TEX Output for the Future

Leslie Lamport

It seems clear that dvi files are a TEX idiosyncracy, and the rest of the reasonable world is going to adopt PostScript as the standard device-independent output format. TEX will be a lot more useful, and reach a much wider set of users, if it could produce PostScript instead of dvi output. This would also permit the development of standards for incorporating figures drawn with other systems into TEX documents.

What are the problems involved in doing this? The existence of dvi → PostScript converters indicates that there is no serious problem at the back end. Allowing the incorporation of other PostScript figures into a TEX document simply requires implementors of these converters to agree on a convention for the `\special` commands. Since I gather that there are now just two such dvi → PostScript programs, I urge their authors to agree on some standard that the rest of the world can use.

The more serious problem lies at the front end, with the fonts. A PostScript font comes with an .afm file to specify font metrics. TEX requires a .tfm file that has additional parameters needed to use the font in math mode. Consequently, as I understand the situation, one can use only CMR fonts in math mode. (I suspect that the .amr file also lacks parameters for the proper placement of accents.) It is my understanding that there are no PostScript versions of the CMR fonts; they must be printed by converting each character to a set of pixels, and drawing each pixel individually — a time-consuming operation.

A first solution to this problem might be a method of getting METAFONT to produce PostSript fonts with .tfm files. If METAFONT becomes the wave of the future, with lots of fonts being drawn with it, this will be a satisfactory solution. If, as I suspect will be the situation, METAFONT is ignored by most of the world, one will ultimately, want a method for producing .tfm files for fonts not produced by METAFONT.

The problem of converting TEX to the PostScript world is important to anyone who wants to see TEX survive. It seems to me that the current dvi → PostScript drivers are not a viable long-term solution. I haven't the expertise or the time to contribute much to a solution. However, I'd be happy to do what I can to act as a catalyst. There are a number of people out there who have a financial stake in the survival of TEX; I urge them to start cooperating now on solving this problem.

Software

WEB Adapted to C, Another Approach

Silvio Levy
Princeton University

I read with great interest a recent TUGboat article about a C version of WEB, by Klaus Guntermann and Joachim Schrod (October, 1986). Since I, too, have written such a CWEB program, I would like to share some of my experiences. I will concentrate on the differences between my version and Knuth's original Pascal version.

I start with TANGLE, since it is easier. I decided that TANGLE should respect the user's line breaks and insert `#line` statements into the C file, so that the compiler, debugger, etc., would print messages that refer to the CWEB file and not to the C file, which is difficult to consult. I am very happy with this arrangement, especially in terms of the debugger: I never have to refer to the C file.

Knuth endowed WEB with a macro command `@d` because the generic Pascal does not handle macros. But the C preprocessor has a standard and powerful macro capability, and between having (the traditional) WEB's treatment of macros and C's I prefer the latter option, because of WEB's limitation to one parameter and, even more annoyingly, because of the fact that in WEB you can't use a variable name (even in TEX text!) before defining it in the source file. This second constraint runs counter to the overall philosophy of WEB, which is that things should be introduced where they logically belong; e.g., an *error_message* macro should be introduced in the section that deals with error handling, and it may not be convenient, or even logical, to have that section in the beginning of the source file.

For this reason my version of TANGLE does not process macros; instead it transforms the WEB file's `@d` statements into `#define` statements that it groups at the beginning of the C file. Naturally, `#define` statements can still be interspersed in the C code, if for any reason they should not migrate to the beginning of the C file. This has the disadvantage that one cannot write macros with a variable number of parameters; but in my experience the gain in simplicity and uniformity far outweighs this drawback.

Finally, my version of TANGLE always inserts a blank space after an '=' token. This is because the C compiler, for reasons of backward compatibility,

interprets `x=-1` to mean "subtract 1 from x," which is very annoying. (The current idiom for this instruction is `x-=1`, and has been for over a decade.)

In order to "understand" input code and typeset it correctly, WEAVE's parser transforms it into a sequence of *scraps*. Each scrap has a *category* (or *cat*, in the lingo), which is like its part of speech; when several scraps with the right cats are found in sequence, they "fire" a *production rule*; for this reason I also call them *sparks*, a quasi-anagram of scraps. It turns out that C's syntax is different enough from Pascal's that I needed to rewrite the production rules from scratch. For example, WEAVE should distinguish between the use of '`*`' and '`&`' as unary or binary operators: in the common idiom `char **argv`; both `*`'s "belong" to *argv*, so the output should look something like

char ∗∗*argv*;

Here's what the TEX output of my version of WEAVE looks like:

```
\&{char} ${*}{*}\\{argv};$
```

(I'm thankful to Guntermann and Schrod for pointing out that this makes TEX treat the asterisk as an Ord atom, not as a unary operator; but then I tried making them Op symbols, and the output didn't look as good to me. Op symbols are meant for large operators, and things like log.)

Following the syntax definition of C (appendix A of Kernighan and Ritchie's *The C Programming Language*), I wrote a relatively small set of rules (fewer than in the original WEB) that correctly parses all C constructs, including variable and function definitions. (It can fail spectacularly when module names or macro names are used in unusual ways; then manual formatting is called for.) In addition all variables being defined automatically get an underlined entry in the index. This means that it is no longer necessary to insert `@!` by hand when certain variables are being defined; I only use `@!` in special circumstances.

In C, when you say `typedef double foo`, the indentifier `foo` can no longer be used to hold the value of a variable and it becomes syntactically equivalent to `double`. Thus WEAVE must give it the same syntactic treatment as a reserved word like `double`, and should also give it the same typographical treatment. Furthermore this should preferably be done automatically. Currently my version of WEAVE takes care of this by changing the *ilk* of the identifier at parsing time. This is not very elegant, and doesn't work if the **typedef** definition is in a separate file; but then one can use WEB's `@f` control sequence. There is also a new control sequence `@i` which works like `#include`, but actually does include the file into WEAVE's input.

Thanks to these changes, if I write

```
double inner_prod(vec1,vec2)
double vec1[dim],vec2[dim];
```

the variables *inner_prod*, *vec1* and *vec2* automatically get an underlined reference in the index; and if I write

```
typedef double vector[dim];
```

the word **vector** will from now on appear in boldface, and its "part of speech" becomes the same as that of **double**.

The last addition I made to WEAVE doesn't show in the output, but it simplifies the grammar a lot. In the original WEB sparks of certain cats can be printed in math mode only, others in either mode and others in non-math mode only. With the relatively more complex grammar of C this scheme would imply a great increase in the number of cats and of production rules. Guntermann and Schrod's solution (letter of December 11, 1986) was to typeset everything in math mode, and have the TEX macros for the various output tokens switch back to non-math when necessary (using the `\ifmmode` primitive). My solution is somewhat different: my sparks have a new attribute, their *mathness*, which is independent of their cat. When a production rule is fired, there is a special bit of code that inserts a '`$`' between sparks of different mathness, but the grammar itself doesn't have to contain any mathness information. This makes WEAVE run about 2% slower, but TEX'ing the WEAVE'd file is faster because TEX doesn't have to check the modes for lots of control sequences.

In conclusion, I am quite happy with CWEB, and do all my programming in it. CWEB itself is written in CWEB. Although I still consider the program experimental, I'm distributing it to interested people, and looking forward to comments and suggestions for improvements.

My heartfelt thanks to Klaus Guntermann and Joachim Schrod, for their helpful correspondence, and to Helmut Jürgensen and Barbara Beeton for inviting me to send this paper.