Unicode and UTF-8

1 About Text

The Problem

Most computer science students are familiar with the ASCII *character encoding scheme*, but no others. This was the most prevalent encoding for more than forty years. The ASCII encoding maps characters to 7-bit integers, using the range from 0 to 127 to represent 94 printing characters, 33 control characters, and the space. Since a byte is usually used to store a character, the eighth bit of the byte is filled with a 0.

The problem with the ASCII code is that it does not provide a way to encode characters from other scripts, such as Cyrillic or Greek. It does not even have encodings of Roman characters with diacritical marks, such as ę, ą, ś, or ó. Over time, as computer usage extended world-wide, other encodings for different alphabets and scripts were developed, usually with overlapping codes. These encoding systems conflicted with one another. That is, two encodings could use the same number for two different characters, or use different numbers for the same character. A program transferring text from one computer to another would run the risk that the text would be corrupted in the transition.

Unifying Solutions

In 1989, to overcome this problem, the *International Standards Organization* (*ISO*) started work on a universal, all-encompassing character code standard, and in 1990 they published a draft standard (ISO 10646) called the *Universal Character Set* (UCS). UCS was designed as a superset of all other character set standards, providing *round-trip compatibility* to other character sets. Round-trip compatibility means that no information is lost if a text string is converted to UCS and then back to its original encoding.

Simultaneously, the *Unicode Project*, which was a consortium of private industrial partners, was working on its own, independent universal character encoding. In 1991, the Unicode Project and ISO decided to work cooperatively to avoid creating two different character encodings. The result was that the code table created by the *Unicode Consortium* (as they are now called) satisfied the original ISO 10646 standard. Over time, the two groups continued to modify the respective standards, but they always remain compatible. Unicode adds new characters over time, but it always contains the character set defined by ISO 10646-x. The most current Unicode standard is Unicode 6.0.

Unicode

Unicode contains the alphabets of almost all known languages, as diverse as Japanese, Chinese, Greek, Cyrillic, Canadian Aboriginal, and Arabic. It was originally a 16-bit character set, but in 1995, with Unicode 2.0, it became 32 bits. The Unicode Standard encodes characters in the range U+0000..U+10FFFF, which is roughly a 21-bit code space. The code reserves the remaining values for future use.

In Unicode, a *character* is defined as the smallest component of a written language that has semantic value. The number assigned to a character is called a *code point*. A code point is denoted by “U+” following by a hexadecimal number from 4 to 8 digits long. Most of the code points in use are 4 digits long. For example, U+03C6 is the code point for the Greek character φ.

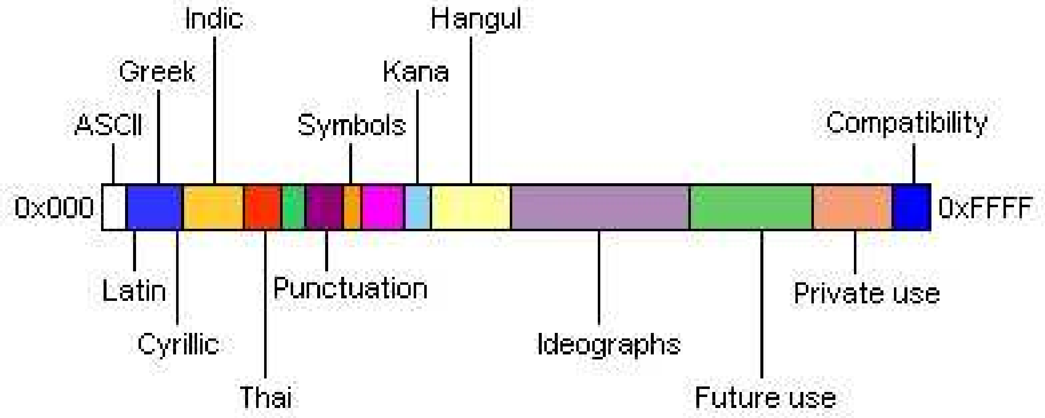

Figure 1: Unicode layout

UTF-8

Unicode code points are just numeric values assigned to characters. They are not representations of characters as sequences of bytes. For example, the code point U+0C36 is not a sequence of the bytes 0x0C and 0x36. In other words, it is not a character encoding scheme. If we were to use it as an encoding scheme, there would be no way to distinguish the sequence of two characters '\f' '$' (form feed followed by $) from the Greek character φ.

There are several encoding schemes that can represent Unicode, including UCS-2, UCS-4, UTF-2, UTF-4, UTF-8, UTF-16, and UTF-32. UCS-2 and UCS-4 encode Unicode text as sequences of either 2 or 4 bytes, but these cannot work in a UNIX system because strings with these encodings can contain bytes that match ASCII characters and in particular, "\0" or "/", which have a special meaning in filenames and other C library function parameters. UNIX file systems and tools expect ASCII characters and would fail if they were given 2-byte encodings.

The most prevalent encoding of Unicode as sequences of bytes is UTF-8, invented by Ken Thompson in 1992. In UTF-8 characters are encoded with anywhere from 1 to 6 bytes. In other words, the number of bytes varies with the character. In UTF-8, all ASCII characters are encoded within the 7 least significant bits of a byte whose most significant bit is 0.

UTF-8 uses the following scheme for encoding Unicode code points:

1. Characters U+0000 to U+007F (i.e., the ASCII characters) are encoded simply as bytes 0x00 to 0x7F. This implies that files and strings that contain only 7-bit ASCII characters have the same encoding under both ASCII and UTF-8.
2. All UCS characters larger than U+007F are encoded as a sequence of two or more bytes, each of which has the most significant bit set. This means that no ASCII byte can appear as part of any other character, because ASCII characters are the only characters whose leading bit is 0.
3. The first byte of a multibyte sequence that represents a non-ASCII character is always in the range 0xC0 to 0xFD and it indicates how many bytes follow for this character. Specifically it is one of 110xxxxx, 1110xxxx, 11110xxx, 111110xx, and 1111110x, where the x's may be 0's or 1's. *The number of 1-bits following the first 1-bit up until the next 0-bit is the number of bytes in the rest of the sequence.*

   All further bytes in a multibyte sequence start with the two bits 10 and are in the range 0x80 to 0xBF. This implies that UTF-8 sequences must be of the following forms in binary, where the x's represent the bits from the code point, with the leftmost x-bit being its most significant bit:

```
0xxxxxxx
110xxxxx  10xxxxxx
1110xxxx  10xxxxxx  10xxxxxx
11110xxx  10xxxxxx  10xxxxxx  10xxxxxx
111110xx  10xxxxxx  10xxxxxx  10xxxxxx  10xxxxxx
1111110x  10xxxxxx  10xxxxxx  10xxxxxx  10xxxxxx  10xxxxxx
```

4. The bytes 0xFE and 0xFF are never used in the UTF-8 encoding.

A few things can be concluded from the above rules. First, the number of x's in a sequence is the maxiumum number of bits that a code point can have to be to be representable in that many bytes. For example, there are 11 x-bits in a two-byte UTF-8 sequence, so all code points whose 16-bit binary value is at least 0000000010000000 but at most 0000011111111111 can be encoded using two bytes. In hex, these lie between 0080 and 07FF. The table below shows the ranges of Unicode code points that map to the different UTF-8 sequence lengths.

Number of Bytes	**Number of bits in Code Point**	**Range**
1	7	00000000 - 0000007F
2	11	00000080 - 000007FF
3	16	00000800 - 0000FFFF
4	21	00001000 - 001FFFFF
5	26	00200000 - 03FFFFFF
6	31	04000000 - FFFFFFFF

You can see that, although UTF-8 encoded characters may be up to six bytes long in theory, code points through U+FFFF, having at most 16 bits, can be encoded in sequences of no more than 3 bytes.

Converting a Unicode code point to UTF-8 by hand is straightforward using the above table.

1. From the range, determine how many bytes are needed.

2. Starting with the least significant bit, copy bits from the code point from right to left into the least significant byte.

3. When the current byte has reached 8 bits, continue filling the next most significant byte with successively more significant bits from the code point.

4. Repeat until all bits have been copied into the byte sequence, filling with leading zeros as required.

Example 1. To convert U+05E7 to UTF-8, first determine that it is in the interval 0080 to 07FF, requiring two bytes. Write it in binary as

```
0000 0101 1110 0111
```

The rightmost 6 bits go into the right byte after 10:

```
10 100111
```

and the remaining 5 bits go into the left byte after 110:

```
110 10111
```

So the sequence is 11010111 10100111 = 0xD7 0xA7, which in decimal is 215 in byte1 and 167 in byte 2.

Example 2. To convert U+0ABC to UTF-8, since it is greater than U+07FF, it is a three-byte code. In binary,

```
0000 1010 1011 1100
```

which is distributed into the three bytes as

```
1110 0000
10 101010
10 111100
```

This is the sequence 11100000 10101010 10111100 = 0xE0 0xAA 0xBC, which in decimal is 224 170 188, the Gujarati sign Nukta.

Exercise. Write an algorithm to do the conversion in general.