MetaPost

Kanji-Sudokus: Integrating Chinese and graphics

Denis Roegel

1 Introduction

Recently, I had the need to get my hands on Werner Lemberg's excellent CJK package, for a talk on the Chinese calendar in which I wanted to use METAPOST figures with Chinese labels. This worked almost seamlessly.

Actually, this isn't quite true, but CJK is better and better integrated into TEX Live these days, and writing in Chinese, Japanese or Korean has become pretty much mundane with an up-to-date TEX environment. This wasn't so even a year ago. Nowadays, you still need to install various Linux (say) packages, and one is likely to run into trouble because some crucial element is missing. For instance, on my latest Ubuntu, the TEX Live setup wasn't complete and I was missing some Korean fonts I needed. By the time you read this, the problem may have been solved already.

To sum up, with the latest TEX Live 2007 setup, and perhaps a few additional Linux packages, as well as the latest Emacs, you are all set for typesetting beautiful CJK documents! Typesetting CJK has even become easier, because one can now write almost everything in UTF-8, without a need to postprocess the input file with Emacs macros (this procedure used to output a `.cjk` file which could only then be processed by LATEX). Now, the file you write is the file you process, and processing has become faster.

For METAPOST figures, the matter was also made easier. Up until recently, when including Chinese in METAPOST, one had first to produce a `.cjk` file, which could unfortunately not be processed by METAPOST. The `.cjk` file had to be slightly altered first, because the Emacs macros were not aware of the METAPOST format. This could have been corrected within the Emacs macros, but in fact, since the conversion to the `.cjk` file is now mostly an old story, the processing problems have also vanished. So, my advice is not only to switch to the latest TEX Live and Linux, but also to write CJK in UTF-8. It works!

2 A small example

I will illustrate the integration of Chinese and METAPOST with a small example. I will draw a Sudoku grid, not with Hindu-Arabic numerals, but with Chinese numerals. These numerals are 一 (1), 二 (2), 三 (3), 四 (4), 五 (5), 六 (6), 七 (7), 八 (8), and 九 (9).

A typical Sudoku problem reads as follows (this example from Wikipedia):

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

2.1 The grid

The whole Sudoku problem can be drawn as follows in METAPOST:

```
beginfig(1);
  string sol[];
  drawgrid(1.5pt,.5pt);
  % first row at the bottom
  % last row at the top
  sol1="000080079";sol2="000419005";
  sol3="060000280";sol4="700020006";
  sol5="400803001";sol6="800060003";
  sol7="098000060";sol8="600195000";
  sol9="530070000";
  fillgrid(sol)(false);
endfig;
```

The `drawgrid` macro is straightforward; it produces the horizontal and vertical lines (with `u` being for instance equal to 1 cm):

```
def drawgrid(expr tha,thb)=
  pickup pencircle scaled thb;
  for i=0 upto 9:
    draw (i*u,0)--(i*u,9u);
    draw (0,i*u)--(9u,i*u);
  endfor;
  pickup pencircle scaled tha;
  for i=0 upto 3:
    draw (3i*u,0)--(3i*u,9u);
    draw (0,3i*u)--(9u,3i*u);
  endfor;
enddef;
```

2.2 Filling the grid

In order to fill the grid, we need to access the position (i, j), where i is the column and j is the row, all numbered from 1 at the bottom-left cell. We therefore define the following macro, which takes i, j and a label that it centers in the middle of the cell. In our case, the label is scaled 200%, but how much you scale depends on the dimensions of the frame and on the base size of the font.

```
def pos(expr i,j,l)=
  label(l scaled 2,((i-.5)*u,(j-.5)*u));
enddef;
```

2.3 Cell entries

In order to put, say, the value 3 at position (2,9), we could write

```
pos(2,9,btex 3 etex);
```

However, we want to be more general and draw our figures from the string array `sol`. That way, some program can produce a problem and/or a solution, and the problem can easily be plugged into our macros. So, instead, we could write

```
pos(2,9,TEX(s));
```

where `s` is a string provided to the `TEX` macro. The latter is defined by loading the package `TEX`:

```
input TEX;
```

This, however, is not very efficient, because it will call TEX up to 81 times. And besides, it won't take care of Chinese numerals when we need them. So, we are looking for something more flexible. The `latexmp` METAPOST package will suit our needs. The previous label is now obtained with

```
pos(2,9,textext(s));
```

One of the advantages of the `latexmp` package is that it will require only two runs of LaTeX, and not one for every label. This package also makes it easy to load LaTeX packages, in particular for Chinese. So, our METAPOST file will begin as follows:

```
input latexmp;

setupLaTeXMP(class="article",
  packages="CJKutf8",
  preamble=(
     "\let\N\newcommand"
    &"\N\0{}\N\1{一}\N\2{二}\N\3{三}"
    &"\N\4{四}\N\5{五}\N\6{六}\N\7{七}"
    &"\N\8{八}\N\9{九}"
    &"\AtBeginDocument{"
    &   "\begin{CJK}{UTF8}{bsmi}}"
    &"\AtEndDocument{\end{CJK}}"));
```

This preamble will load the `CJKutf8` package, which is what we need for UTF-8 input. It then defines the commands `\0` (for void), `\1` (Chinese numeral 1), `\2` (Chinese numeral 2), etc., up to `\9`.

Then, we start an appropriate (for these characters) CJK environment at the `\begin{document}` hook:

```
\AtBeginDocument{
  \begin{CJK}{UTF8}{bsmi}}
```

and we close the environment at the end of the document:

```
\AtEndDocument{\end{CJK}}
```

2.4 Putting all the pieces together

We now have a definition of cell values, we can draw a grid, and we have macros for Chinese numerals. What's next? Well, we want to be able to do two kinds of things: draw problems, and draw solutions. For our purposes, a problem is merely an array of cell values with some cells being equal to 0. These 0s will be displayed as empty cells. In addition to this switch, we want to display the non-void values either with Hindu-Arabic numerals or with Chinese numerals.

Our coding of cell values makes this rather easy, because we will use (for instance) `4` for the Hindu-Arabic numeral, and `\4` for the Chinese numeral. So, care must be taken of this additional `\` when needed. The special case of 0 must also be considered, because the Hindu-Arabic numeral 0 must not be displayed, whereas the Chinese `\0` can be displayed, since it is void.

The '0' switch is handled with the `zerospace` macro. This macro takes a character `s` and replaces this character by a space only when it is 0 and when the output uses Hindu-Arabic numerals.

```
def zerospace(expr chinese,s)=
  if not chinese and (s="0"): " "
  else: s fi
enddef;
```

Finally, filling the grid is done with `fillgrid`. The first parameter is the name of the string array and the second parameter is a switch for Chinese or Hindu-Arabic numerals. `substring` is used to isolate the character of interest.

```
def fillgrid(text grid)(expr chinese)=
  for i=1 upto 9:for j=1 upto 9:
    pos(j,i,textext(if chinese: "\" & fi
         zerospace(chinese,
           substring(j-1,j) of grid[i])));
  endfor;endfor;
enddef;
```

The result is then as follows for the problem and the solution, with Chinese numerals:

五	三			七				
六			一	九	五			
	九	八					六	
八				六				三
四			八		三			一
七				二				六
	六					二	八	
			四	一	九			五
				八			七	九

五	三	四	六	七	八	九	一	二
六	七	二	一	九	五	三	四	八
一	九	八	三	四	二	五	六	七
八	五	九	七	六	一	四	二	三
四	二	六	八	五	三	七	九	一
七	一	三	九	二	四	八	五	六
九	六	一	五	三	七	二	八	四
二	八	七	四	一	九	六	三	五
三	四	五	二	八	六	一	七	九

3 Conclusion

This example demonstrates how straightforward the integration of Chinese and METAPOST has become. What remains to be done is to link these macros with a general problem solving algorithm for Sudokus.

◇ Denis Roegel
LORIA, BP 239
54506 Vandœuvre-lès-Nancy
FRANCE
roegel (at) loria dot fr
http://www.loria.fr/~roegel