Encoding script-specific writing rules based on the Unicode character set

Malek Boualem, Mark Leisher, Bill Ogden

Computing Research Laboratory (CRL), New Mexico State University,
Box 30001, Dept 3CRL, Las Cruces, NM 88003, USA
E-mail: {malek,mleisher,ogden}@crl.nmsu.edu
http://crl.nmsu.edu

The World Wide Web is now the primary means for information interchange that is mainly represented in textual format. However programs that create and view these texts generally do not adequately support texts using non-Latin scripts, particularly right-to-left scripts. Unicode as a universal character set solves encoding problems of multilingual texts. It provides abstract character codes but does not offer methods for rendering text on screen or paper. An abstract character such "ARABIC LETTER BEH" which has the U+0628 code value can have different visual representations (called shapes or glyphs) on screen or paper, depending on context. Different scripts contained in Unicode can have different rules for rendering glyphs, composite characters, ligatures, and other script-specific features. In this paper we present a general approach to encoding script-specific rendering rules based on the Unicode character set and using finite state transducers. The proposed formalism for character classification and writing rules is modular and easy to read and to modify by users. The associated program is written in JAVA, which makes it portable to many environments. This approach will be demonstrated with writing rules for some languages that use the Arabic script and a short example that renders certain Hindi words.

I. Introduction

The Unicode Standard does not define character glyphs. It defines how characters are interpreted, not how glyphs are rendered on the screen. Context analysis is necessary for many scripts to present the correctly shaped and combined glyphs. The context analysis portions of many programs are based on fonts containing glyphs arranged in an order specific to the application or platform. This makes the application much less portable because fonts then have to be distributed along with the program.

The objective of this work is to present a model for context analysis that is at the same time, modular, accessible by users, and uses Unicode. This work on the MUTTCA (Multilingual Unicode Text Toolkit Context Analyzer) was undertaken to improve the context analysis module of the MUTT [MUTT.98] rendering engine and the work was done in Java to make it more portable across platforms. The design of this context analyzer was inspired by the context analyzer used in the MtScript[1], the Multext [MULT.97] multilingual text editor [BOUA.97].

[1] MtScript 1.1 for Solaris and Linux systems is freely available for download at:
http://www.lpl.univ-aix.fr/projects/multext/MtScript/

II. Current context analyzers

Clearly all programs that render text in scripts that are considered complex in some way use some form of context analysis to render the text. The following are some examples and their weaknesses.

Yudit[2] is a freely available Unicode text editor that has a very simple context analysis model. It works on the primary assumption that character codes are the same thing as glyph codes and there is basically a one-to-one correspondence between character and glyph.

MtScript has a context analysis system is well adapted for combining Latin letters with accents and shaping for Arabic. The language-specific context analysis files are loaded at runtime. However, MtScript does not deal with Unicode and the context analyzer model is not really suited for more complicated scripts.

MUTT currently uses hard-coded context analysis rules. This is a problem because all changes and additions for context analysis require recompilation of the software, which implies the hard-coded glyph codes.

The arabjoin[3] Perl script written by Roman Czyborra is a simple and effective Arabic context analyzer. This script is an elegant and easy solution for users familiar with Perl and who do not mind modifying the script for the different fonts that might be encountered.

III. MUTTCA architecture

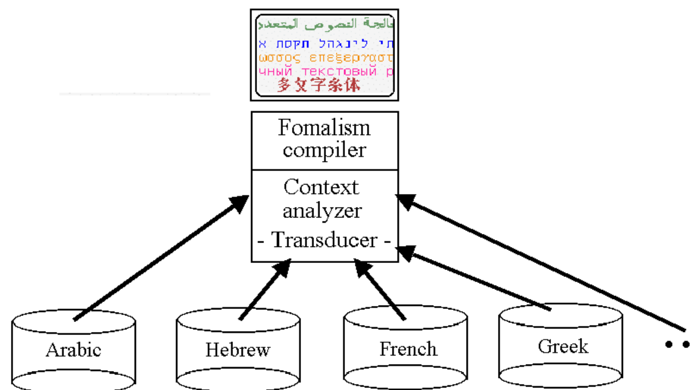


Figure 1: C*ontext analyzer architecture*.

Each language is represented by a formalism consisting primarily of five parts, the header, the numeric values, the mapping tables, the character class specification, and the language-specific writing rules, illustrated as follows.

[2] ***Yudit*** is available in source and binary form from ***ftp://sunsite.unc.edu/pub/Linux/apps/editors/X/***
[3] The ***arabjoin*** script is available on the Web from ***http://czyborra.com/***

```
# COMMENTS
SCRIPT <Script Name>
LANGUAGE <Language Name>
VARIANT <Variant Name>

BEGIN VALUES
 <Name> = U+NNNN                     /* comments */
 <Name> = U+NNNN                     /* comments */
  ...
END VALUES

BEGIN MAPPING
 U+NNNN:U+NNNN
 U+NNNN:U+NNNN
  ...
END MAPPING

BEGIN CLASSES
 C1={U+NNNN,U+NNNN,U+NNNN,...}       /* comments */
 C2={U+NNNN,U+NNNN,U+NNNN,...}       /* comments */
  ...
END CLASSES

BEGIN RULES
 C1,C2,..,Cn:p,"a,b,..,z";      /* comments */
 C1,C2,..,Cn:p,"a,b,..,z";      /* comments */
  ...
END RULES
```

3.1 Header

The header consists of three lines that provide information about the context analyzer. The `SCRIPT` tag provides a script name, the `LANGUAGE` tag provides a language name, and the `VARIANT` tag distinguishes this context analyzer from others for the same script and language.

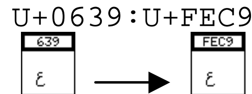
3.2 Numeric values

Because certain arithmetic and replacement operations are allowed in the rules, it is sometimes useful to associate names with numeric values. Names for numeric values are required to be longer than two characters to avoid conflict with variables used in rules. Names make the use of the value clearer as demonstrated in this example from the sample Devanagari context analyzer (cf. Section 4.2):

```
BEGIN VALUES
sub_ra = U+E07E         /* Subjoined RA location in this font.   */
repha = U+E07F          /* REPHA location in this font.          */
half = U+100            /* Half-consonant offset for this font.  */
dependent = U+100       /* Dependent vowel offset for this font. */
END VALUES
```

3.3 Mapping tables

The mapping tables primarily specify the mapping from character codes to their corresponding glyph codes. This is done under the assumption that the glyph codes are arranged so that the different shapes needed can be accessed with a simple arithmetic operation. Character classes are therefore expressed as sets of glyph codes. The following shows the character to glyph mapping for the ARABIC LETTER AIN:



3.4 Character classes

Character classes are sets of characters that have some property in common for the purposes of writing a language. For example, Arabic characters can be classified in a few sets for purposes of rendering [UNST.96]: class of dual-linking characters, class of right-linking characters, class of non-linking characters, class of digits, etc.

3.5 Language-specific writing rules

The language-specific rules are specified as a group of three fields on one line: the left-hand side, or pattern, the edit factor, and the right-hand side, or rewrite rule.

- **The left-hand side**

  This side consists of a comma-separated list of character classes that represent a pattern in the input text. When this pattern is matched, the associated rule is evaluated. The first character class specified in this list represents the last glyph code added to the output string. This field is terminated with a colon.

- **The edit factor**

  The edit factor is 0 or a positive integer value that specifies how many glyph codes have to be removed before the current position in the output string prior to applying **the right-hand side** rules. A value of 0 means no edit occurs. This field is terminated with a comma.

- **The right-hand side**

  This side consists of the rewrite rule in which each letter, **a..z**, represents the character code that matched the class in the same *position* on the **left-hand side**. The order in which the letters are specified on the right-hand side is important. The order of the letters effectively rewrites the text in another order, which is necessary for some scripts. This symbolic character can additionally participate in simple arithmetic operations because it represents the numeric code of the character matched. The right-hand side is terminated by a semi-colon.

Here are two rules taken from the listings near the end of this paper to provide examples of how MUTTCA rules work.

In this example, the `C4` class of characters represents the *word-final* forms of the Arabic letters and the `C1` class represents the *isolated* form of the Arabic letters (cf. Section 4.1.1).

```
C4,C1:1,"a+2,b+1";
```

What this example says is that if the last glyph code is a *word-final* form which is followed by an *isolated* form, replace the *word-final* form represented by `C4` with a *word-medial* form (offset + 2) and change the *isolated* to a *word-final* form (offset + 1) before adding it to the output string. The simple arithmetic operations are based on knowledge that the different Arabic letter shapes are within a constant offset from their other shapes in the font being used.

In the next example, the `C2` class of characters represents the Hindi semi-vowel RA, the `C5` class represents the VIRAMA, the `C1` class represents the consonants, and the `C4` class represents the vowel I. The two symbolic names used on the right-hand side represent constant values.

```
C2,C5,C1,C4:1,"d+dependent,c,repha";
```

What this example says is that if the last glyph code is a RA followed by a VIRAMA, a consonant, and the vowel I, delete the last glyph code in the output string, change the vowel to the dependent form and add it to the output string, add the consonant to the output string, and finally, add a special form of RA to the output string. This example demonstrates that glyphs can be reordered by right-hand side and also demonstrates the use of named values.

IV. Examples

4.1 Example context analyzer for rendering Standard Arabic letters from Unicode

The Arabic script block in Unicode contains not only the letters needed for Standard Arabic but also most of the letters needed for other languages that use the Arabic script. For various reasons, the contextual forms and ligatures for some of the Arabic letters were encoded in Unicode in two blocks of presentation forms. For example purposes, only the letters used for Standard Arabic from the presentation area Arabic characters will be used for illustration of a context analyzer for rendering. Figure 2 below shows the blocks of Unicode to be used.

To render letters that are not in the set used for Standard Arabic, the glyph codes for the font being used would be needed to construct a context analyzer for rendering with that font. The reason for this is that the positioning of Arabic contextual forms in fonts is not standardized. A short discussion about Arabic consonants and vowels is presented next.

600	601	602	603	604	605	606	607	608	609	60A	60B	60C	60D	60E	60F
												،			
610	611	612	613	614	615	616	617	618	619	61A	61B	61C	61D	61E	61F
											؛				؟
620	621	622	623	624	625	626	627	628	629	62A	62B	62C	62D	62E	62F
	ء	آ	أ	ؤ	إ	ئ	ا	ب	ة	ت	ث	ج	ح	خ	د
630	631	632	633	634	635	636	637	638	639	63A	63B	63C	63D	63E	63F
ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ					
640	641	642	643	644	645	646	647	648	649	64A	64B	64C	64D	64E	64F
ـ	ف	ق	ك	ل	م	ن	ه	و	ى	ي	ً	ٌ	ٍ	َ	ُ
650	651	652	653	654	655	656	657	658	659	65A	65B	65C	65D	65E	65F
ِ	ّ	ْ													
660	661	662	663	664	665	666	667	668	669	66A	66B	66C	66D	66E	66F
٠	١	٢	٣	٤	٥	٦	٧	٨	٩	٪	٫	٬	٭		
670	671	672	673	674	675	676	677	678	679	67A	67B	67C	67D	67E	67F
ٰ	ٱ	ٲ	ٳ	ٴ	ٵ	ٶ	ٷ	ٸ	ٹ	ٺ	ٻ	ټ	ٽ	پ	ٿ

FE80	FE81	FE82	FE83	FE84	FE85	FE86	FE87	FE88	FE89	FE8A	FE8B	FE8C	FE8D	FE8E	FE8F
ﺀ	ﺁ	ﺂ	ﺃ	ﺄ	ﺅ	ﺆ	ﺇ	ﺈ	ﺉ	ﺊ	ﺋ	ﺌ	ﺍ	ﺎ	ﺏ
FE90	FE91	FE92	FE93	FE94	FE95	FE96	FE97	FE98	FE99	FE9A	FE9B	FE9C	FE9D	FE9E	FE9F
ﺐ	ﺑ	ﺒ	ﺓ	ﺔ	ﺕ	ﺖ	ﺗ	ﺘ	ﺙ	ﺚ	ﺛ	ﺜ	ﺝ	ﺞ	ﺟ
FEA0	FEA1	FEA2	FEA3	FEA4	FEA5	FEA6	FEA7	FEA8	FEA9	FEAA	FEAB	FEAC	FEAD	FEAE	FEAF
ﺠ	ﺡ	ﺢ	ﺣ	ﺤ	ﺥ	ﺦ	ﺧ	ﺨ	ﺩ	ﺪ	ﺫ	ﺬ	ﺭ	ﺮ	ﺯ
FEB0	FEB1	FEB2	FEB3	FEB4	FEB5	FEB6	FEB7	FEB8	FEB9	FEBA	FEBB	FEBC	FEBD	FEBE	FEBF
ﺰ	ﺱ	ﺲ	ﺳ	ﺴ	ﺵ	ﺶ	ﺷ	ﺸ	ﺹ	ﺺ	ﺻ	ﺼ	ﺽ	ﺾ	ﺿ
FEC0	FEC1	FEC2	FEC3	FEC4	FEC5	FEC6	FEC7	FEC8	FEC9	FECA	FECB	FECC	FECD	FECE	FECF
ﻀ	ﻁ	ﻂ	ﻃ	ﻄ	ﻅ	ﻆ	ﻇ	ﻈ	ﻉ	ﻊ	ﻋ	ﻌ	ﻍ	ﻎ	ﻏ
FED0	FED1	FED2	FED3	FED4	FED5	FED6	FED7	FED8	FED9	FEDA	FEDB	FEDC	FEDD	FEDE	FEDF
ﻐ	ﻑ	ﻒ	ﻓ	ﻔ	ﻕ	ﻖ	ﻗ	ﻘ	ﻙ	ﻚ	ﻛ	ﻜ	ﻝ	ﻞ	ﻟ
FEE0	FEE1	FEE2	FEE3	FEE4	FEE5	FEE6	FEE7	FEE8	FEE9	FEEA	FEEB	FEEC	FEED	FEEE	FEEF
ﻠ	ﻡ	ﻢ	ﻣ	ﻤ	ﻥ	ﻦ	ﻧ	ﻨ	ﻩ	ﻪ	ﻫ	ﻬ	ﻭ	ﻮ	ﻯ
FEF0	FEF1	FEF2	FEF3	FEF4	FEF5	FEF6	FEF7	FEF8	FEF9	FEFA	FEFB	FEFC	FEFD	FEFE	FEFF
ﻰ	ﻱ	ﻲ	ﻳ	ﻴ	ﻵ	ﻶ	ﻷ	ﻸ	ﻹ	ﻺ	ﻻ	ﻼ			

Figure 2: *Arabic blocks of Unicode used for example.*

4.1.1 Arabic consonants

Arabic consonants make up most of the Arabic text seen every day. Vowels are optional and most often determined from context by the reader. Because Arabic is written in a cursive form, each consonant can have multiple shapes depending where the letter occurs in a word. With few exceptions, Arabic consonants have either two or four different shapes. The four different contexts are *isolated*, *word-initial*, *word-medial*, and *word-final*. For consonants that have two shapes, one is used for isolated and word-initial and the other is used for word-medial and word-final. Figure 3 provides and example of a letter that has four shapes.

The ع character is written:

ﻋ at word beginning (science علم)

ﻌ inside the word (institute معهد)

ﻊ in the word ending (group جمع)

ع when it is not linked (section فرع)

Figure 3: *Example of Arabic letter with four contextual shapes.*

4.1.2 Arabic vowels

The Arabic vowels are non-spacing marks that apply to the preceding consonant and are not often used because vowels can be clearly determined by a reader. The only thing to be aware of for context analysis is the legal combination of vowels. For example, the vowel ARABIC SHADDA (U+0651) is the only one which may be combined with other vowels and the vowel ARABIC SUKUN (U+0652) can not be combined with any other vowels because it removes any implicit vowel following a consonant.

4.1.3 Example MUTTCA Arabic context analyzer for rendering

Below is an example source file specifying a very simple context analyzer for rendering Standard Arabic text encoded in Unicode. This simplistic example does not provide a rule for

generating the LAM-ALIF ligature or rules for skipping vowels and the TATWEEL, but such rules are simply a matter of adding some more classes and iterating through the cases that can be expressed as rules.

```
# The Multilingual Unicode Text Toolkit (MUTT)
# Computing Research Laboratory, New Mexico State University
# Arabic writing rules
# Copyright (c) 1998, 1999 - CRL
#
# The rendering rules are applied on the range of the basic Arabic characters
# of Unicode (U+0600D to U+06FF / U+060C to U+06F9) and use the contextual
# glyphs of the Arabic Presentation Forms-B (U+FE70 to U+FEFF).
#
SCRIPT Arabic
LANGUAGE Arabic
VARIANT CRLArabic

BEGIN MAPPING
U+0621:U+FE80
U+0626:U+FE89
U+0628:U+FE8F
U+062A:U+FE95
U+062B:U+FE99
U+062C:U+FE9D
U+062D:U+FEA1
U+062E:U+FEA5
U+0633:U+FEB1
U+0634:U+FEB5
U+0635:U+FEB9
U+0636:U+FEBD
U+0637:U+FEC1
U+0638:U+FEC5
U+0639:U+FEC9
U+063A:U+FECD
U+0641:U+FED1
U+0642:U+FED5
U+0643:U+FED9
U+0644:U+FEDD
U+0645:U+FEE1
U+0646:U+FEE5
U+0647:U+FEE9
U+064A:U+FEF1
U+0622:U+FE81
U+0623:U+FE83
U+0624:U+FE85
U+0625:U+FE87
U+0627:U+FE8D
U+0629:U+FE93
U+062F:U+FEA9
U+0630:U+FEAB
U+0631:U+FEAD
U+0632:U+FEAF
U+0648:U+FEED
U+0649:U+FEEF
END MAPPING
```

```
BEGIN CLASSES
# Dual joining characters

# Independent forms of dual joining characters
C1={U+FE89,U+FE8F,U+FE95,U+FE99,U+FE9D,U+FEA1,U+FEA5,U+FEB1,U+FEB5,
    U+FEB9,U+FEBD,U+FEC1,U+FEC5,U+FEC9,U+FECD,U+FED1,U+FED5,U+FED9,
    U+FEDD,U+FEE1,U+FEE5,U+FEE9,U+FEF1}

# Initial forms of dual joining characters
C2={U+FE8B,U+FE91,U+FE97,U+FE9B,U+FE9F,U+FEA3,U+FEA7,U+FEB3,U+FEB7,
    U+FEBB,U+FEBF,U+FEC3,U+FEC7,U+FECB,U+FECF,U+FED3,U+FED7,U+FEDB,
    U+FEDF,U+FEE3,U+FEE7,U+FEEB,U+FEF3}

# Medial forms of dual joining characters
C3={U+FE8C,U+FE92,U+FE98,U+FE9C,U+FEA0,U+FEA4,U+FEA8,U+FEB4,U+FEB8,
    U+FEBC,U+FEC0,U+FEC4,U+FEC8,U+FECC,U+FED0,U+FED4,U+FED8,U+FEDC,
    U+FEE0,U+FEE4,U+FEE8,U+FEEC,U+FEF4}

# Final forms of dual joining characters
C4={U+FE8A,U+FE90,U+FE96,U+FE9A,U+FE9E,U+FEA2,U+FEA6,U+FEB2,U+FEB6,
    U+FEBA,U+FEBE,U+FEC2,U+FEC6,U+FECA,U+FECE,U+FED2,U+FED6,U+FEDC,
    U+FEDE,U+FEE2,U+FEE6,U+FEEA,U+FEF2}

# Right joining characters

# Independent-initial forms of right joining characters
C5={U+FE81,U+FE83,U+FE85,U+FE87,U+FE8D,U+FE93,U+FEA9,U+FEAB,U+FEAD,
    U+FEAF,U+FEED,U+FEEF}

# Medial-final forms of right joining characters
C6={U+FE82,U+FE84,U+FE86,U+FE88,U+FE8E,U+FE94,U+FEAA,U+FEAC,U+FEAE,
    U+FEB0,U+FEEE,U+FEF0}
END CLASSES

BEGIN RULES
C1,C1:1, "a+2,b+1";      /* Convert first to initial, isolated to final. */
C1,C5:1, "a+2,b+1";      /* Convert first to initial, isolated to final. */
C4,C1:1, "a+2,b+1";      /* Convert final to medial, isolated to final.  */
C4,C5:1, "a+2,b+1";      /* Convert final to medial, isolated to final.  */
C5,C5:0, "b+1";          /* Convert isolated to final, ignore the first. */
C5,C1:0, "b+1";          /* Convert isolated to final, ignore the first. */
END RULES
```

4.2 Simple context analyzer for Hindi words matching the pattern **ur*i* encoded in Unicode

```
#
# A short context analyzer that captures the Hindi word "purti" as well as
# other words matching a similar pattern, based on the CRL variation of the
# freely available Sibal Devanagari fonts.
#
# Devanagari writing rules
# Copyright (c) 1999 – CRL
#
```

```
SCRIPT Devanagari
LANGUAGE Hindi
VARIANT CRLSibalFont

BEGIN VALUES
sub_ra = U+E07E         /* Subjoined RA location in this font.   */
repha = U+E07F          /* REPHA location in this font.          */
half = U+100            /* Half-consonant offset for this font.  */
dependent = U+100       /* Dependent vowel offset for this font. */
END VALUES

BEGIN CLASSES
# Devanagari consonants minus RA which is handled specially.
C1 = {U+0915, U+0916, U+0917, U+0918, U+0919, U+091A, U+091B, U+091C, U+091D,
      U+091E, U+091F, U+0920, U+0921, U+0922, U+0923, U+0924, U+0925, U+0926,
      U+0927, U+0928, U+0929, U+092A, U+092B, U+092C, U+092D, U+092E, U+092F,
      U+0931, U+0932, U+0933, U+0934, U+0935, U+0936, U+0937, U+0938, U+0939}

C2 = {U+0930}                       /* The RA. */

# The independent vowels minus I which is handled specially.
C3 = {U+0905, U+0906, U+0908, U+0909, U+090A, U+090B, U+090C, U+090D, U+090E,
      U+090F, U+0910, U+0911, U+0912, U+0913, U+0914}
C4 = {U+0907}                       /* The vowel I. */
C5 = {U+094D}                       /* The VIRAMA. */
END CLASSES

BEGIN RULES
C1,C3:0, "b+dependent";              /* Rule to capture consonant+vowel. */
C2,C5,C1,C4:1, "d+dependent,c,repha"; /* Capture RA+VIRAMA+consonant+I.   */
END RULES
```

V. Conclusion

Although the context analyzers for rendering presented above are simple, they demonstrate that this architecture is easy for users to create, maintain, and modify. It should be noted that this approach says nothing about rearranging text for correct directionality. A well-known algorithm already exists for this sort of reordering and this context analysis approach can be applied to text as it is being reordered for directionality. Some effort is being made to make sure the context analyzer parser and compiler are as platform-independent as possible by implementing the initial version in Java.

VI. References

[BOUA.97] Boualem A.M., Harié S., “MtScript: a multilingual text editor”, Computers and the Humanities, Volume 31, No. 2, Kluwer Academic Publishers, 135-151, 1997.

[MULT.97] MULTEXT project was coordinated by the CNRS "Parole et Langage" Laboratory for building standard methods for linguistic data representation and developing language processing tools for about fifteen languages.
http://www.lpl.univ-aix.fr/projects/multext/

[MUTT.98] The Multilingual Unicode Text Toolkit, CRL, New Mexico State University.
ftp://crl.nmsu.edu/CLR/multiling/unicode/

[UNST.96] The Unicode Standard, Version 2.0, The Unicode Consortium, Addison-Wesley Developers Press, 1996.