1-0 Transformation Form of UTF-8

Shengyuan Wu

Shandong University, Shandong 250061, China
Email: wsy@sdu.edu.cn

Abstract. Based on Multilevel Mark Theory, 11-10 and 1-0 transformation form of UTF-8 are proposed in this paper. The transformation between UCS and 1-0 form of UTF-8 is introduced, then, the transformation between Local Code and 1-0 Form of UTF-8 is discussed in detail.

Keywords: Multilevel Mark theory; Unicode; UTF-8; UNIX

1. Introduction

Based on Multilevel Mark Theory(1, 2), a mixture object search method is realized, it can search various objects just like searching characters, and raise Chinese text search to word level. (3, 4) UTF-8 provides a UNIX compatible transformation format of Unicode Code System (UCS), and makes UNIX systems support multi-lingual text in a single encoding with multi-bytes. The mechanism of realizing UTF-8 influences the average length of UTF-8 and the transformation efficiency. The original form of UTF-8 (5, 6) was proposed in 1992, since then not any other forms have been proposed.
In the original proposal of UTF-8, the first byte of multi-byte code determines the number of bytes of the code by setting the number of bits in high order to 1 and following a 0 bit; and the high-order bit of other byte is set to 1 and following a 0 bit. So, this form can be called as 110-10 form of UTF-8. Based on Multilevel Mark Theory(1, 2), 110-10 form of UTF-8 can be simplified to 11-10 form of UTF-8, and 11-10 form of UTF-8 can be further simplified to 1-0 form of UTF-8.

2. 11-10 transformation form of UTF-8

The proposal of 11-10 form of UTF-8 is shown in Table 1. It is easy to differentiate the bytes of a code, and to differentiate codes in a code sequence. Although the byte number of a code is not marked in the first byte, it is easy to count it.

Table 1. Illustrating the 11-10 form of UTF-8

UCS	Free bits	11-10 UTF-8 sequence
U-00000000 - U-0000007F:	7	0*xxxxxxx*
U-00000080 - U-00000FFF:	12	11*xxxxxx* 10*xxxxxx*
U-00001000 - U-0003FFFF:	18	11xxxxxx 11*xxxxxx* 10*xxxxxx*
U-00040000 - U-00FFFFFF:	24	11*xxxxxx* 11*xxxxxx* 11*xxxxxx* 10*xxxxxx*
U-01000000 - U-3FFFFFFF:	30	11xxxxxx 11*xxxxxx* 11*xxxxxx* 11*xxxxxx* 10*xxxxxx*
U-40000000 - U-7FFFFFFF:	36	11xxxxxx 11xxxxxx 11*xxxxxx* 11*xxxxxx* 11*xxxxxx* 10*xxxxxx*

Take following code sequence as an example.
0xxxxxxx11xxxxxx 10xxxxxx 0xxxxxxx 11xxxxxx 11xxxxxx 10xxxxxx

Scanning the sequence from left to right, the first byte with mark bit 0 in the first bit is an ASCII code, the second byte is with mark “11”, so this is the first byte of a multi-byte code, the third byte is with mark “10”, so this is the end of the multi-byte code. The fourth byte is an ASCII code. The last 3 bytes is a 3 byte code.
A character code in a 11-10 form of UTF-8 code sequence starts with a byte marked with “11”, ended with a byte marked with “10”, or starts with a byte marked with “0” and ended this byte.
It is easy to make the transformation programs by the data structure as shown.
Based on Multilevel Mark Theory (1, 2), this form can be simplified further; mark 11 can be simplified as mark 1, and mark 10 can be simplified as mark 0.

3. 1-0 transformation between UCS and 1-0 form of UTF-8

3.1 1-0 transformation form of UTF-8

11-10 form of UTF-8 can be simplified as 1-0 form of UTF-8, as shown in Table 2. UTF-16 or UTF-32 is not compatible with existing UNIX implementations. The main problem is some control character of ASCII, such as null bytes and/or the ASCII slash ("/"). To make these character encodings in compatibility with historical file systems; the control character in the last byte of multi-byte encoding sequence should be delete. If the 7 bit value of last byte is < 20, i.e. the third bit is 0 (counted from left); then, the third bit is set to 1. The original information of third bit is put in the rightest bit in the byte just left to the rightest byte, as shown in table 2.
In 1-0 form of UTF-8, at most 5 bytes is needed to transform UTF-32, and the implication mechanism is much simpler.
It is easy to distinguish the character codes in the 1-0 form of UTF-8 code sequence. Take following code sequence as an example.
0xxxxxxx1xxxxxxx 0xxxxxxx 0xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx
Scanning the sequence from left to right, the first byte with mark bit 0 in the first bit, it is an ASCII code, the second byte is with mark 1, so this is a multi-byte code, the third byte is with mark 0, so this is the end of the multi-byte code. The fourth byte is an ASCII code. The last 3 bytes is a 3 byte code.

Table 2. Illustrating The 1-0 form of UTF-8

UCS	Free bits	7 bit value	1-0 UTF-8 sequence
U-00000000 - U-0000007F:	7		0xxxxxxx
U-00000080 - U-00001FFF:	13	<20 >=20	1xxxxxx0 001xxxxx 1xxxxxx1 0xxxxxxx
U-00002000 - U-000FFFFF:	20	<20 >=20	1xxxxxxx 1xxxxxx0 001xxxxx 1xxxxxxx 1xxxxxx1 0xxxxxxx
U-00100000 - U-07FFFFFF:	27	<20 >=20	1xxxxxxx 1xxxxxxx 1xxxxxx0 001xxxxx 1xxxxxxx 1xxxxxxx 1xxxxxx1 0xxxxxxx
U-08000000 - U-FFFFFFFF:	34	<20 >=20	1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxx0 001xxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxx1 0xxxxxxx

A character code in a 1-0 form of UTF-8 code sequence starts with a byte marked with “1”, followed with byte marked with “1”, and ended with a byte marked with “0”, or starts with a byte marked with “0” and ended this byte.

3.2 Transformation between UCS and 1-0 form of UTF-8

The transformation between UCS and 11-10 form of UTF-8 is easy and similar to the transformation between UCS and 1-0 form of UTF-8, so the former is omitted.

Table 3. Concatenation preparation for 1-0 form of UTF-8

UCS	1-0 UTF-8 sequence
U-00000000 - U-0000007F:	0xxxxxxx
U-00000080 - U-00001FFF:	1yyyyyyy$_0$ 0xxxxxxx
U-00002000 - U-000FFFFF:	1zzzzzzz 1yyyyyyy$_0$ 0xxxxxxx
U-00100000 - U-07FFFFFF:	1uuuuuuu 1zzzzzzz 1yyyyyyy$_0$ 0xxxxxxx
U-08000000 - U-FFFFFFFF:	1qqvvvvv 1xxxxxxx 1zzzzzzz 1yyyyyyy$_0$ 0xxxxxxx

Table 4. Demonstration Concatenation for 1-0 form of UTF-8

UCS	Concatenation of UTF-8 sequence
U-00000000 - U-0000007F	0xxxxxxx
U-00000080 - U-00001FFF	000yyyyy yx y$_0$xxxxx
U-00002000 - U-000FFFFF	00000000 000zzzzz zzzyyyyy yx y$_0$xxxxx
U-00100000 - U-07FFFFFF	00000uuu uuuuzzzz zzzyyyyy yx y$_0$xxxxx
U-08000000 - U-FFFFFFFF	vvvvvuuu uuuuzzzz zzzyyyyy yx y$_0$xxxxx

The UCS value is easy to be concatenated as shown in table 3 and table 4. The bits marked by q in table 3 are not used in the transformation. y_0 in the UTF-8 sequence above refers to the rightest bit in the byte left to the rightest byte of a character code, $y_0=0$ as the 7 bit value of the rightest byte <20, otherwise, $y_0=1$.
It is easy to implement the transformation programs by table 3 and table 4.

4. Transformation between Local Code and 1-0 Form of UTF-8

The transformation between local codes and UTF-8 are frequently used, this used to take two steps, the local code (or UTF-8) is first transformed into UCS, and then transformed to UTF-8 (or local code).
For 1-0 Form of UTF-8, the transformation between local codes and UTF-8 can be done by one step. This makes computer time and space greatly saved.

4.1 Transformation for 1 byte local codes

Because the first bit of each byte of 1-0 Form of UTF-8 is taken as mark bit; to avoid losing information in the transformation between local codes and 1-0 Form of UTF-8, codes of a local code system should be divided into one or more code classes. For 1 byte character code, if it is 7 bit coded characters, i.e. the first bit of the character code is always 0 or 1; then, the code system can be represented by one class

codes, such as: ASCII, ISO-8859-1, ISO-8859-2, and etc.
A byte sequence of 1-0 Form of UTF-8 for 1 byte local codes consists of 2 bytes, as shown in Table 5.

Table 5. Illustration the transformation between 1 byte local codes and 1-0 Form of UTF-8

Related local codes		1-0 UTF-8 sequence	
		Code class byte	Character code
ASCII	0xxxxxxx	10000000	0xxxxxxx
ISO-8859-1	1xxxxxxx	10000001	0xxxxxxx
ISO-8859-2	1xxxxxxx	10000010	0xxxxxxx
/	/	/	/
ISO-8859-16	1xxxxxxx	10010000	0xxxxxxx

In a code string, if ACSCII code is as default code for 1 byte code, then the class byte of ACSCII can be omitted, as shown in the following example.
0xxxxxxx100000010xxxxxxx0xxxxxxx100000100xxxxxxx0xxxxxxx
Scanning the string from left to right, the first code is an ASCII code, 1 byte long; the second is an ISO-8859-1 code, 2 byte long; the third is also an ASCII code; the fourth is an ISO-8859-2 code, 2 byte long; the fifth code is also an ASCII code.

4.2 Transformation for multiple byte local codes

GB18030 includes three kind codes, 1 byte, 2 bytes and 4 byte codes, as shown in Table 6. The 2 byte codes 0f GB18030 is the same as GBK. According to the second byte of GBK, 0xA0~ 0xFE, 0x40 - 0x7E and 0x80 -0x9F; GBK can be divided into 3 code classes: GBK-0, GBK-1, and GBK-2. GB2312 is included in code class: GBK-0. KSC 5601 can also be represented by one class codes. According to the second byte: 0x40~ 0x7E and 0xA1~0xFE, Big5 can be represented by 2 class codes: Big5-1 and Big5-2.
1 byte code of GB18030 can be divided into 2 parts, ASCII and 0x80; 0x80 can be represented by 1 class of code.

Table 6. GB18030 code space

	Code space				Code number
1 byte	0x00~0x80				129
2 bytes	First byte		Second byte		23940
	0x81 ~ 0xFE		0x40 ~ 0x7E	0x80 ~ 0xFE	
4 bytes	First byte	Second byte	Third byte	Fourth byte	1587600
	0x81~ 0xFE	0x30~0x39	0x81~0xFE	0x30~0x39	

To make these character encodings in compatibility with historical file systems; the control character in rightest byte of multi-byte encoding sequence should be deleted. If the 7 bit value of last byte is < 20, the third bit is 0; then, third bit is set to 1. The original information of third bit can be distinguished by the class byte. For example, the 7 bit value of the second byte of GBK-2 is less than 0X20, so the third bit of the rightest byte of the UTF-8 sequence is set to 1 as shown in Table 7.

Table 7. Illustration the transformation between 2 byte local codes and 1-0 Form of UTF-8

Related 2 byte local codes			1-0 UTF-8 sequence		
	First byte	Second byte	Class byte	Character code	
GBK-0	1xxxxxxx	1xxxxxxx(0xA0~0xFE, GB2312 included)	10000000	1xxxxxxx	0xxxxxxx
GBK-1	1xxxxxxx	0xxxxxxx(0x40~ 0x7E)	10000001	1xxxxxxx	0xxxxxxx
GBK-2	1xxxxxxx	1x0xxxxx(0x80~ 0x9F)	10000010	1xxxxxxx	0x1xxxxx
BIG5-1	1xxxxxxx	1xxxxxxx(0x40~ 0x7E)	10000011	1xxxxxxx	0xxxxxxx
BIG5-2	1xxxxxxx	0xxxxxxx(0xA1~0xFE)	10000100	1xxxxxxx	0xxxxxxx
JISX 0208	1xxxxxxx	1xxxxxxx	10000101	1xxxxxxx	0xxxxxxx
KSC 5601	1xxxxxxx	1xxxxxxx	10000110	1xxxxxxx	0xxxxxxx

If 2 byte default character is GBK-0 code（GB2312 belong to GBK-0 class）, then code class byte for GBK-0 can be omitted in a string. Scanning the string in Fig.1, from left to right, the first and the second character are GBK-0 code, each 2 byte long; the third is a character of GBK-1, 3 byte long; the fourth is a code of BIG 5-1 code, 3 byte long; the fifth is a JISX 0208 code, 3 byte long; the sixth is a KSC 5601 code, 3 byte long.

1xxxxxxx0xxxxxxx1xxxxxxx0xxxxxxx100000011xxxxxxx0xxxxxxx100000111xxxxxxx0xxxxxxx
100001011xxxxxxx0xxxxxxx100001101xxxxxxx0xxxxxxx

Fig. 1. a string of UTF-8 sequence for 2 byte local codes

4 byte code of GB18030 can be represented by 1 class as shown in Table 8.

Table 8. Illustration the transformation between 4 byte local codes and 1-0 Form of UTF-8

4 byte code of 18030	1-0 UTF-8 sequence
1xxxxxxx0xxxxxxx1xxxxxxx0xxxxxxx	1xxxxxxx1xxxxxxx1xxxxxxx0xxxxxxx

4.3 Code class byte and code class table

Code class table is the relation between local code and related 1-0 Form of UTF-8. Table 5 can be as a 1 byte code class table. Table 7 can be as a 2 byte code class table.
It doesn't need to put the code class byte before the 1-0 Form of UTF-8 if only one code class of character code of the same length in the text. For example, for 1 byte character, ASCII is default code, for 2 byte character, GBK-0 is default code; in this case, no need code class byte for ASCII and GBK-0 codes. However, the other classes of GBK should be with code class byte. The frequency of GB 2312 in GBK is about ：99,999%; therefore, only one out of 10000 Chinese characters with code class byte in the byte string, i.e. only one out of 10000 Chinese characters is 3 byte long; so, the average length of Chinese character codes is about 2.0001 bytes.
There are two kinds of code class byte: major and minor, the major points out which code class the right

bytes belong to; as shown in Fig.1 and Fig.2; the minor only points out the right byte is a code class byte. To distinguish the two kinds of code class bytes, a definite value of the minor is assumed, for example, a value of 0XFF is assigned here. If value of the code class byte is 0XFF, then the code class byte is minor; otherwise, major. The code class of the last character and the fourth character as shown in Fig.2 are minor.

A string of UTF-8 sequence for 1, 2 and 4 byte local codes is illustrated in Fig.2, 1 byte and 2 byte default class code are defined as before, and the 4 byte default character is 4 byte codes of GB18030. Scanning the string from left to right, the first code is an ASCII code, 1 byte long; the second is a GBK-0 code, 2 byte long; the third is also an ASCII code; the fourth is a GBK-1 code, 3 byte long; the fifth code is a 4 byte code of GB18030. The sixth code is a BIG5-1 code. The last code is an ISO-8859-1 code; 3 byte long. This is because the 2 byte default code is GBK-0, the ISO-8859-1 code with 2 code class bytes; the first code class is a minor, the second is major. GBK-1 and BIG5-1 code is also with class byte.

0xxxxxxx1xxxxxxx0xxxxxxx0xxxxxxx100000011xxxxxxx0xxxxxxx1xxxxxxx1xxxxxxx
1xxxxxxx0xxxxxxx100000111xxxxxxx0xxxxxxx1111111100000010xxxxxxx

Fig. 2. a string of UTF-8 sequence for 1, 2 and 4 byte local codes

If a text or a web page includes multiple code class of character codes, then a code class table should be included in the text or page.

5. Conclusion

For 11-10 form of UTF-8 and 1-0 form of UTF-8, the number of bytes in a code is unlimited. So as used in coding characters or other objects, the code space is unlimited. 1-0 Form of UTF-8 is compact, compatible, efficient, and easy to be parsed.

A method of the transformation between local code and 1-0 Form of UTF-8 is discussed; the distinguishing of different local codes is by code class just as the distinguishing of different local telephone number by area code.

Word segmentation is a bottleneck problem in Chinese information processing; this is because Chinese isn't a segmented language; 1-0 Form of UTF-8 can make Chinese to be segmented, this will be discuss later.

References

1. Shengyuan, Wu, Object representing and processing method and apparatus http://patents1.ic.gc.ca/details?patent_number=2526701&language=en_ca, Canadian intellectual property office, 2006
2. Shengyuan, Wu, object representing and processing method and apparatus, http://www.wipo.int/ipdl/ipdl-cimages/view/pct/getbykey5?key=04/109492.041216, patent cooperation treaty, 2004
3. Shengyuan, Wu, A search Method for text and multimedia, Journal of Shandong University (natural science), Vol. 41 Supp2, 2006, 7, 62-67
4. Shengyuan, Wu, Multiple object software demo, http://www.mostcode.com/newteck/VideoDemo.htm/, 2006
5. The Unicode Consortium, The Unicode Standard 4.0, *http://www.unicode.org/versions/Unicode4.0.0, 2006*
6. John O'Conner, UTF-8: What is it and why is it important? *http://www.joconner.com/javai18n/articles/UTF8.html#_ftn1,2001*