Design by Template in a Production Macro Package

Michael Downes
American Mathematical Society, 201 Charles Street, Providence, Rhode Island 02904 USA
mjd@math.ams.org

Abstract

The American Mathematical Society has been involved in the development of TEX from the beginning and began using it for journal production ten years ago; we now produce nearly all of our publications (a couple of dozen journals and book series) with TEX, using AMS-developed macro packages. One of the goals set for a major overhaul of the primary in-house macro package, begun in 1992, was to make revisions to the visual design of a given publication easier. In the new version of the macro package the design specifications for a particular document element (such as an article title) are not embedded in TEX code, but are entered into an element specs template that is comparatively easy to read and modify, and that corresponds more directly to traditional book design specs (e.g., vertical spacing is expected to be given in base-to-base terms).

Introduction

Some of the terms used herein have a specialized meaning in the publishing industry; two that should be mentioned in particular are *composition*, meaning the general process of composing characters into words, paragraphs, and pages (historically done by setting lead characters in type frames, nowadays done with software), and *design*, meaning the visual style and physical layout of a book or other publication, including choice of fonts, dimensions of the type block, and arrangement of document elements. Publishers employ freelance or in-house *publication designers* (more commonly known as *book designers*) to analyze authors' manuscripts and devise appropriate designs.

The transition of the publishing industry in the last few decades to working with electronic documents was impelled initially by the desire for more efficient production of traditional printed forms. It has become clear by now, however, that the electronic document should be the *primary* goal of authors and publishers; and moreover, that documents in information-rich formats such as SGML are many times more valuable than documents limited to a single medium and visual format. The challenge now for compositors is to make composition software that enables the composition process to be driven more directly by an electronic document, when the content and structure of the document are adequately marked. Given a visual design in suitable form, the typesetting operations to be applied can in principle be deduced from the information present in the document. If software can do this task, it will render unnecessary some of the expensive rote work traditionally done by human copyeditors when marking up manuscripts with instructions for the compositor. The copyeditor served as a sort of interpreter between the author and the compositor, who were not expected to understand each other's language. And the book designer may be thought of as the linguist who wrote the bilingual dictionary used by the copyeditor in doing the interpreting.

Book designers use a high-level, rather informal language that has evolved over the last few centuries together with printing technology and methods. By 'informal' I mean that traditional book design specifications aren't sufficiently detailed and well-structured to be directly interpreted by typesetting software, even after doing the obvious streamlining of vocabulary and syntax. In the past the work of translating book designs into suitable typesetting operations was done by skilled compositors who brought to the translation process a great deal of enriching knowledge and craftsmanship. In recent years, the computerization of typesetting has shifted more and more of that knowledge and craftsmanship into typesetting software.

But the state of the TEX world today is that not enough of the knowledge and craftsmanship has been transferred to the software. There is a wide gap between the customary design specs that publishers pay designers for, and the actual application of a design to the pure information content of a document by a TEX macro package. Outside the sphere of TEX, this gap is usually closed by idiosyncratic stylesheet capabilities of commercial publishing software ranging from word-processors to high-end publishing systems. The internal format used for holding style information, the internal typesetting operations used to apply the style information, and the underlying analysis of document design are proprietary information, and I imagine that not too many software companies are rushing to make theirs public.

The language used in FOSIs[1] is the only major, publicly accessible attempt I know of to formalize visual document design into a standard language suitable for automated typesetting. And as it is, FOSIs only deal with abstract specifications; you still have to make your typesetting software understand the elements of a FOSI and do the right thing with them. An extended discussion of using TEX and FOSIs together appeared not long ago in *TUGboat* (Dobrowolski 1991).

In the TEX world the gap between design and actual typesetting is usually bridged only by the brow-sweat of a skilled TEXnician. Consider these design specifications for an article abstract:

AB: Abstract. Body text 8/10 Times Roman justified × 27 picas, indented 1 1/2 picas, para indent 1em. Heading "Abstract" and period set 8/10 Times Roman bold, flush left × 27 picas, followed by N space, run in body text. 18 points base-to-base above and below.

As written in typical TEX macro code, the AB element described above might look roughly like this:

```
\def\abstract{\par
  \ifdim\lastskip<\medskipamount
    \removelastskip \medskip
  \fi
  \begingroup \parindent 1em
    \leftskip=1.5pc \rightskip=\leftskip
    \typesize{8}{10}\justify
    \noindent {\bf Abstract.\enspace}}

\def\endabstract{\par \endgroup
  \ifdim\lastskip<\medskipamount
    \removelastskip \medskip
  \fi }
```

In LATEX you could avail yourself of the `list` environment to simplify the task. But it remains clearly a TEX macro writing task, needing to be done by someone familiar with TEX.

Given that the limits of book design are scarcely less wide than the limits of human visual imagination, it's unlikely that custom programming will ever disappear from the picture; unprecedented demands will always require new solutions. Nevertheless, the majority of scientific and academic publications have scholarly communication rather than visual design innovation as their *raison d'être*, and hence are characterized by relatively sober designs with many routine aspects suitable for automation. Most typesetting software doesn't do as much as users would like to make the routine aspects easy to deal with:

> The frequent need for new or differently formatted entities presents a serious problem. It is typical, that either there are too many limitations on the kind of formatting that can be prescribed, or else the formatting prescriptions are very difficult to write and comprehend. — Bo Stig Hansen (1990)

TUGboat articles about the Lollipop macro package by Victor Eijkhout (1992), and the ZzTEX macro package by Paul Anagnostopoulos (1992), show something of what is possible with TEX.

The design-to-typesetting gap needs to be bridged from two directions: from the designer end, by a careful analysis of document design, leading to a formalized (to the extent possible!) language for prescribing design elements and rules, independent of any particular typesetting software;[2] and from the TEX end, by more powerful and flexible macros that match up with the design analysis (along the lines of LATEX's `\@startsection` and `\@hangfrom`). If these two pieces are done well, then the process of transferring a document design from the designer's mind to the typesetting software can be made very easy.

In this paper I discuss some methods being employed to bridge the design-to-typesetting gap in a large TEX macro package, developed over the past two years for use at the American Mathematical Society as the in-house, production-oriented version of the publicly available AMS-TEX package. As the new package has no established name (we've been just calling it 'the new production system') I suppose I'd better define a name of convenience for use hereinafter. Let's call it DBT: design-by-template system. Although there's more to it than the visual design-related features, that should serve well enough.

The goals of DBT, as compared to its predecessor, include:

1. improvements in document markup to enrich and regularize the information content of documents passing through the AMS production system;
2. more sophisticated formatting routines to solve fundamental TEX typesetting problems;
3. change management;
4. a 'fill-in-the-blank' system for specifying the visual design of a given publication through simple variable assignments rather than through TEX macro programming.

As an example of the second item, the page-breaking routines in the new macro package automatically allow a page to run half a line long or short to accommodate the deviation from the text grid that occurs more often than not in pages containing many

[1] FOSI: Formatted Output Specification Instance. Part of the U.S. Department of Defense CALS initiative, see Mil. Std. MIL-M-28001B.

[2] Here I echo the call for a 'front end programming language for style design' of Victor Eijkhout and Andries Lenstra (Eijkhout and Lenstra 1991).

```
\thm{9} The elegance of a mathematical
result is equal to the quotient of power
and length: $E = P/L$.\endthm
```

Figure 1a: Document source text for a theorem.

Theorem 9. *The elegance of a mathematical result is equal to the quotient of power and length:* $E = P/L$.

Figure 1b: Output of the given theorem.

```
[THM]description:{theorem}
[THM]typesize:{10}
[THM]linespacing:{12}
[THM]font:{it}
[THM]wordspacing:{\normalwordspacing@}
[THM]case:{\normalcase@}
[THM]justification:{%
  \fulljustification@{\colwidth}}
[THM]paragraphshape:{\indented@{18pt}}
[THM]breakbefore:{\badbreak@{1}}
[THM]spacebefore:{%
  1.5\linespacing plus.25\linespacing}
[THM]breakafter:{\goodbreak@{1}}
[THM]arrangement:{R head * . {\enskip} body *}
%    Subcomponent 'THM head':
[THM head]font:{bf}
```

Figure 2: Representative specs (pretending no defaults or inheritance) for a theorem element.

math formulas. We have also gone through some rather extensive experiments with nonzero mathsurround.

The focus of this paper is on the fourth item: How to make the process of creating and changing publication designs easier. It seems best to begin with an example, to serve as a point of reference for later discussion. Figure 2 exhibits a representative set of DBT specs for a theorem element. In practice many of the style variables would be given default values (such as 'inherit from context') by omitting the corresponding lines.

For comparison Figure 1 shows a portion of a document file and the output that would be produced given the specs in Figure 2. The main point is that the document file has only information content, not visual formatting instructions, and all the formatting specified in Figure 2 is applied automatically by the software.

Style variables. Here are descriptions of the style variables currently recognized by DBT for major text elements. It should be fairly obvious that this is *not* a universally sufficient set; rather, it is a set that seems to be more or less sufficient for the relatively modest design needs of most AMS journals and books.

description mandatory; used in printing documentation
placement floating or nonfloating
otherbefore catch-all hook (historically this has been used for making things work where the implementation hasn't caught up yet; thus tending to disappear later when the implementation does catch up)
typesize Self-explanatory
linespacing Self-explanatory
typeface Typeface name such as Garamond or Palatino
font One of: rm, bf, it, sc, bit, ...; for simplicity each style/weight/width combination is addressed by a single distinct name.
wordspacing name of a function that sets all relevant parameters
case upper, lower, normal
justification full, raggedright, raggedleft, centered, ...
paragraphshape indented, hangindented, ...
interparspace extra amount, usually 0
breakbefore bad break or good break; the TEX range 0-10000 is collapsed to 0-10
spacebefore base-to-base
actionbefore inner hook
arrangement how subcomponents are combined
actionafter inner ending hook
breakafter cf breakbefore
spaceafter cf spacebefore
otherafter catch-all ending hook

The set of variables is extensible in the sense that a new variable can be freely added for any element, and the assignment will be stored in proper form with the other specs for the element; it's just that you would also have to add some internal TEX processing to do the right thing with the new variable, in order for it to have any effect.

In addition to the above variables that are applied for elements within the text stream, there are a number of global variables that address page layout and other aspects. These are simply handled as TEX integer or dimension variables. Some examples:

```
\typeblockwidth    \linesperpage
\typeblockheight   \trimwidth
\runheadspace      \trimheight
\runheadheight     \headmargin
\textwidth         \guttermargin
\textheight        \dropfoliodepth
```

Features and Limitations of DBT

Concentration of visual design information in a single location. All the visual design and layout aspects of a given publication are kept in a single file with an extension of `.pds` (publication design specifications). This file is editable ASCII text, like TeX documents, with the same advantages and disadvantages. An added interface layer with menus, context-sensitive help, and *tsk-tsk* sound effects for bad design decisions would be nice but we haven't done anything of that sort yet.

Minimized redundancy. Publications with similar designs can share all the style variable settings that coincide by putting them in a common file to be 'inherited' via TeX `\input` statements. This is important for AMS use because we have several designs that differ only in a few aspects. If each design were kept as a separate full copy, maintenance would be more difficult.

Separation of visual style concerns from information transfer. A large part of some FOSIs that I've seen is taken up with specifying how certain information should be moved around (such as running heads or table-of-contents information). In DBT such information transfer is kept separate from visual style specs. In a `.pds` file there is nothing to say what information should go into the running head; only 'if such-and-such information happens to turn up, here is how to format it'.

Style template. The style of a given element is almost entirely specified in the element 'style template'—nothing more than a list of assignment statements for style variables—and is applied by a generic element-printing function. Little remains to be done by TeX macro writing.

A small number of generic element-printing functions. Essentially four generic element-printing functions are required—one for major elements (slices of the vertical text column), one for floating elements, one for displayed equations, and one for minor elements (distinct logical entities within paragraph text).

Predefined functions for subordinate typesetting tasks. Some of the variables in the element style templates are intended to take on function values: justification (ragged right, ragged center, full justification, etc.), paragraph shape (indented, non-indented, hang-indented, etc.), word spacing (default, 'french', loose). The idea is that a new function should be created whenever existing functions fail to provide the desired style in a form that can be easily called in an element template.

Compact notation for subcomponent handling. The 'arrangement' variable in the element style template is a special variable whose value is a description of the subcomponents (optional or mandatory) that an element may have, and how they should be combined. The notation is effective for concisely describing what is to happen when optional subcomponents are absent.

For the THM element in Figure 2, the arrangement is

```
R head * . {\enskip} body *
```

Each arrangement has seven parts. The first part is the arrangement name. `R` in this case is shorthand for 'run-in'; there are also `H` for horizontal and `V` for vertical. Less common arrangements have fully spelled out names. The second and sixth parts are the names of the subcomponents that are to be combined—here, `head` and `body`. The fourth and fifth parts are in-between material, loosely speaking punctuation and space, respectively. If one of the two components in an arrangement is optional, and absent in a particular instance, the in-between material is omitted.

Arrangements can be combined recursively. The third and seventh parts of an arrangement are slots for fitting in subordinate arrangements. A subordinate arrangement is enclosed in braces so that it can be read as a single macro argument by various arrangement-scanning functions. For example, suppose that we wished to allow an optional note component in the THM head, like the `[note]` option of LaTeX's theorem environments. The arrangement for THM would be expanded by replacing the * after `head` with a second-level arrangement:

```
R head {H mainhead * - {\ } note *}
  . {\enskip} body *
```

A hyphen for part four or five means 'null', in this case 'no in-between punctuation'.

Thus an arrangement is a sort of binary tree of subcomponents. Although restricting to binary combinations requires thinking up more component names than might otherwise be the case (viz the introduction of 'mainhead' above), higher-order combinations are unable to handle an optional middle component without ambiguity. Consider rewriting the above example as a three-way combination:

```
head R - {\ } note H . {\enskip} body
```

If the note is omitted, it's not clear which pieces of the in-between material should be used between the head and the body. Simple strategies such as 'use the first set of in-between material and ignore the second' (or vice versa) proved to be unreliable when I tried them with a range of real examples.

One current limitation of the arrangement notation has to do with list-like arrangements, where an element consists of an indeterminate number of identical subcomponents. An example might be a list of author names and addresses. Such arrangements are sufficiently binary in nature to have no ambiguity problems, but I'm not sure how to extend the current notation to handle them.

Arbitrary number of subcomponents for elements. Elements can be broken down as far as necessary to yield the desired level of independent control over fonts and other aspects of style for each component. A typical THM arrangement is slightly more elaborate than the one in the example given earlier, treating the word 'Theorem' and the number as separate components. The most complex element I've had to deal with so far consisted of four arrangements eight components.

By suitable combination of simpler arrangements, the design for a given element can become an arbitrarily complex two-dimensional structure.

Inheritance. If one element has nearly the same style as another, there is a way in a DBT template to specify that the element is 'based on' the other, and reset only the style variables that have differing values.

Variable clustering. Instead of having separate entries in the element specs template for every variable provided by TEX, and every variable added by the macro package, some of the entries reflect clusters of related variables. To get ragged-right you need just one line:

```
[XX]justification:{\raggedright{30pc}}
```

instead of many lines:

```
[XX]hsize:{30pc}
[XX]leftskip:{0pt}
[XX]rightskip:{0pt plus3em}
[XX]parfillskip:{0pt plus1fil}
[XX]exhyphenpenalty:{3000}
[XX]hyphenpenalty:{9000}
[XX]pretolerance:{200}
[XX]tolerance:{400}
```

I've vacillated about leaving `\hsize` as a separate parameter, perhaps under a different name. The reasons for folding it into the justification parameter are: (a) this corresponds well to the way justification is specified in traditional book designs; and (b) there is a check in the internal processing of the generic element-printing functions to see if a new value for justification is the same as the previous value; if so, the resetting operation can be skipped. If `\hsize` were a separate parameter, the check would have to test two variables instead of one to decide whether the skipping can be done.

Some aspects of style templates remain TEXnical. Although the syntax of element style templates has been intentionally deTEXified, towards the goal of making them accessible to nonTEXnicians, the example in Figure 2 exhibits some backslashes, curly braces, and (gasp) even @ characters. The main reason for this was convenience during the development phase. Further improvements to the syntax would not be that difficult but have not yet reached high enough priority.

There were two reasons for allowing private control sequences: first, it leaves open the door to enter arbitrary TEX code in the value of a variable (though in our experience so far this has not been needed as much as I expected); and second, it avoids name clashes for things like `\goodbreak` and `\uppercase` that seemed natural for certain values.

Another significant practical constraint was parsability. Currently all design specs are parsed and assimilated at run-time. Though it may not be obvious at first sight, behind the syntax shown in Figure 2 lie many rejected variations that would have made it much more difficult to write the routines that scan component names and variable values. For example, use of anything other than curly braces to delimit variable values would lead to various problems: if end-of-line is used to mark the end of the value, then multiline values become difficult to deal with; if parentheses are used as delimiters, then it becomes difficult to use parentheses in a variable's value; and so on.

Printing out written specs from the `.pds` file. The organized structure of element arrangements allows them to be traversed by a suitable function in order to print out a transcription (which comes closer than you might expect to traditional specs written by a human designer). Vertical spacing values are not only entered in base-to-base form, but also stored that way, so printing out the values does not require backward conversion. The application of the specified vertical spacing is highly accurate by virtue of complicated internal code, which is beyond the scope of this paper.

Black-box math. Math style is dealt with in a more-or-less black-box manner: Here is a whole math setup, take it or leave it. As a matter of fact, the knowledge necessary for high-quality math typesetting has traditionally resided more in the hands of compositors than in the hands of book designers, so the current sketchy treatment of math style in `.pds` files is wholly typical. Although it would be good to open up math style for easy access, the necessary work hasn't been attempted yet. Note that the variables needed to control math style are almost completely different from the variables needed for normal text elements.

Memory hogging. Compared to other macro packages DBT is a memory hog, easily going over 64K in TEX's main memory category before even starting the first page, not to mention loading any hefty extension modules like a table package. And this is after moving error message and help message texts out of macro storage onto disk. In general I followed the philosophy of the authors of the X window system: Write the software the way you think it ought to be done, ignoring the fact that available computer resources are strained to support it, and hope that the resources will catch up by the time the software reaches maturity. Some further routine optimizations can still be done but it would be premature to do them now.[3]

Slowness. And it runs slowly, because it's trying to be so clever and do everything the right way, rather than the expedient way. For example, the reason that `.pds` files are scanned at run-time rather than precompiled, is that this puts a part of the maintenance burden where it ought to be — on the computer, rather than on the persons who set up design specs; an extra compilation step would make development and testing more onerous. Rumors that we would get a super-fast new machine on which to run in-house TEX production have not yet been substantiated by putting it in our hands. At the moment, running on a not-too-shabby two-year-old Unix workstation (circa 20 Specmarks, 30 MIPS), a typical book run of around 300 pages may take more than two hours, as opposed to thirty minutes or so with the macro package that preceded DBT. (That includes some non-TEX overhead such as `dvips` processing, and with the workstation simultaneously serving other processes.)

Various Complications

In this section I want to describe some of the technical complications and problems that have crossed my path. A few of them are TEX-specific, while others are relevant for any system that seeks to automate the application of document designs.

(1) Intent: Style variable values should be carefully specified to reflect the designer's intent, as much as possible, rather than the results of that intent. For example, suppose that the designer calculates the point values for space around a section head to make the head occupy exactly two lines of space at 12pt linespacing, setting the `spacebefore` and `spaceafter` variables to 21pt and 15pt respectively. If the linespacing subsequently changes to 13pt, the values of `spacebefore` and `spaceafter` need to be updated by hand, whereas if they had been set to `1.75\linespacing` and `1.25\linespacing` then the change from 12pt to 13pt could automatically propagate as desired.

(2) When to evaluate: It's not always desirable to fully evaluate the value of a variable at the time of assignment. Immediate full evaluation makes it impossible for a dependent variable to keep in sync with another variable as it changes. For example, if the space above a section head is specified in relation to linespacing, and the normal linespacing value is overridden later to produce a double-spaced proof copy of a document, you would probably want the section head spacing to change proportionally. This won't happen if the spacing value was fully evaluated at the time of first assignment. Compare the well-known distinction in computer science between passing function arguments by value or by reference.

(3) Inheritance: When specifying the style of two similar elements, the one with a more complex subcomponent structure should be based on the one with a simpler structure, rather than vice versa. This is almost self-evident but I once had to set up four or five unnumbered footnote-type elements, and in my first attempt I thoughtlessly spec'd the normal numbered footnote first and based the unnumbered elements on that, before realizing it ought to be done the other way around.

(4) Inheritance: The question of immediate versus delayed evaluation for individual style variables applies also to collections of variables, when a major element is specified to be based on another. If the similarity of style is coincidental rather than due to a logical relationship between the two elements, then immediate evaluation would probably be desired.

(5) Documents that are nominally supposed to have the same documentstyle, in practice often don't. For a book series, the primary design for the series is typically subject to minor modifications in individual volumes. For example, in one volume the style of the footnote marks was changed because they too closely resembled some elements of the math formulas in the volume.

Similarly certain kinds of style variations are routinely permitted between different articles within a journal issue. In an article where a bullet • is used as a math symbol, it would probably be a good idea to override a standard list style that marks unnumbered list items with bullets.

Different authors prefer different numbering schemes, and because the numbering scheme of a document is closely bound up with the logical structure of the document, we routinely allow style for theorem heads and section heads to vary (within certain standard limits) to better suit the numbering scheme, rather than rigidly enforce a single numbering scheme. Here are three of the most common variations in theorem numbering:

Theorem 1.1. *Normal.*

1.2. **Theorem.** *Swapped numbers.*

1.3. *Numbers only.*

Numbers are typically swapped to the front when there are many numbered elements and different element types share the same numbering sequence. The font and intervening space then change for design reasons. All of the variations can be produced from identical markup by style override statements at the beginning of the document.

[3] "Premature optimization is the root of all evil." Donald Knuth, `tex.web` (version 3.14, 1991, §986).

(6) In fact, different articles in a journal frequently differ not only in style aspects but in the very elements that they contain. In a recent example, an article contained two different kinds of proofs instead of the normal one kind: proofs given in full, and brief sketches that left the details to be filled in by the reader. The two kinds were marked by two different QED symbols.

As anticipating all possible elements and forbidding unknown elements are equally impractical, the solution seems to be to make it easy to declare a new element and its associated style, and put those declarations into the individual document where needed.

(7) Arrangements in DBT are applied only to subcomponents within a single element. But documents may also contain sequences of major elements where some of the elements themselves are optional, which leads to the same sort of potential ambiguities as with subcomponent arrangements. Suppose the opening of an article consists of title, author, dedication, key words, subject classification, abstract, table of contents, and finally, main text. And suppose that the dedication, key words, abstract and table of contents are optional. It gets to be rather tricky to specify the vertical spacing to be used in all possible combinations. In DBT this is handled mainly by application of a single vertical spacing rule: when adding a major element to the page, compare the space after the preceding element and the space before the new element and use whichever is larger. Practically speaking this seems to suffice most of the time, if care is taken in choosing where the various space values are specified (e.g., for a given set of space values it may work out better to leave the `spaceafter` variable for the author element at zero and rely only on the `spacebefore` values of all the components that could possibly follow after the author). There is a rudimentary mechanism in DBT for specifying inter-element space depending on the type of the two elements, but it hasn't been needed much.

(8) Communication between information-handling and design-handling functions: In DBT a TeX command such as `\thm` is used in a document to collect the contents of a theorem. `\thm` sends this data to a generic element-printing function that takes the given pieces of information and feeds them into the declared arrangement for theorem elements. Thus the definition of `\thm` is interdependent with the arrangement. Ideally the definition of `\thm` could be derived in some semi-automatic way from the arrangement structure but there are many complications, so at present DBT doesn't attempt to be too clever; someone has to explicitly define all the components that `\thm` should look for, and the document syntax to expect. There are high-level syntax-related functions that make this task fairly easy, but the person doing the defining has to actually look at the declared arrangement when setting up the parallel structure in the `\thm` command, rather than having any of the transfer done automatically.

(9) Information content for the components of an element can be provided either explicitly in the document or by giving a default value in the element template. For example, a proof head will normally be given a default value of 'Proof' in this manner:

```
[PRF]arrangement:{%
  R head * . {\enskip} body *}
[PRF]head:{Proof}
```

In DBT there is a mechanism for optional substitution of alternate text for such a component, for selected instances of the parent element in a document. This is needed occasionally to get an alternative proof head such as 'Sketch of the Proof'.

It's not clear whether default information content specifications like this should be lumped together with style specifications. Is the material content or style? I would say content, but then consider the following ways of marking the beginning of a proof:

1. 'Proof' heading:

   ... preceding text.
   Proof. Text of the proof ....
2. Dingbat:

   ... preceding text.
   ¶ Text of the proof ....
3. Inline horizontal rule:

   ... preceding text.
   —— Text of the proof ....
4. Full-measure rule and whitespace:

   ... preceding text.

   Text of the proof ....
5. White space only:

   ... preceding text.

   Text of the proof ....

At what point does the material that marks the beginning of the proof stop being content and start being style?

(10) In math formulas font changes are mathematically significant. Suppose that an author uses bold and non-bold versions of Θ (Greek Theta) in a paper for different mathematical entities. Consider what happens in a bold section head:

4. Let's talk about Θ

The reader can't tell which version of Θ was intended. In more complicated examples serious garbling of the formula's meaning can occur. To prevent such garbling the AMS policy is to never vary the fonts in a math formula for purely stylistic reasons. (As a tangential benefit, this prevents practical problems with availability of bold or sans serif math symbols.)

(11) In traditional composition, certain minor points of style cut across the logical structure of the text and are therefore hard to handle by automated typesetting. For example, if the declared font for a theorem head is bold, then the logical thing to do for all in-between material *within* the theorem head, such as punctuation, would be to use bold. But it may be that one subcomponent within a theorem head (e.g. the number in the swapped-numbers example above) is not to be printed in bold. The result would be a bold period after a nonbold number, which tends to look bad:

1.2. **Theorem.** *Swapped numbers.*

(Look closely at the periods.) A bold period also looks odd following a math formula at the end of a run-in section head, since math formulas are not printed in bold in accordance with the AMS policy mentioned above. To prevent such distracting oddities, most manuals of composition style say that punctuation should take on the weight (and sometimes slant, if applicable) of the preceding text. The DBT system takes care of this, but not without effort.[4]

There is also the question, should the period be considered in-between material, or an interior part of the preceding component? I tend to believe treating it as in-between material is better, but the weight mismatch problem is one example of the complications lurking behind that approach.

(12) It's unclear how best to specify vertical spacing around displayed equations. Specifying it in base-to-base terms leaves open, for many equations, a question 'The base of what??'. And the reasonably satisfactory answer 'the base of the highest and lowest full-size entities in the equation, disregarding delimiters' is rather difficult to implement. Not to mention the fact that egregiously large sub or superscripts can still make a mockery of the intended spacing. The alternative of specifying the space around equations as visual space to the topmost and bottom-most points of the equation doesn't always work perfectly either. The mechanism built into TEX is a sort of composite method that uses base-to-base spacing until the equation contents get too tall, then switch to visual spacing. Although this works pretty well, it often results in a discrepancy of two or three points in the spacing above and below equations on the same page.

(13) In the presence of running heads, accurate calculation of type block height and text block height is not easy. Doing it properly requires knowing the alphabet height of the fonts used for the running head and the main text. Then placing the main text and the running head properly within the defined heights requires some careful work.

[4] A numeric code for the most recently used font is recorded in `\spacefactor` for later reference.

(14) 'Above display short space'. There is a mechanism built into TEX for automatically reducing the vertical space around a displayed equation if the horizontal distance from the end of the preceding text line is large enough.

Here we are testing a short skip.

$$\frac{B_1}{A}$$

Note the space below.

Contrasting to this test of the not-short skip.

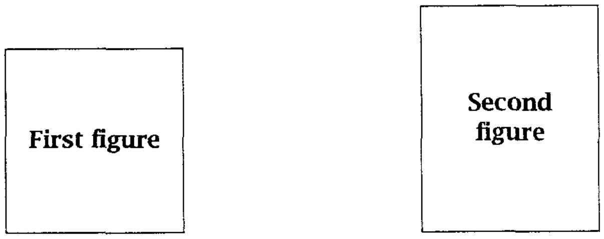
$$A = B + \frac{C - D}{E}$$

Note the space below.

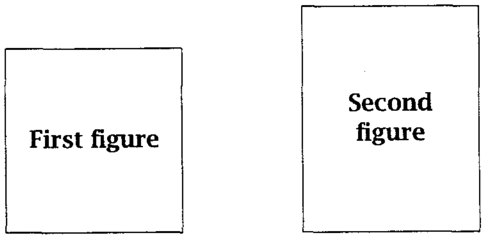
How should this design requirement be specified in a style template?

(15) Side-by-side figures: Proper specification of the surrounding space is a little tricky.

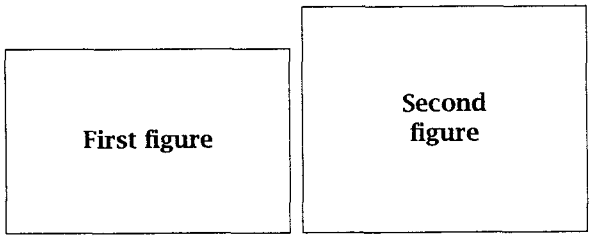
First try: put all the available space between the figures.



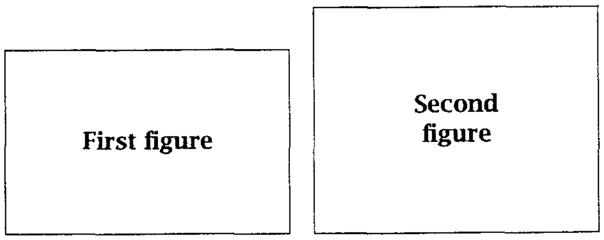
Second try: Divide the available space into four parts, put two parts in the middle, and one on each side.



But when the figures are close to half of the available width, the results can be poor:



Third try: Specify a minimum space between the figures, then divide up the space remaining into four equally distributed parts.



(16) QED symbol placement: The end of a mathematical proof is frequently marked by the letters 'Q.E.D.' or some sort of dingbat, as an aid to the reader. As the QED symbol is consistently applied to all proofs in a document, it seems best to treat it as an aspect of the proof style and add it automatically, rather than requiring it to be included in the content portion of the proof. To avoid wasting vertical space, the QED symbol is usually fitted into the bottom right corner of the last paragraph. But then if the proof ends with a displayed equation, or anything other than a plain paragraph, the formatting of the last text in the proof will probably be finished before the `\endproof` command kicks into action; retrofitting the QED symbol into the preceding text becomes a real problem.

(17) Omitting redundant punctuation: e.g., a period after a run-in section head that ends with a question mark; compare the omission of the sentence-ending period when a sentence ends with *etc.* (This is a problem in documents where sentences are marked up with `\sentence` ... `\endsentence`.)

(18) Running two different elements together. If a numbered list falls at the very beginning of a headed element such as a proof, it is common practice to run the first list item in on the same line as the proof head (saves vertical space, thus tends to save paper = cost = final cost to the reader). But the rule for deciding when to allow such running in is difficult to express in a way that can be automated.

(19) How to evaluate: For some style variables it is occasionally desirable to give a value in terms of other variables. But evaluation in TEX is problematic for any arithmetic expression more complicated than a factor multiplying a register. You cannot write

```
[THM]linespacing:{\currtypesize + 2pt}
```

The TEX way of writing such an assignment is

```
\linespacing = \currtypesize
\advance\linespacing 2pt
```

This can't easily be jammed into the value slot of a template entry. It would be possible to apply some sort of arithmetic processing when reading the value of a variable, but designing a syntax suitable for TEX would not be particularly easy.

(20) Case changing. TEX's `\uppercase` or `\lowercase` cannot be applied indiscriminately to an element component if the component might contain (for example) a math formula. Also, in AMS editorial practice the uppercase form of McLeod is McLEOD, not MCLEOD. For related TEXnical reasons, case changes in DBT cannot be applied to a composite component, only to 'atomic' components.

It would be better if case changes were combined with font changes — in other words, if uppercasing were done by switching to an uppercase font instead of by applying `\uppercase`. This would automatically avoid problems with embedded math formulas, for example. But the implementation details would be a bit thorny. You can make a virtual font that substitutes capital letters for the lowercase letters, but it seems sort of silly to create a separate font to access characters that already exist in the current font; so perhaps some sort of output encoding change would be better. But TEX 3.x doesn't provide that capability.

Conclusion

Most of the analysis in DBT for breaking down style specifications into suitable variables is straightforward. A few aspects, however — notably the 'arrangement' concept — have little precedent that I know of and have not had sufficient testing and analysis to raise their status beyond 'experimental'. The implementation in TEX of DBT approaches or surpasses some of the current typical limits on TEX memory resources and processing speed. Nonetheless, the system is currently in use in-house at the AMS and is doing a pretty good job of delivering the desired easy maintenance of our publication designs. The possibility of wider release at some point is not out of the question, but the amount of work necessary to polish it up for such release (including some optimization to decrease the strain on TEX memory capacities) would be rather large.

References

Anagnostopoulos, Paul. "ZzTEX: A macro package for books." *TUGboat* **13** (4), pages 497-504, 1992.

Bringhurst, Robert. *Elements of Typographic Style.* Hartley & Marks, Point Roberts, Washington, USA, 1992.

Brown, P.J. "Using logical objects to control hypertext appearance." *Electronic Publishing — Origination, Dissemination and Design* **4** (2), pages 109-118, 1991.

Dobrowolski, Andrew. "Typesetting SGML documents using TEX." *TUGboat* **12** (3), pages 409-414, 1991.

Eijkhout, Victor. "Just give me a Lollipop (It makes my heart go giddy-up)." *TUGboat* **13** (3), pages 341-346, 1992.

Eijkhout, Victor and Andries Lenstra. "The document style designer as a separate entity." *TUGboat* **12** (1), pages 31-34, 1991.

Hansen, Bo Stig. "A function-based formatting model." *Electronic Publishing — Origination, Dissemination and Design* **3** (1), pages 3-28, 1990.

Livengood, William P. *The Maple Press Company Style Book.* Maple Press Co., York, Pennsylvania, USA, 1931.