with the front end. As a result now this software can store, edit and print Vedic text in RTF and ACI formats.

Keywords:

ISCII, ISFOC, Vedic Accents, Vedic characters, ACI , Vdbw-ttswara, DFA, Converter, Keyboard overlay

[1] Fourteen different branches of knowledge in Hinduism

1 Introduction

Purana-nyaya-mimamsa-dharmasastrangamisritah
Vedah sthanani vidyanam dharmasya ca caturdasa

-Yajnavalkyasmrti

Vid means *to know* and is the root of *Vidya*,a work that imparts knowledge or sheds light on the truth. That there are fourteen treatises on *vidya* is indicated by: *vidyanam dharmasya ca caturdasa*. The fourteen are not only sastras that impart knowledge but also treatises on moral principles. They are therefore called *Vidyasthanas* and *Dharmastanas*: *sthanani vidyanam dharmasya ca caturdasa*. This entire knowledge was bequeathed from one generation to another generation from time immemorial through the *Gurushishya* tradition. However, most branches of this knowledge are lost to antiquity and the *Gurushishya* tradition itself is under decline for manifold reasons. Recognizing that this knowledge is very much essential for the realization of the truth at all the times, modern technology is making an attempt to preserve and bequeath this knowledge to future generations.

Some of the popular solutions available today such as iLeap are not platform-independent, while others such as Shree Lipi support only few Vedic accents. Currently, there is no editor is that is both platform independent and supports editing of Vedic text including both Vedic swaras and non-swara characters.

This paper will illustrate the development of an enterprise and platform independent multilingual word processor that supports editing English, Sanskrit and Vedic texts including Vedic swaras and non-swara characters. The 16-bit Vdbw-ttswara is a true type font that complies with the ISFOC standard and is used for rendering text, while the ISCII standard is used for editing and storing text. A converter handles the intermediate level of encoding for text processing at the ISCII level. It converts the text from ASCII to ISCII, ISCII to ISFOC, ISFOC to ISCII and ISFOC to ASCII and ISCII. It should also identify the beginning and the end of ISCII and ISFOC syllables along with support for conjuncts that are unique to Indian languages. Two new keyboards for editing Sanskrit and Vedic texts along with the existing English keyboard are also supported. The front end is minimalist and intuitive, and supports saving and opening files in RTF and ACI forms and printing them. Supporting cut, copy, and paste for Vedic text posed a significant challenge, and these operations had to be redefined.

The first part of the paper illustrates the development of a converter for handling Vedic text. The second part of the paper illustrates interfacing of the converter with the front end by redefining the keyboard actions for handling Sanskrit and Vedic text. Finally we would conclude with a look at possible future enhancements and discuss issues related to the enhancement of the UNICODE standard.

2 Method of Solution

2.1 Developing a Converter

ISCII standard specifies a common alphabet for all Indian scripts because of their common origin from the same ancient Brahmi script. This also includes ASCII characters. ISCII standard is used here mainly for intermediate level of text processing while editing and storing the text edited in the processor. For displaying the text we use 16-bit Vdbw-ttswara font complying with ISFOC standard. Therefore a converter is needed to convert the ISCII text representation into ISFOC text representation. This is used while opening the ISCII file and while displaying the edited contents. Another converter is needed to convert ISFOC text representation into ISCII text representation. This is needed whenever we want to save the file and also during updating the edited contents in the display.

2.1.1 Representing Characters in ISCII code Standard

Let us take a brief look at the English, Sanskrit and Vedic characters and its representation using ISCII standard [2]. ISCII standard already includes ASCII characters along with this I am considering only characters, which are used in Sanskrit and few characters required for handling extended character set. Hence we can categorize them into 7 different types. A-ASCII (also includes all the English alphabets) and rest of the symbols identify Sanskrit characters C-Consonants, V-Vowels, M-Matras, D-Vowel Modifiers, H-Halant, S-Danda.

Print section headings in 12-point bold type in the style shown in these instructions. Leave a blank space of approximately 10 points above section headings. Leave a blank space of approximately 10points below section, subsection, and subsubsection headings. Number sections with arabic numerals. For example, “1 Introduction” is an example of a section heading.

There are certain special symbols[3] such as Om, Avagraha and characters lru, lruu and ruu, which are not included in ISCII standard. Hence they are represented as a combination of an ISCII character and N-Nukta.

There is Q-Vedic non-swaras and R-Vedic swaras, which are an extension to the above character set[4]. These characters are also not included in the ISCII standard. Hence they are represented as a combination of an E-Extension character and an ISCII character i.e. Vedic character = Extension character + ISCII character.

2.1.2 Identifying the Structure within the Syllables

Different characters will combine together to form syllables. We edit the Sanskrit and Vedic text in the processor as syllables. Hence in order to handle them we need to identify the structure within these syllables.

[2]Refer ISCII manual
[3]Refer to ISCII manual
[4]Refer to Extension character set Fig 3 and Fig 4

The Deterministic Finite Automata (DFA)[5] describes the structure within these syllables. In the DFA we start from initial state indicating the beginning of the new syllable and based on the next character type we determine the next state. We identify the end of the syllable based on the combination of the type of current character and the type of next character, which follows it. For example now if the current state is C and if the next character type is halant or matra or vowel modifier or Vedic swara then you move to state H or M or D or Q respectively else for other type of characters you immediately move back to the initial state, which indicates the end of the previous syllable and beginning of the new syllable. Then after moving back to the initial state immediately switch to the next state based on the type of the current character.

2.1.3 Developing ISCII to ISFOC Converter

Now this converter converts ISCII text representation into ISFOC text representation. This conversion mainly depends on the number of glyphs available in the font. Vdbw-ttswara font has more than 150 glyphs having individual glyphs for single characters, combined glyphs to represent different consonant combinations of the conjuncts and also different level representations for matras and Vedic swaras. Handling conjuncts is the most complex part in the development of the converter. Theoretically a conjunct can be made up of more than a single consonant as shown in the DFA but while implementing we have assumed that a conjunct can have at most five consonants.

Consider the entire ISCII text and pass it through DFA in a loop and identify the beginning and end of the syllables based on the type of current character and type of the next character. Convert individual syllables at a time in the text. At every state of DFA we look at the type of the next character within the single syllable and convert the individual characters and finally we have converted the single ISCII syllable into single ISFOC syllable. This is repeated for all the syllables in the text to convert entire text from ISCII to ISFOC representation. Hence we need to look at the conversion of characters at different states of DFA.

Let us start from initial state. Here previous character is null at the beginning of the loop. If the type of the next character is ASCII, or a consonant, or a vowel, or a Vedic non-swara then you need to move to state A, C, V, and R respectively by appending corresponding ISFOC glyph to the output buffer. For all other type of characters you move to O-others state. O-state indicates the improper beginning of the next new syllable. Hence we need a way to indicate that the user has a wrong sequence of characters. We indicate this by prefixing that character with a dotted circular glyph or you can choose any other suitable representation. All the combination of characters, which are other than the above-described sequence in DFA, is a wrong sequence of characters. Example for a wrong sequence of characters is that a matra or a vowel modifier or a halant following an ASCII character.

If you are at state -A irrespective of the type of the next character you immediately move back to the initial state indicating the beginning of the new syllable.

When you are at state-C or at fourth state-C (conjunct C) and if the next character type is H,

[5] Refer to Fig 1

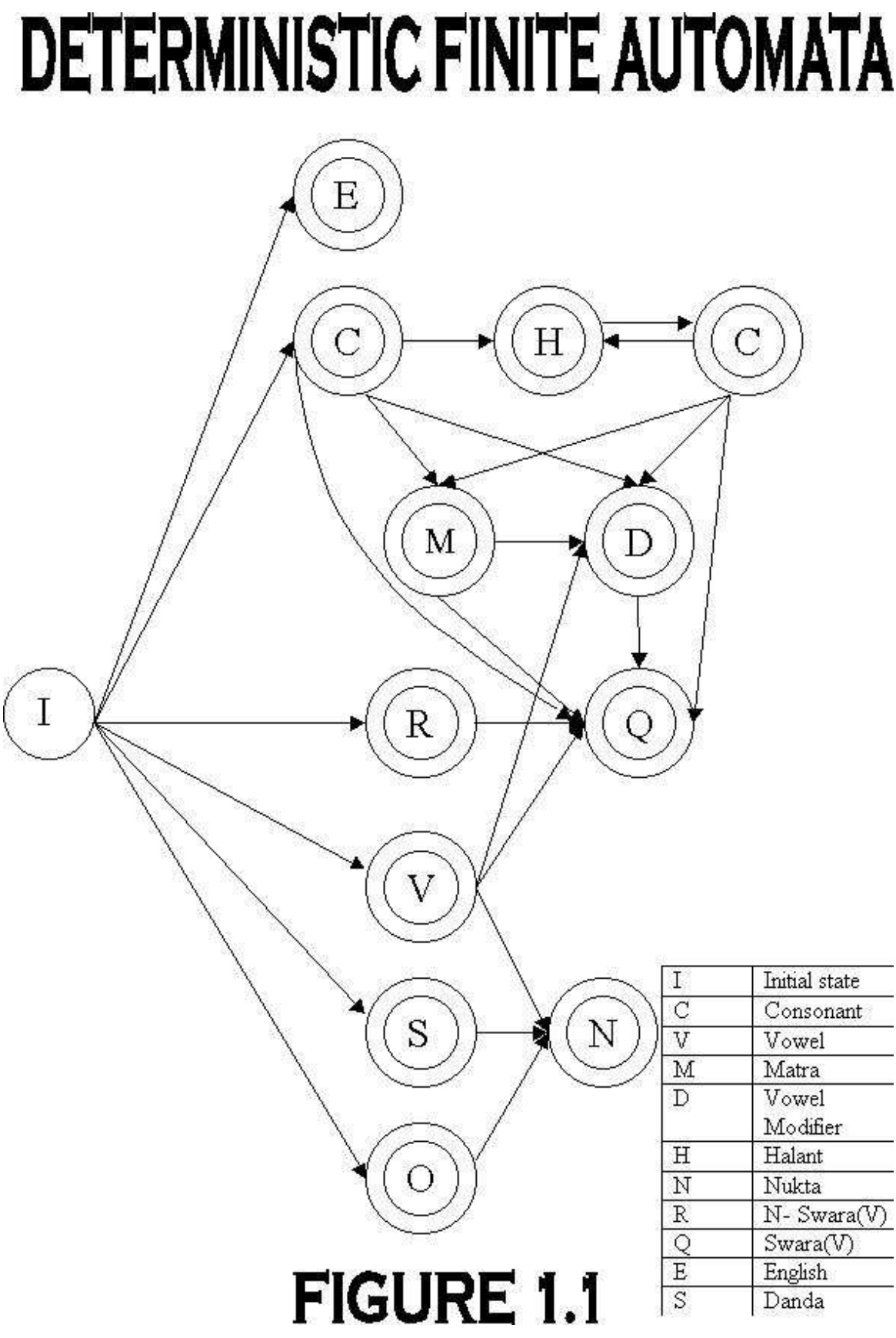


Figure 1: DFA showing structure within a vedic syllable

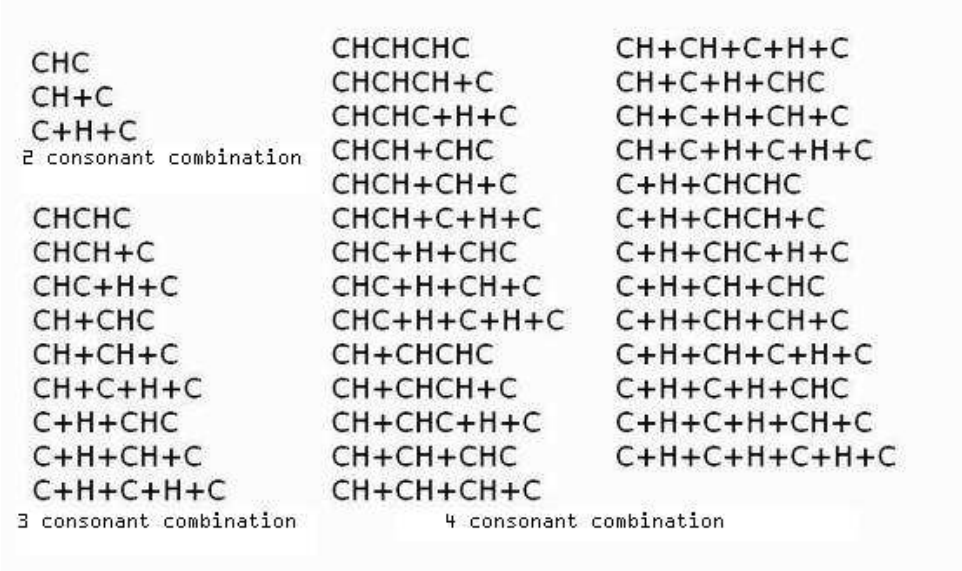


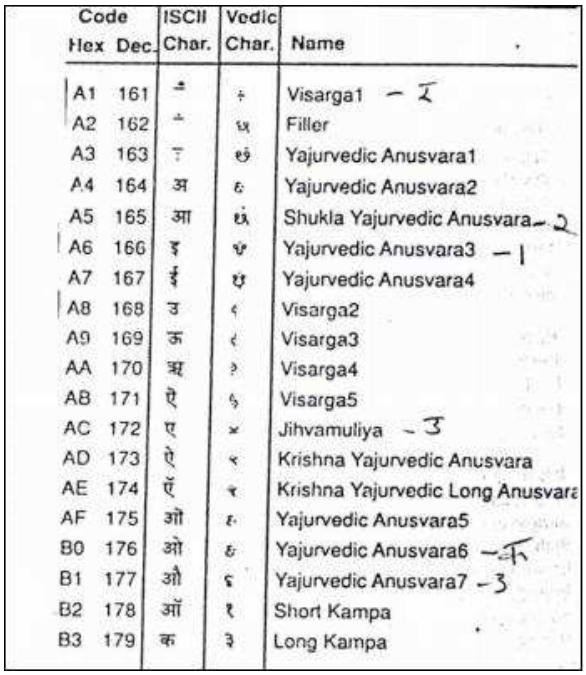
Figure 2: Character combinations in a conjunct

then append the corresponding ISFOC glyph to the output buffer and move to the state-H. At state-H if the next character type is C then you might be in a loop of CHCH of the conjunct. Based on the sequence of consonants and the number of consonants in the conjunct you should find out the appropriate form of display. If the number of consonants is 1 just append the halant and continue. If the number of consonants is 2, 3, 4 then there are 3, 9, and 29 different ways of displaying[6] it respectively. When a conjunct has 5 consonants then you consider the first four consonants find an appropriate form of display among the 29 ways of displaying it then append a halant followed by appending the fifth consonant. When a conjunct has more than 5 consonants then all the consonants following the fifth consonant within the conjunct are ignored as stated in the assumption.

But there are several cases need to be considered while handling conjuncts. First case is when we encounter Ra + halant at the beginning of a conjunct having more than one consonant then and append a repha glyph at the end of the syllable in the output. It is also needed to keep track of the existence of repha in a syllable until the end of that syllable is reached because it is helpful while handling matras as well as conjuncts. Second case is when we encounter halant + Ra within a conjunct then see if it combines with the previous consonant otherwise represent it explicitly as halant + Ra. Third case is when we encounter halant + Ra at the end of a conjunct then see if it combines with the previous consonants else look either it has to be replaced by a special symbol or display them explicitly.

Whenever you encounter a matra following a conjunct or a consonant then find out the type of

[6]Refer to Fig 2



Code Hex	Code Dec	ISCII Char.	Vedic Char.	Name
A1	161	ः	ः	Visarga1 – 1
A2	162	ऀ	ऀ	Filler
A3	163	ँ	ँ	Yajurvedic Anusvara1
A4	164	अ	ः	Yajurvedic Anusvara2
A5	165	आ	ः	Shukla Yajurvedic Anusvara – 2
A6	166	इ	ः	Yajurvedic Anusvara3 – 1
A7	167	ई	ः	Yajurvedic Anusvara4
A8	168	उ	ः	Visarga2
A9	169	ऊ	ः	Visarga3
AA	170	ऋ	ः	Visarga4
AB	171	ऐ	ः	Visarga5
AC	172	ए	×	Jihvamuliya – 3
AD	173	ऐ	ः	Krishna Yajurvedic Anusvara
AE	174	ऍ	ः	Krishna Yajurvedic Long Anusvara
AF	175	ओ	ः	Yajurvedic Anusvara5
B0	176	ओ	ः	Yajurvedic Anusvara6 – 4
B1	177	औ	ः	Yajurvedic Anusvara7 – 3
B2	178	ऑ	ः	Short Kampa
B3	179	क	३	Long Kampa

Figure 3: List of Vedic characters

matra whether it is top, left, right, or a bottom matra. If it is top, or bottom or right matra and if you have repha in the syllable insert the matra before repha else you append the matra at the end of the syllable. If left matra then, insert the matra at the beginning of the syllable. Finally move to the state-M.

Whenever you encounter a Vedic swara following a conjunct or consonant or a Sanskrit syllable then you find out the type of Vedic swara whether it is top, bottom, or right Vedic swara. If it is right Vedic swara then append the corresponding ISFOC glyph to the output buffer. If it is of type top or bottom then find out the number of consonants in the conjunct varying from one to five and also if matra is present in syllable find out the type of the matra. Based on the number of consonants in the conjunct and the type of the matra find the appropriate heightened glyph and append it the end of output buffer. Finally move to the state-Q.

When you are at state-M then if you encounter a vowel modifier or Vedic swara following the matra then you move to the state D, or Q respectively. Handling Vedic swara has already been explained in previous paragraphs. When you have encountered a vowel modifier then based on the type of matra and the existence of repha and the vowel modifier you can replace all these individual glyphs by still optimized glyphs, which combines all these glyphs into a single glyph.

When you are at state-D then if you encounter a Vedic swara then handle it appropriately as discussed above and move to state-Q.

		SWAR		
B4	180	ঀ	ঁ	Jatya Svarita-Atharvaveda
B5	181	ঁ	'	Svarita
B6	182	ং	"	Long Svarita
B7	183	ঃ	‸	Kampa
B8	184	঄	-	Anudatta
B9	185	অ	‿	Jatya Svarita (Shukla Yajurveda)
BA	186	আ	‿	Jatya Svarita (Maitrayaniya)
BB	187	ই	..	Sentence ending Udatta
BC	188	ঈ	‿	Jatya Svarita
				(Non-Taittiriya Yajurveda)
BD	189	উ	.	Svarita (Maitrayaniya)
BE	190	ঊ	,	Anudatta (Kathaka)
		MISCELLANEOUS		
BF	191	ঋ	ঃ	Abbreviation sign

Figure 4: List of Vedic accents

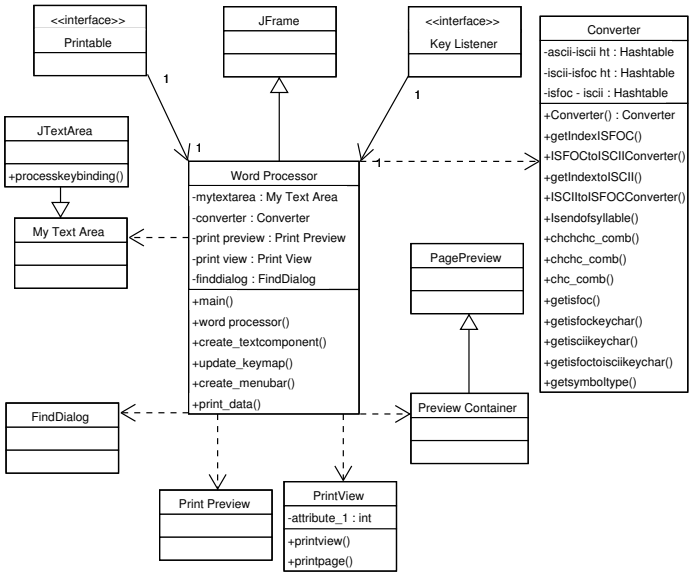


Figure 5: Design of Vedic word processor

When you are at state-R if you encounter a Vedic swara following it then find the corresponding single level ISFOC glyph and append it to the output buffer and move to state-Q.

When you are at state-V if you encounter a vowel modifier following it then find the corresponding ISFOC glyph and append them to the output buffer and move to state-D. If you encounter a Vedic-swara following vowel find the single level ISFOC glyph and append it to the output buffer and move to state-Q. If you encounter nukta following vowel only for vowels e, ee and ru you combine with nukta to replace them with special character glyphs and move to state N otherwise consider nukta as the beginning of a new syllable and move to the initial state immediately.

When you are at state-S it may at the end of syllable or if nukta follows danda you replace them by avagraha and move to state-N.

In O-state when you encounter combination of wrong chandrabindu followed by nukta replace them by Om else you treat nukta as the beginning of the new syllable. If you have wrong e, ee and ru matra following nukta then you replace them by lru, lruu, ruu respectively otherwise nukta is considered as beginning of the new syllable.

2.1.4 Developing ISFOC to ISCII Converter

This converter will convert the ISFOC text representation into ISCII text representation. A single ISFOC glyph can represent more than a single ISCII character hence consider single ISFOC glyph at a time in the loop and convert it into corresponding ISCII characters. Use temporary buffers in the loop for maintaining the single complete ISFOC syllable and corresponding ISCII syllable hence finally when you reach the end of the ISFOC syllable after conversion clear these buffers and again fill them with next new syllable. In every pass of the loop pass the ISCII characters in the temporary buffer into the DFA and evaluate two different parameters EOS-end of syllable and NOS-number of syllables. At every pass in the DFA loop find out whether you have reached the end of a syllable inside DFA and it will be stored in a Boolean variable as EOS and when you have reached the end of syllable increment NOS count.

At every pass in the loop check for the condition ((NOS==1 & EOS==false) OR (NOS==2 & EOS==true) OR (NOS==2 & EOS==false)) and if it is satisfied then you have reached the end of an ISFOC syllable. By the time you have reached the end of syllable you would have also surpassed a glyph ahead in the next new ISFOC syllable. At this stage check for E-matra and repha switches and insert the characters in appropriate positions in the output ISCII syllable. And finally move back a single ISFOC glyph to stand at the beginning of the next new syllable. Clear both the temporary buffers and store again the next new syllables in the loop. Else if previous condition is not satisfied, then if this is the last glyph of the loop, then perform all operations as defined under previous condition except moving back to the previous glyph. Else if neither of them are satisfied then you have not yet reached the end of the syllable and continue the loop considering the next glyph.

But there are three special cases, which need to be considered. First case is because there is no ISCII character representation for circular glyph, second case is the appearance of e-matra is at the beginning of the syllable in ISFOC representation but DFA shows that in ISCII representa-

tion of a syllable all types of matras should come after conjunct or a consonant, and third case is the appearance of repha at the end of the conjunct in ISFOC representation but DFA shows it should be stored as Ra + halant in ISCII representation at the beginning of the ISCII syllable. Hence we need to handle these three cases at the beginning of the loop before going through the steps indicated above inside the loop. Below, three cases have to be checked in the same order as described, at the beginning of the loop.

2.2 Integrating the Front end with Converter

Here we redefine some of the keyboard and mouse operations of the front end with the help of the converter hence integrate the front end with the converter.

The solution for developing front end with minimal features is easy and can be found in any book[7], which introduces Java Swing. We need to support two different types of file storage. One is "ACI" i.e. in ISCII format and another one is "RTF". While storing the file in ACI type we convert the entire contents of text area, which is in ISFOC to ISCII using ISFOC TO ISCII converter and store ISCII contents in a file, but in RTF file storage type ISFOC glyphs are stored as it is. When the user chooses to open ACI file then the file contents is accessed and entire ISCII contents is converted to ISFOC using ISCII to ISFOC converter and the contents are displayed on the text area, but if the user chooses to open an RTF file then we set the file ISFOC contents as it is to the text area.

All the editing operations are localized around current cursor position in the text area of the editor. Hence some amount of ISFOC text surrounding the current cursor position needs to be considered before performing any operations. This can be called "Range of Text". This will be usually one more word towards left from the current word and whose beginning position is the final left position and one more word towards right from the current word whose end position is the final right position. Hence the range of text starts from final left position up to final right position.

It is the fundamental assumption in the editor that the cursor should always be placed at the end of the syllable i.e. cursor should never stand anywhere in between the syllable. Hence in order to set the cursor positions to appropriate position first get the current cursor position. Find the appropriate range of text. Generate an array of values identifying where exactly syllables begin and end in the range of text. Move the cursor to the next highest end of syllable position if it is not already at the end of the syllable.

2.2.1 Defining Keyboards

Enable Sanskrit keyboard and Vedic keyboard using keyboard overlays[8] with appropriate switches. When CAPS LOCK is ON Sanskrit keyboard should be enabled and when both CAPS LOCK and NUM LOCK are ON Vedic keyboard should be enabled. Implementing keyboards are done in

[7] Refer to the books cited in the reference section

[8] Due to lack of space both keyboard overlays are not included here please Refer to ISCII manual

Java using Key listener interface and Keystroke class by using hash tables. Required action to be taken for different keyboards is defined in the below sections. Use hash tables for easy mapping of ASCII characters to ISCII characters.

2.2.2 Editing operations using ISCII text

It is understood that the length of the ISCII syllable and the corresponding length of the ISFOC syllable vary. Hence few editing operations such as editing new text, backspace, delete, cut, copy, paste operations does not work appropriately at ISFOC level where we deal with only glyphs but not with single characters. Hence these operations need to be done at ISCII level. Challenging part of the programming is that even though some of the editing operations are done at ISCII level it should appear to the user as though that he is doing at the ISFOC level. For example whenever user types a character or a backspace or delete key all the processing on text is done at the ISCII level by consuming the keys default action and we perform the actions newly for ISCII text then final ISFOC text will be displayed on the screen. While changing a single syllable the size of the syllable in the form of glyphs might decrease or increase or might remain the same. Hence the cursor needs to be kept track in order to place it at the end of appropriate syllable.

Whenever we want to type a new syllable or add some characters within existing syllables they should be inserted at the end of the current ISCII syllable when the key is pressed. Whenever delete key is pressed it deletes a single ISCII syllable at once rather than deleting a single glyph. Whenever the backspace key is pressed it deletes a last single character in the previous ISCII syllable rather than deleting a single glyph. For all the editing operations cursor should always stand at the end of the ISCII syllable, which is finally reflected at display in ISFOC text.

So when a key is typed then, get the current cursor position. Find the range of text. Within the range of text find out which nth ISFOC syllable we are trying to edit. Convert the range of the editing text into ISCII text.

If a new character is typed then insert the ISCII equivalent of the key typed at the end of the corresponding nth ISCII syllable. If delete key is pressed we should delete the nth syllable in ISCII text. Find the beginning and end of positions of the nth syllable in the ISCII text and delete only those characters from beginning to end. If backspace key is pressed find the beginning and end positions of corresponding nth syllable in ISCII text and delete the last character of the nth syllable in ISCII text.

Now convert the modified ISCII text to ISFOC text. Replace the old span of the editing ISFOC text with the new ISFOC text. Find out where the cursor in ISCII text should be standing because due to the changes in the text some syllables surrounding the modified text can get combined and it is possible that currently cursor stands in between the ISCII syllable. Hence find the new cursor position in ISCII text to stand at the end of the syllable by moving the current cursor position near to the end of the next higher end position. Now find out at the end of which nth ISCII syllable cursor is standing. Now in the ISFOC text place the cursor in the text area at the end of corresponding nth ISFOC syllable.

The above delete and backspace operations when applied to English and Sanskrit text only single

ISCII key corresponding to those characters needs to be deleted but while handling Vedic text careful to delete both the extension key and ISCII character to remove a single Vedic character. Similarly while inserting a Vedic character insert the extension character before ISCII character corresponding to Vedic character.

The above concept of delete and typing new text can be extended to perform cut, copy and paste operations for individual syllables as well as to a block of text.

2.2.3 Editing operations using ISFOC

Whenever we perform any one of the following editing operations page up, page down, up cursor movement, down cursor movement, mouse click then it is required to immediately alter the cursor position so that it always stands at the end of the syllable.

Defining the Left, Right arrow movements, which moves back, or forward a single ISCII syllable (same as single ISFOC syllable) at one press rather than a single glyph. So when you press either a left or right arrow button immediately get the current cursor position. Find out the range of text. Find out the positions of the end of syllables in the range of text and place them in an array. Then move the current cursor position to next higher cursor position in the array so that it moves a syllable ahead for right cursor movement. Similarly for the left cursor movements find next small position in the index obtained the set the cursor to that position so that you have moved a single syllable left.

We can select a block of text in editors, which is usually done by selecting individual glyphs at a time. But here we should be selecting a complete syllable at a time rather than selecting individual glyphs, which can be extended to both the directions for any amount of text. The above concept of right cursor movement and left cursor movement can be extended to selecting block of text in both the directions.

3 Results

Now we can convert any ACI file contents from ISCII to ISFOC and vice versa. Now if CAPSLOCK IS OFF English keyboard is enabled, CAPSLOCK IS ON Sanskrit keyboard will be enabled and CAPSLOCK IS ON & NUM LOCK IS ON Vedic keyboard is enabled. Front end will support the editing of English, Sanskrit, and Vedic text, with Sanskrit conjunct having at most 5 consonants. You can store the files contents in ACI or RTF file formats. Any ACI or RTF files can be opened in the editor. The ACI and RTF files you have created using any of the popular software such as iLeap can be opened and modified in this editor and you can also open and edit files created in this editor in any other editors supporting ACI and RTF file formats. You can easily print the document contents with previewing the document contents before printing. Front-end snapshots vedictext.aci file contents [9] has been included and the high level design [10]of Vedic word processor has been shown.

[9]Refer to Fig 6
[10]Refer to Fig 5

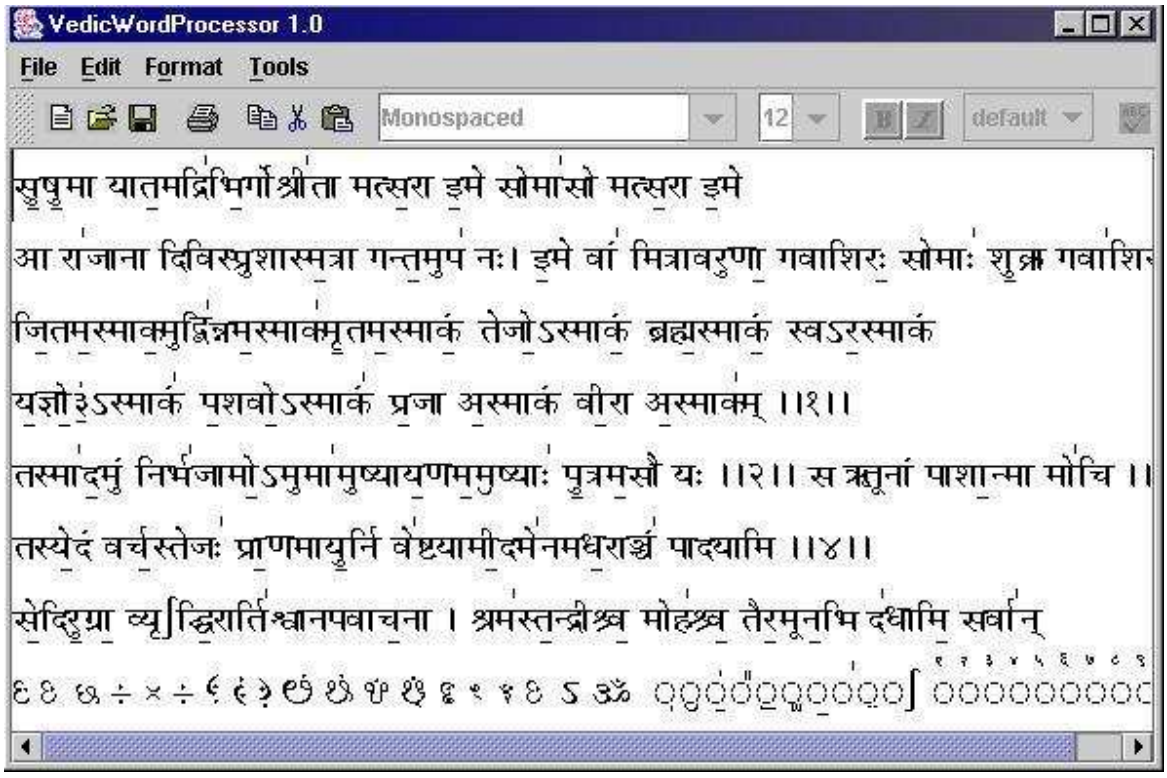


Figure 6: output

4 Conclusion and Future work

Hence the paper methodically illustrates the development of a Platform Independent Multilingual Vedic Word Processor with easy and efficient way of handling Vedic text and conjuncts.

C-DAC has developed a separate standard for handling Vedic texts called PCI standard. We know that Java supports UNICODE standard. UNICODE standard supports only ISCII standard but not PCI standard. Characters from 128 to 161 of PCI are not properly represented in the UNICODE standard. It is interpreting these characters as a 16-bit value instead of interpreting it as an 8-bit value. This is a serious bottleneck for handling Vedic texts, which are stored in PCI format. On the other hand UNICODE standard itself does not have Vedic characters included in it. Hence UNICODE neither supports PCI nor have included Vedic characters. Due to this we have represented Vedic characters as a combination of two ISCII characters. Hence UNICODE should either support PCI standard or it should extend itself by including Vedic characters for easy Vedic text handling.

We have supported the editing of a Sanskrit conjunct having at most five consonants. Even

though no conjunct in Sanskrit texts so far has been found having more than 5 consonants it can be viewed as a limitation while editing the text from software perspective.

Currently implemented ACI file format does not support text style such as bold, italic. But RTF file format supports the above styles. We can easily extend the converter for supporting bold, italic with ATR character in ISCII code standard. This also does not support changing the font only a single font Vdbw-ttswara is supported.

The method of solution is so flexible that you can easily extend the software with additional features such as changing fonts, formatting paragraphs and changing text style. Supporting additional utilities such as Dictionary, Spell checker, exporting and importing files enhances the software product.

References

[Horton, 1998] Ivor Horton (author). *Beginning Java 2*. Wrox Publications 1998.

[Eckstein, Loy, Wood, 1998] Robert Eckstein, Marc Loy, and Dave Wood (1998)(authors). *Java Swing*. O'Reilly Publications 1998.

[Gary, Gary, 1998] David M. Gary, and David Gary (1998) (authors). *Graphic Java2, Vol 2:Swing*. Prentice Hall Publication.

[C-DAC, 1985] Center for Development of Advanced Computing (1985) (author). *ISCII manual*.