

The latest developments in Ω

John Plaiice

Département d'informatique, Université Laval, Ste-Foy (Québec) Canada G1K 7P4
John.Plaiice@ift.ulaval.ca

Yannis Haralambous

Atelier Fluxus Virus, 187, rue Nationale, F-59 800 Lille, France
Yannis.Haralambous@univ-lille1.fr

Abstract

The Ω system has been available since early 1995, and has been used experimentally in several sites around the world. We gather here some conclusions from this experimenting and explain what aspects will be included in version 1.3 of Ω , which should be the first large-scale release of the system. Not only will the portability and performance of Ω be improved substantially, but new features, including smart fonts and multidirectional support, will be included.

— * —

When Ω was first conceived, the primary objectives were to remove the 8-bit restrictions imposed by the original design of \TeX (number of characters, fonts, registers of each kind, etc.), as well as to offer the means necessary for multilingual typesetting, no matter how complex the script.

The 8-bit restrictions were removed quite easily by simply doubling the size of all data structures in the \TeX program and by introducing a variant of the `.tfm` file, called the `.xfm` file, in which fonts of up to 65,536 characters could be built.

For typesetting complex scripts, such as classical Arabic or Hebrew, a series of finite state automata, called Ω Translation Processes (Ω TPs), can be successively applied to the input character stream to do arbitrarily complex manipulations. After each application of an Ω TP, the macro-expansion facilities of \TeX are reinvoked, which means that the full power of \TeX is available every time an Ω TP is used.

Performance and portability

The current version of Ω currently resides on the `ftp.ens.fr` server, and has been used experimentally by several different groups in different countries. From their responses, we now understand what must be done for Ω to be a realistic replacement for \TeX .

First, Ω is too big! A typical run of Ω uses about 14MB, which is just fine when you are sit-

ting in front of a 500MHz-machine with 512MB, but certainly not on a typical portable. This tremendous size comes from the \TeX program structure, in which static arrays are allocated to handle primitives such as `\catcode` or `\delcode`, to store register values and font information, etc. However, the average user will never need 65,536 fonts of 65,536 characters each, nor 65,536 mu-registers, etc. Most of these huge tables are full of zeros, and it seems silly to have to go out and buy RAM and see your system slow down just so you can have lots of empty tables in your program.

This problem will be solved in the next version through the introduction of several primitives of the form `\MaxActiveCharacter`, `\MaxRegister`, `\MaxFont` or `\MaxWrite`. These primitives correspond to compile-time constants, which should be overridable upon loading Ω . By doing this, a single binary can be used, whatever the resources needed. According to Benjamin Bayart (École Supérieure d'Ingénieurs en Électrotechnique et Électronique in Paris), these primitives also make it possible for macro packages to determine whether the existing system has the required resources or whether a new run should be undertaken, with larger tables.

Second, Ω does not run correctly on Little Endian machines. This problem was solved by Benjamin Bayart and will be incorporated in version 1.3. As a result, there should be working versions of Ω for Intel boxes running DOS, Windows and Linux.

Finally, performance is unsatisfactory for Ω TPs that are being used for multilingual applications. Because the macro-expansion facilities are applied with every use of an Ω TP, the use of two successive Ω TPs can slow down Ω so that it runs 40% as fast as the original \TeX . This may be acceptable for limited applications where specialized effects are wanted, but it is certainly unacceptable in a production setting where thousands of pages are being generated every day.

Support for complex scripts

It turns out, however, that for any given language, very few \TeX primitives are required for typesetting high-quality output. As a result, we are looking for more efficient techniques that can be used for the standard cases.

In particular, one author (J.P.) has worked with ArborText, Inc. (Ann Arbor, Michigan), a supplier of SGML authoring and composition software and services, in developing typesetting support for ideogram scripts from East Asia. ArborText uses a \TeX -based engine for printing SGML documents, and this engine was modified so that it could output Japanese text.

The fundamental understanding of a font in \TeX is that each character has width, height, depth and italic correction. Characters are placed in order on the baseline, and the exact choice of character and the exact horizontal positioning can be adjusted using the ligature/kerning table.

This technique is reasonable for laying out alphabetic scripts where the characters are printed separately, as is the common case for Latin, Greek, Cyrillic, Armenian, Georgian, among others. Nevertheless, even for the Latin script, problems can arise: the Unicode standard provides for more than 900 precomposed characters. Building a complete ligature/kerning table for a font would require inordinate amounts of memory. Furthermore, it would be unlikely that, say, character 1EA9 (LATIN SMALL LETTER A WITH CIRCUMFLEX AND HOOK ABOVE), used in Vietnamese, would be found next to character 01CF (LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON), which is a Croatian digraph: much of the table would be useless. In fact, many of the characters have similar attributes: the difference between ‘è’ and ‘é’ is unlikely to influence the ligature/kerning program, so there are many opportunities for compressing it.

When we pass on to the ideogram scripts found in East Asia, all the characters have the same dimension. There are no ligatures, nor kerning. However, if a fixed grid is not chosen, then glue must be placed between successive characters. Furthermore, line breaks cannot occur after left-bracket-like characters or before right-bracket-like characters. To handle such situations, penalties must be placed automatically in the appropriate places.

For vowelized Arabic, the requirements are different yet again. Not only must the correct presentation form — isolated, initial, medial or final — of each consonant be chosen, but the diacritics, including vowels and hamza, must be properly placed with

respect to the consonants. To do this requires additional parameters about each character, designating the horizontal and vertical placement required to place the different diacritics. Finally, *keshidehs* (straight lines or Bézier curves) must be placed between consonants to fill out lines.

For the Arabic script in Nastaliq style, as is normally used for Farsi or Urdu, typesetting becomes even more complicated, since successive characters are not placed on the baseline. Rather, the characters within a word are placed in a sort of staircase situation. The first character, to the right, is placed highest. The lowest is the last character, to the left. Once again, extra character parameters are required so that the successive characters can be displaced vertically by the right amount.

The work undertaken with ArborText implied designing another extension to the font metric files so that an arbitrary number of different kinds of parameters could be defined for the font as a whole or for each individual character. Currently, five sorts of parameter can be defined: integer, fixword, rule, glue and penalty. In addition, the ligature/kerning program has been modified to allow the automatic insertion of glue and penalties between characters, as is required for East Asian ideogram fonts. In addition to changes to the \TeX driver, the `pltotf` and `tftopl` both had to be modified.

Under an agreement with ArborText, we will be incorporating these ideas into Ω . In fact, Ω will support a generalization of these ideas: the ligature/kerning table will allow two-dimensional capabilities, thereby solving all of the difficulties in typesetting calligraphic scripts such as Arabic.

Multiple directions

Multilingual typesetting requires printing in several directions. Scripts currently in use today can be categorized into four groups.

1. The most common group includes most of the world’s scripts (Europe, Caucasus, India and South-East Asia). Lines are read from left to right and pages are read from top to bottom. When diacritics are used, in most cases, they are placed above characters in a line.
2. The next group includes several scripts that originate in West Asia, such as Arabic and Hebrew. Lines are read from right to left and pages are read from top to bottom. Diacritics are placed both above and below characters in a line.
3. The third group includes the scripts found in East Asia. When the traditional scripts are

used, lines are read from top to bottom and pages are read from right to left. It is standard for technical documents to be printed in the same manner as the first group. When the traditional means are used for printing and, say, English text is inserted, then it will be rotated 90° clockwise, and one can read the English insertion as if the text were in landscape mode.

4. The final group consists of Mongolian and other languages in the Uighur region of China and in Mongolia. In these languages, lines are read from top to bottom and pages are read from left to right. One way to perceive such texts is that they are similar to Arabic or Syriac, but rotated 90° counterclockwise.

It should be understood that the writing direction affects the entire page. An Arabic text will indent from the right, and successive entries in a table will be placed from right to left. Similarly, headers in traditional ideogram typesetting are found at the right-hand side of the page. Nevertheless, some aspects remain constant across the different groups. No matter what group is being typeset, mathematics will remain as if it were part of the first group.

In Ω , a writing direction is defined using two parameters: the *primary direction* corresponds to the direction in which successive lines follow each other on a page and the *secondary direction* corresponds to the direction in which successive characters follow each other on a line. The above four groups can be summarized as follows:

	primary	secondary
1.	top-down	left-right
2.	top-down	right-left
3.	right-left	top-down
4.	left-right	top-down

In TEX parlance, the primary direction corresponds to ‘vertical’ displacement and the secondary direction to ‘horizontal’ displacement. For example, an `\hspace` command in Mongolian would actually mean a downwards displacement on the page, along the baseline.

Different writing directions can interact on the same page, possibly even in the same paragraph, for quotations, insertion of mathematics, etc. The common interactions are 1–2, 1–3, 1–4 and 3–4. The first two have been the subject of *TUGboat* articles (Knuth and MacKay 1987, Hamano 1990), and Ω will use similar techniques to effect the proper changes.

The writing direction can even change when printing the same script. This was a technique used in boustrophedon, for ancient Greek texts: it com-

bined the first two writing directions, alternating from line to line, with the characters mirrored as the text changed directions. In addition, ancient Egyptian texts would freely intermix the first two writing directions as well. We are looking at means to support these different techniques.

Conclusion

The coming releases of Ω , as outlined here, will ensure that Ω will truly become a multilingual successor to TEX .

References

- Hamano, Hisato. Vertical typesetting with TEX . *TUGboat* 11(3):346–352, 1990.
- Knuth, Donald, and Mackay, Pierre. Mixing right-to-left texts with left-to-right texts. *TUGboat* 8(1): 14–25, 1987.
- International Organization for Standardization/International Electrotechnical Commission. Information technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane (ISO/IEC 10646-1), 1993.
- The Unicode Consortium. The Unicode Standard: Worldwide Character Encoding, Version 1.0, Volume 1. Addison-Wesley, 1991.