Omega — why Bother with Unicode?

Robin Fairbairns
University of Cambridge Computer Laboratory
Pembroke Street
Cambridge CB2 3QG
UK
Email: `rf@cl.cam.ac.uk`

Abstract

Yannis Haralambous' and John Plaice's Omega system employs Unicode as its coding system. This short note (which previously appeared in the UKTUG magazine *Baskerville* volume 5, number 3) considers the rationale behind Unicode itself and behind its adoption for Omega.

Introduction

As almost all TEX users who 'listen to the networks' at all will know, the Francophone TEX users' group, GUTenberg, arranged a meeting in March at CERN (Geneva) to 'launch' Ω. The UKTUG responded to GUTenberg's plea for support to enable TEX users from impoverished countries to attend, by making the first disbursement from the UKTUG's newly-established Cathy Booth fund. The meeting was well attended, with representatives from both Eastern and Western Europe (including me; I also attended with UKTUG money), and one representative from Australia (though he is presently resident in Europe, too).

The speakers at the meeting were Michel Goossens (the president of GUTenberg[1]), as host for GUTenberg and as an expert on background to, and the use of Unicode), and Yannis Haralambous and John Plaice, Ω's two developers.

The meeting can be accounted a success; all that attended enjoyed themselves, and also learnt a lot.

This paper is a minor revision of an article I wrote for the UKTUG magazine *Baskerville*, volume 5, number 3, and outlines some of my views of (and support for) the choices that Haralambous and Plaice have made. I will consider the arguments for using Unicode as a foundation for future text processing, in particular (of course) TEX-related processing.

What *is* Ω?

Ω (Haralambous and Plaice, 1994) is an extension of TEX and related programs that has been designed and written by Yannis Haralambous (Lille) and John Plaice (Université Laval, Montréal). It follows on quite naturally from Yannis' work on exotic languages (see, among many examples, Haralambous, 1990; 1991; 1993; 1994), which have always seemed to me to be bedevilled by problems of text encoding.

Simply, Ω (the program) is able to read scripts that are encoded in Unicode (or in some other code that is readily transformable to Unicode), and then to process them in the same way that TEX does. Parallel work has defined formats for fonts and other necessary files to deal with the demands arising from Unicode input, and upgraded versions of METAFONT, the virtual font utilities, and so on, have been written. Ω itself is based on the normal `Web2C` distribution that is at the base of most modern Unix implementations, and of at least one of the PC versions that is freely available.

Why Unicode?

There are something between 3000 and 6000 languages in use in the world, for which a writing system exists. (The set of languages is shrinking all the time as the deadening effect of cultural intrusion, primarily through the electronic media, overwhelms the desire to support existing cultures to the extent of teaching their language to the young.) The distribution of languages is by no means even throughout the globe (Michel showed us a map), and there are many that have not been and will presumably now never be formally recorded.

When we come to writing systems, we find almost every variation imaginable in use somewhere in the world. The Latin-like system (written left to right with modest numbers of diacritics simply arranged) has very wide penetration, not least because so many languages were first written down by

[1] And now (at the time of writing) president-elect of TUG itself.

Western European missionaries or other explorers. Languages such as Vietnamese are classified as 'complex Latin-like', with ≥ 2 diacritics per character; an artificial example of the same effect is IPA (the International Phonetic Alphabet) which has sub- and super-scripts and joining marks. Languages such as Hebrew and Arabic are written right to left, and constitute another class. Then there are the multiple-ligature writing systems typified by the Indic languages such as Devanagari (of which we had a fascinating exposition at the 1993 UKTUG Easter meeting on 'non-American' languages, from Dominik Wujastyk), and finally the syllabic scripts (such as Korean Hangul and Japanese Hiragana and Katakana), and the ideographic scripts (Chinese and Japanese Kanji).

Encodings are needed for computer operations on language of any sort. There are differences between the coded representation and the written (or printed) representation. Everyone who's read about TeX at all will know about ligatures (the CM fonts, and most PostScript fonts, implement ligatures so that, for example, '`fl`' typed appears as 'ﬂ' printed). More significantly, almost all adults in Western cultures write 'joined-up', which is in itself application of a form of ligature. All these ligatures are for presentation, not for information, and so it is unreasonable for them to be represented in a character set. Other ligatures, however, form real characters in some languages (examples are æ in Danish and Norwegian, and œ in French).

Each encoding represents a 'character set' that is to be used by the computer; the history of how these character sets have developed is long and sorry indeed. In the 'dark ages' (in fact, as recently as the early 1960s, when I started computing), every make of computer system had its own character code, many of them based on the 5-bit teleprinter codes used in telex printers. Eventually, the rather more sophisticated teletypes appeared, which used seven bits of an eight-bit code; this 7-bit codification was standardised as ASCII (the American Standard Code for Information Interchange), which was (in the area of application it was designed for) an excellent code. It had all the properties needed for many of the significant development of computers in the 1960s, but it had one serious flaw: it was not able to encode diacritics, which are used in almost every language (but which your all-American information interchanger would seldom have a need for).

To regularise the resulting mess, ISO adopted the ASCII standard as the basis for an international 7-bit character set, ISO 646. ISO 646 is identical to ASCII in the code points that it specifies; however, some of the characters that ASCII does specify are left "for national variation" in ISO 646; ASCII itself then became the USA national variation of ISO 646. An example of national variation is defined for the UK, which specifies that the code point that holds '`#`' in ASCII should hold a pounds sign (£). There are versions for various Nordic languages that include characters such as æ or å in place of braces, a version for French with acute, grave and circumflex-accented letters, one for German that offers letters with umlauts and 'sharp s' (ß).

There were various attempts at mechanisms to assign different character sets for use by those who need to use characters from several different sets (for example, someone writing a Swedish-English Dictionary); an example is ISO 2022, which defines escape sequences for such switches. These efforts proved impractical (at least they seemed so to me), and 8-bit developments of ISO 646 arose, with the ability (comfortably) to express more than one language.

Thus were born the ISO 8859 character sets. The commonest of these (at least in the ken of most English speakers) is ISO Latin-1 (ISO 8859-1, that is, part one of the multi-part standard), which was designed for use by Western Europeans. As well as the 'basic ASCII set' in the first 128 characters, it has diphthongs and vowels appropriate to most Western European languages. Oddly, it omits the œ dipthong that French uses, and (perhaps less surprisingly[2]) it omits some of the accent forms used by Welsh. ISO 8859 didn't stop with part 1, though; there are variants that accommodate Cyrillic (for Russian, Serbian, and several other languages of the old Soviet Union), Arabic, Hebrew, and so on.

This is all well and good, but it doesn't answer the needs of a writer preparing multilingual documents, except in the case that the multiple languages are accommodated in the same part of ISO 8859: it will happen some of the time, but most 'interesting' combinations will require switches of character set whenever the language changes.

So ISO (by this time, jointly with IEC) started development of an all-encompassing character set, to be numbered ISO/IEC 10646 (the difference of 10 000 is no accident). ISO/IEC 10646 was to accommodate every possible language in the world by the simple expedient of allowing 32-bit characters. Of course, no-one can comprehend a 32-bit character set, and so the set was to be structured, as a hypercube of different repertoires; the $(0, 0, 0, *)$ repertoire would

[2] Given that Wales would have been represented by the BSI in the standardisation process.

be the same ISO-Latin 1, but all the other sets could be accomodated, too.

Independently, Apple and Microsoft got together to found the Unicode consortium, whose aim was to define 16-bit characters that would cover all the economically important world. This criterion of economic importance could easily have brought down the whole edifice: the (increasingly important) languages of the Far East are at best syllabic (e.g., Korean; Korea claims 11 000 of the code points in Unicode), or even one character per word (e.g., Chinese; a full classical Chinese repertoire would require well in excess of 65 536 characters, thus sinking a 16-bit code single-handedly).

Unicode's sponsors therefore enforced a process called 'Han unification', which aims to put the 'same' character in any of Chinese, Japanese and Korean in the same slot in the table. This unification is a distinctly dubious exercise: the same character may have different significance in the different languages, but they are all represented by the same code point. Contrariwise, the Latin 'H', the Russian 'H' (which sounds as Latin 'N') and the Greek 'H' (capital 'η') all get different code points despite having the same paper representation. For this reason (among others), there remain doubts as to whether the Japanese, in important particular, will adopt Unicode as a long-term replacement for their own national standards.

In the shorter term, however, there remained the possibility that there would be two conflicting standards for the future of character codes – a *de facto* one (Unicode) and ISO/IEC 10646. The ISO/IEC standard reached its (nominal) final ballot without addressing the relation to Unicode ... but (fortunately) it failed at that hurdle, and for that reason. Standards people are notorious for ignoring the real world[3], but this time, they conceded defeat. ISO/IEC 10646 was edited to have the whole of Unicode as its $(0, 0, *, *)$ plane, and it has thus passed into the canon of published standards.

So we may now discuss Unicode without running out against the ISO/IEC standard: a splendid example of the behaviour known as "common sense prevailing".

Virtual Metafont and Fonts to Support Unicode

It is known that TeX is a general-purpose programming language. In 'plain' text, we would type `"hello world"`. For TeX output we would type ``` ``hello world'' ```, which would be transparently converted to "hello world". Thus, the two grave accents and the two single quotes constitute 'programming'. In the last analysis, you can "do everything with TeX".

When English is typeset, the convention is that the space, after the full stop is the end of a sentence, is expanded; TeX makes provision for this to happen by way of the `\sfcode` mechanism. When French is typeset, the convention is that the space is not expanded; the `\sfcode` mechanism can provide this style of typesetting, as well (cf. the `\frenchspacing` macro of plain TeX).

Other features of French typesetting are more difficult to provide in TeX. For example, an exclamation mark is separated from the sentence: "en français !"; to program this, the exclamation mark needs to become an 'active character', which is always a tricky thing to do.

Setting the French quotation marks (known as guillemets) becomes even more tricky; the guillemets look like little `<<` and `>>`, and the natural way to program them is by using repeated `<` or `>` characters; Bernard Gaulle's `french.sty` does this (also setting a space between the text quoted and the guillemets), but it's becoming more and more complicated; even more so when we consider the French rules for quotes within quotes.

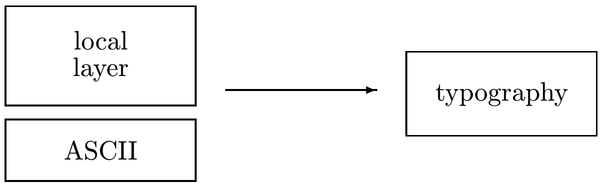
More problems arise when we consider the question of diacritics. English rather infrequently has diacritics, so it's not surprising that TeX's method of dealing with them isn't perfect. To typeset an accented character, e.g. ä, one must type `\"a`; which is typeset as two little boxes stacked on top of one another, rather like . This does work, but these composite glyphs no longer qualify (to TeX) as something that it's willing to hyphenate – TeX only hyphenates 'words' made up of sequences of letters. A language such as German, with hyphenation suppressed for many words, is hardly a language at all. These observations are what led to the definition of the Cork font encoding, in which a goodly proportion of Western European letters with diacritics appear as single characters; if they are thus represented, words containing them may be hyphenated.

With the Cork encoding, which is in effect an output encoding, we encounter a further problem relating to the nature of communication. The problem arises from the nature of character sets; while there are many well-established character sets, there are seriously different camps into which they fall. For example, the character 'Þ' (Thorn), appears in Microsoft Windows' character set but not in the Macintosh set, while 'Ω' appears in the Macintosh set but not in the Windows set; both of these sets are

[3] The author has spent an unconscionably long period of his life on these things, and is therefore in a position to know.

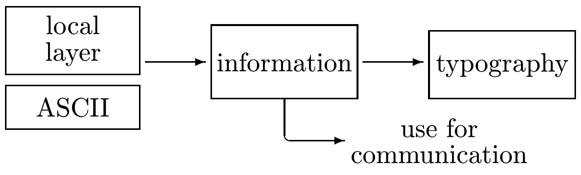
based on ASCII. To solve this problem, of encoding everything that appears in any character set, there has to be a super-encoding. This can be either a multi-character representation, as in the www encoding, html (for example the encoding would for é would be `&eacute;`), or a super-character set, as in Unicode.

In the present arrangement of typesetting technology, we have the situation where non-English users sit at a computer, and express their own language via a local layer in ASCII or a derivative of it – i.e., we have a picture like this:

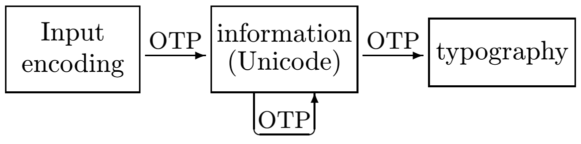


In this arrangement, the human interface allows the use of local characters, and the display will show what's typed. The typography does the display job again (possibly differently); however, communication of the text to be typeset is difficult, because of the local nature of the interface.

The information to be transmitted needs to be encoded. There is no limit to the number of local encodings that may exist; equally, there is no constraint on the representations used by the typographic system. However, to facilitate the transmission of information, a common schema of its representation in the coded date must exist.



The ultimate mechanism for ensuring that such a schema exists is to require that everything be transmitted in a common encoding scheme; Ω employs ISO/IEC 10646/Unicode for this. Input text is transformed into Ω's internal 'information' by an Omega Translation Process (OTP); OTPs may also be used to transform the information during its processing withing Ω, and an OTP is also used to derive the coding of the font, to be used for typesetting, from the Unicode-encoded information within Ω:



Conceptually, at least, this is exactly what one wants. In practice, the usefulness of Ω remains to be seen; the implementors are promising a version ('soon') that progresses from the status of pre-test. I, for one, am eagerly awaiting its appearance.

References

Y. Haralambous. "Arabic, Persian and Ottoman TEX for Mac and PC". *TUGboat* **11**(4), 520–524, 1990.

Y. Haralambous. "TEX and those other languages". *TUGboat* **12**(34), 539–548, 1991.

Y. Haralambous. "The Khmer script tamed by the Lion (of TEX)". *TUGboat* **14**(3), 260–270, 1993.

Y. Haralambous. "An Indic TEX preprocessor — Sinhalese TEX". *TUGboat* **15**(3), 301–301, 1994.

Y. Haralambous and Plaice, John. "First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer". *TUGboat* **15**(3), 344–352, 1994.