BIJLAGE N

Two faces of text

TEX as a programming tool for advanced document retrieval systems

Joop van Gent

Institute for Language Technology and Artificial Intelligence (ITK)
Tilburg University
PoBox 90153
Tilburg, The Netherlands
gent@kub.nl

Introduction

It is a common misconception to think that a scientific book about earthquakes is a book about earthquakes. If this would be the case the book could be replaced by a database containing the *information about earthquakes* exposed in the book. All scientific books about earthquakes could be replaced by one such database and — still more improbable —all books in the world could be replaced by one huge knowledge base containing all ideas exposed in all books. This is not the case.

A scientific book about earthquakes is also a book about the author, the time of writing, the origins of the ideas and linked ideas that are stored in other books. It's a book about other books about earthquakes and a book about itself. The book presents a lot of different sorts of information, the organisation of which is taken care of by historically developed and commonly accepted methods of referencing (bibliographies, index tables, thesauri) and methods of division (logically splitting up information in parts like nesting paragraphs in sections, sections in chapters, etc.). People find their way in libraries and books by search strategies based on these finegrained organisation systems.

The basis of the main ideas presented in this article is that — *as long as hard copy documents exist parallelly to electronic ones* — document retrieval systems should not only make use of these advanced organisation methods but should also have user interfaces that allow search strategies that people are acquainted with. Document retrieval systems should pair the advantages of fast search performance of computers with 'traditional' bumming around in libraries and 'sniffing' in books.[1] This implies that users should have electronic documents available in a typeset format that looks like a corresponding hard copy as much as possible. It also implies that these systems allow users to query the database in *natural language*.

For the former statement there are several arguments. The most important one is probably the following. In contrast with searching in large databases and quickly overviewing pages, reading large pieces of text is in many ways more comfortable on paper than on a computer screen. Evidently, hard copies are also more easily carried in a handbag. It is therefore to be expected that hard copies (especially large ones) will keep playing a role together with electronic documents for quite a long time. To allow a user to comfortably switch from hard copy to screen and vice versa is therefore a prerequisite for a document retrieval system. This can be achieved by an 'isomorphism' of typography.[2] Another argument is that besides using referencing and division a user often finds his way by *recognition of patterns* without really reading text, by just running a thumb through the pages of a book. This 'method' is evidently supported by having pages in the same typesetting on the screen.

I don't have strong arguments for qualifying interrogation in natural language as an important feature of document retrieval systems. Nevertheless I think document retrieval is one of the 'discourse domains' where a natural language interface is pretty efficient in comparison with alternatives like menu structured interaction or command languages. This is so, because the objects by their nature have an extremely deep hierarchical structure. Therefore menu structuring makes search actions last unnecessary long and command languages — impopular as they are anyway — too complex.

In this article I want to show the important role Knuth's typesetting language TEX — as defined in [6] — can play in the kind of document retrieval systems introduced above (from now on I will call these systems *advan-*

[1] And, of course, with the advantages of new strategies, like those developed in Hypertext(-like) environments (see [3] for a short introduction & survey).

[2] In some sense the physical sectioning of documents in numbered pages can be seen as part of the typesetting. Switching between hard copy and screen is evidently aided by a correspondence in paging and pagenumbering.

ced document retrieval systems). But before doing this I will give a more detailed description of these systems.

Advanced document retrieval systems

Until recently document library systems have been limited by the fact that most documents are only available in (good old) hard copy format. For this reason it was only possible to *automatically* retrieve information about a document by looking at its *description* in a database. These descriptions were (and still are) stated in terms of attributes like author, date, editor, title, keywords etc., like the features one finds on the little paper cards in a library catalog. In other words: only the library *catalog* could be automatized. Most conventional database systems are designed to query 'things' that are not really in the database. For example a database of second hand boats does not really contain second hand boats, it only contains *descriptions* of them.

Since more and more documents become available in (some) electronic format, it becomes interesting to develop library systems with databases containing electronic documents. What makes a database of electronic documents so special is that it does really *contain* electronic documents! Retrieval systems therefore can have the property of being able to *show* the objects in the database and we probably want to use this ability: we want to be able to treat a lot of queries by showing document components in some way on the screen. The system allows us to look *into* a document instead of just looking *at* it.

While containing documents, these systems allow a lot of new kinds of queries. For example queries about the semantic content of a document or about all kinds of properties of the *document components*. [3]

But how do we set up such a system? What exactly will be the objects in the database, what structure do they have and how should they be stored? And secondly: how should information be retrieved from them? To answer these questions we have to look at the 'nature' of documents and the way people find information in them.

Logical structure and search strategy

Documents basically have two types of structures: a logical structure (the organisation of document components) and a typographical structure (the way the document is actually presented on paper), both mirroring on a very abstract level the organisation of the semantic content of the document. In hard copy documents only the typographical structure is 'available', the logical structure has to be abstracted by the reader by pattern recognition.[4] For example chapters can be recognized by their highlighted titles, paragraphs by indentation etc. When we are looking for some information in a hard copy document we often make use of the logical structure of the document reflected in its typography.

It is this natural kind of search strategy that we would like to imitate in our retrieval system. An example will illustrate this. Suppose we have an electronic document about the sales of a certain series of computers and we want to know how many computers were actually sold. No system today will be able to retrieve this information directly from the document by *semantic interpretation* of the text (or figures). But if we know or presume that the information we are looking for is contained in some *table*, we can ask the system to look for a document component that has the property of being a *table*. In this way we can shortcut our search actions.[5]

How could we get the system to find a table? More generally, how can the system find document components? By extracting the logical structure from the typography like humans? Recognition of a logical structure on the basis of typography would request computable methods of *pattern recognition*. In addition to the wellknown problems in this field (algorithm complexity and noise, see for example the problems with current OCR systems) The many different kinds of typography used by different authors and editors make any attempt along these lines hopelessly futile. A better and faster way to achieve our goal is to explicitly impose a logical structure on a document.

In what way? It will be clear that for these purposes storage of documents in *plain text format* will not suffice.[6] We will need some *format* to express attributes of the components of a document. If — for example — we want to know the title of a particular chapter of some document, the system will have to know how to recognize a chapter and a title. This can be done by using a markup language.

The idea of imposing a logical structure on electronic documents by using a markup language is not new of course. SGML, ODA and, in a way, LaTeX have been designed for these purposes, although they were primarily intended for improvement of document interchangability (by separation of logical and typographical structure).

So documents need to be stored in two different formats — a logical structure and some typographical represen-

[3] I use the term 'document components' here to indicate the pieces of text (and figures) a document consists of, like paragraphs, sections, footnotes, etc. Document components are pieces of a document that form somehow a logical unit.

[4] And of course by semantic interpretation...

[5] Sometimes the attributes of document components themselves will be the final target of our query. For example we might want to know the exact title of a particular chapter without being interested in its meaning. Or maybe we want to list the particular components of a document that mention Shakespeare, without being interested in the contents of these components.

[6] With *plain text format* I mean a plain ascii format that contains no codes indicating the logical or typographical attributes of document components.

tation — in order to be able to treat queries about logical components that should result in showing typeset pages. If we drop one of them we are either not capable of representing figures or typeset text, or we have to make inferences about the logical structure by pattern recognition (as typeset formats do not contain codes indicating the logical structure information). Moreover if we we want to see documents in a format that *resembles a printed document as closely as possible*, we will probably want them to be sectioned in *pages*.

The role of TEX

If we choose for a pagewise presentation of typeset documents, the two 'faces' of the document now somehow have to be linked by a function assigning pages to document components. This is so because search actions will always take place on the logical face. Whenever a query is meant to get typeset information on the screen, the system will have to know where to find it. Therefore every component in the logical presentation of a document has to contain a *label* — or *adress* — indicating a position in the corresponding typeset representation. In many cases the typeset version is calculated from the logically formatted version on the basis of *font properties* and other graphical information, like grid, size of figures, style, etc.[7] But, as stated before, the typeset version never contains codes indicating the logical structure of a document. So what kind of computer program should take care of the linking? At this point TEX smiles at us.

Because it is the program that does the typesetting that also has to take care of the right pointers from logical document components to physical positions in the typeset version. If we would try to make this preparation module with an ordinary programming language like C or PASCAL we would have to solve typesetting calculation problems that took Donald Knuth ten years while implementing his TEX.

The TEX language and compilers were originally designed to bring professional typesetting within the reach of (scientific) authors. LATEX was meant to combine typesetting facilities of TEX with an advanced strategy of logically structuring documents, as is stated clearly by Leslie Lamport in [7]. I think most of the popularity of both products is (besides — of course — their quality) due to the elegance of this combination.

In my opinion TEX has another great merit: it can be used to elegantly *prepare* documents for databases of intelligent document retrieval systems as described above. The elegance is due to the parallel processing of semantic representations of documents and corresponding DVI-files. More precisely: The TEX compiler can be used to convert a document to a set of two files, one of which is the standard DVI-format, the other one is a file containing all information about the logical structure of a document and about the logical and typographical properties of the document components. But that's not all. The latter file also contains information about the *physical positions* of all components. With physical positions is meant the positions of the DVI-representations of these components in the DVI-file.

In the next section I will sketch briefly a TEX macro set that creates database objects from LATEX documents. This program has been succesfully implemented and can be extended to other types of document (input) formats like WordPerfect or SGML. The program is now incorporated in a prototype document retrieval system that can be combined elegantly with a natural language interface. I need to say at this point that I owe many suggestions and ideas for implementation of the TEX macro set to Huub Mulders, my ex collegue (and the best TEXnician I know) from the Computer Centre of Tilburg University.

A document preparation module in TEX

The main components of the system we designed are a document *preparation module* and a *document retrieval* module. The former is a *format translator*. It is meant to translate formats used by authors (e.g. WordPerfect, WordStar, LATEX) to formats that are convenient for retrieval. The latter is meant to translate queries of a user to operations on a database.[8]

The preparation module is a TEX macro set consisting of a parser and a generator. In the current implementation only LATEX documents can be parsed. From every document (besides the usual DVI-files and auxiliary files[9]) a set of two files is generated: an LST-file and an LSP-file, which will be examined in depth in the next section. For the moment I only want to make the following remarks about them. The former contains the document representation as a nested list of document components and also all text, the latter a set of statements about the sublists of this list (i.e. about the document components). Both files are in plain ascii format and can be directly interpreted by a Prolog interpreter.

Input and output routines

Only TEX's standard input routines are used. To be able to open extra files and write tokens to them I used TEX's `\openout` and `\write` instructions. To split up the information for the DVI-file and for both logical format files I used a recursive macro like:
`\eat#1 {#1 ...\swallow{#1}..... \eat}`.

[7] E.g. LaTeX , Ventura, Interleaf.

[8] At the moment, only a very limited set of query types can be treated.

[9] For those readers not acquinted with TEX: DVI files are typeset images. Auxiliary files are used for several purposes among which calculating logical references like table of contents. A nice introduction in TEX, though in Dutch is to be found in [4]

This macro reads strings from space character to space character. It takes care for both a direct 'consumption' of `#1` by TEX's standard typesetting mechanism and a 'rechewing' `#1` by a (recursive) macro `\swallow`. The latter reads `#1` token by token. Suppose for example `#1` is the string:

(i) `congres.\footnote{...}.This.`

Then `\swallow` makes:
`c o n g r e s \footnote {...} T h i s`
out of it. `\swallow` passes these 13 tokens to another set of macro's that splits up text and logical information.

It will be clear that the `\begin{document}` macro had to be adapted to initiate the `\eat` macro. To properly exit `eat` and other recursive macro's and to account for a proper nesting of document components in the LST file the `\end{document}` macro was also adapted.[10]

These redefinitions are very trivial:

```
\let\begindoc\document%
\def\document#1{%
   ......(instantiations)....\eat#1}%
\let\enddoc\enddocument%
\def\enddocument{%
   .........(handle nestings).....%
           \enddoc}%
```

For the interpretation of tokens routines were needed to check whether two tokens are the same. This is handled by the folowing macro:

```
\newif\ifsame%
\long\def\testsametoken#1#2{%
       \edef\partwo{\string#2}%
       \edef\parone{\string#1}%
       \ifx\parone\partwo\sametrue%
       \else\samefalse%
       \fi}%
```

Because of the token by token reading I needed a macro to append characters to a string. This could be done by the following macro:

```
\newtoks\ta\newtoks\tb%
\long\def\rightappend#1\to#2{%
          \ta={#1}%
          \tb=\expandafter{#2}%
          \edef#2{\the\tb\the\ta}}%
```

Another problem that had to be handled has to do with the linking of the logical and typographical image. Pagenumbers are generated at the beginning of a logical document component. As TEXnicians know TEX does a lot of recalculating at the end of document components. For this reason sometimes a wrong pagenumber was generated. This problem was solved by `\relax`-ing the pagenumber calculation:

```
\def\tolspfile#1{{\let\thepage\relax%
    \xdef\temppar{%
      \write11{\string#1}}}%
    \temppar}%
```

The rest of the macro set comes down to a partly recursive handling of LATEX's component delimiters.

The output format

It is often suggestes that documents should be stored in SGML format ,or a similar markup language type. From a document retrieval point of view this is not very attractive because it implies parsing of SGML documents while treating a query. We cannot avoid 'precompiling' SGML-documents to some other format that allows faster search strategies.[11]

I chose to take a format that has at least two attractive properties. First it is a format that can be *interpreted* by any proper Prolog interpreter and therefore it can also be compiled. Compiling this format results in a low level pointer structure that allows fast searching. Second it is a format that facilitates the interpretation of (natural language) queries. The latter point needs some further explication.

Interpretation of queries

As in any other query language the evaluation of a natural language query comes down to checking relations between objects in a database. In the case of documents and their components these relations are often very particular types of relations that have to do with the *logical positions* of document components, for example the *precedence* relation ($\alpha\ precedes\ \beta$) and the *sublist* relation ($\alpha\ is\_a\_sublist\_of\ \beta$). If we say — for example — that some chapter contains some footnote we say in fact that a list of a certain type is a sublist of another type.[12] What is so special about this? Well, nested lists (like for example *sets*) are in fact *algabraic* entities with a lot of welldefined properties. Basic operations on lists can be implemented in a very trivial way. The idea is then that we can facilitate the evaluation of queries when the database is a nested list and when we define the semantics of our query language in such a way that it can make use

[10] None of LaTeX macro's are redefined in the system files. All redefinitions take place in a distinct top level macro set in a trivial way by temporarily renaming the LaTeX instructions.

[11] A way to avoid parsing SGML bulk files is to make use of SGML's file pointer mechanism. This comes down to putting markups in files that are interpreted as pointers to other files (for details see [1]). In this way sets of files (document components) can be structured in any desired way. In this way we loose an advantage of SGML, namely its direct interchangability with electronic publishing environments.

[12] The term 'type' should not be taken literally here. It is to be discussed whether being a chapter or a footnote is to be taken as a type or as a predicate.

of the basic operations on nested lists when necessary.[13] One might argue that a lot of queries don't request mathematical operations on lists! I think this is indeed the case with queries about the *contents* of a document. A lot of queries however contain subexpressions (for example definite descriptions) that refer to document *components*. Part of the evaluation of the query therefore comes down to finding the correct referents for these subexpressions. Finding a referent on its turn implies an *extraction* of one or more elements from a list, which is in fact a mathematical operation.

EL/DR, a document representation language

EL/DR is a shorthand for *Ensemble Language for Document Representation*. The term 'Ensemble Language' refers to a type of semantic representation languages proposed by Harry Bunt [2]. EL/DR is defined within this framework. The language is used both for document representation and for the semantics of natural language queries. I will only introduce the sublanguage of EL/DR that is used for document representation here. For details about EL/DR the reader be referred to [5].

The language called EL/DR is based on the idea that documents are in fact *tree structures* or *nested lists* of logical components that are themselves also nested lists. For example, a chapter consists of a list of sections. A section consists of a list of subsections or paragraphs and so on. I call those components *DLS* (Document Logical Structure). *Words*, *figures* and (for the moment) *formula's* and *tables* are considered to be atomic. DLS's are syntactically represented in a trivial way. Consider the next (very small) document:

```
Yesterday I bought:
     - two appels
     - a beer
     - milk
     - some bread
```

Suppose we say that this document consists of a paragraph and an enumeration, the latter consisting of four items. This document can now be logically presented as a nested list as follows:[14]

```
[[Yesterday,I,bought],[[two,appels],
       [a,beer],[milk],[some,bread]]]
```

To avoid ambiguities — every word can occur more than once in a document — occurrences of words have to be labelled in a way different from the syntactic representations of the words themselves, for example:

```
[[w1,w2,w3],[[w4,w5],
          [w6,w7],[w8],[w9,w10]]]
```

In fact this is done in EL/DR, but from an implementation point of view it is more convenient to have labels for DLS's (nodes in a tree) too. Therefore, all nodes in a tree structure (atoms included) are implicitly numbered starting from 0 (the whole document) topdown, leftright depth-first, to n, the rightmost atomic element of the tree.

All properties of these DLS's can now very simply be stated in terms of predicate-argument constructions (or for those acquinted with Prolog, in terms of *horn clauses*). For example:

```
paragraph([w1,w2,w3]).
enumeration([[w4,w5],[w6,w7],
               [w8],[w9,w10]]).
item([w4,w5]).
item([w6,w7]).
item([w8]).
item([w9,w10]).
word(w4).
word(w5).
...
```

Or when using the implicitly imposed labels:

```
paragraph(1).
enumeration(5).
item(6).
item(9).
item(12).
item(14).
word(7).
word(8).
...
```

Chapter titles and crossreferences are two-place predicates, for example:

```
title(100,101).
crossref(221,457).
```

The first clause in this example states that the DLS labeled '101' is the title of the DLS labeled '100'. The second clause states that the DLS labeled '221' refers to the DLS labeled '457'.[15]

Finally, the linking of logical components to pagenumbers is done in a similar way, for example: `pagenumber(221,23).`

The first argument in this example is a label for a logical component, the second argument refers to a pagenumber, which is a physical position in the typeset image.

[13] If we think of representing documents as nested lists we can also make use of their properties in other ways. For example we can use them to *reason* about documents.

[14] For the sake of clarity interpunction is left out.

[15] Note that crossreferences are not seen as attaching *physical positions* in a text to document components, but rather as combining two (or more) logical document components.

Conclusions and final remarks

In this paper I have tried to show that TeX can be used to prepare suitable database objects for document retrieval systems and that alternatives must have particular qualities that are scarcely found in other programming languages. Current software supporting TeX has its limitations though. One of these is that not all graphic information can be represented in DVI-files. This is particularly the case with bitmaps. It seems therefore worthwile investigating alternatives like PostScript.

I have made a lot of presuppositions that are or should be subject of discussion, specifically:

1. advanced document retrieval systems should allow search strategies users have historically got acquinted with (but also — of course — new strategies),
2. hard copies will exist for a long time *together with* their electronic 'sisters'; for this reason a correspondence between the graphic representations of both sisters is desirable,
3. queries that make use of a document's logical structure play an important role in document retrieval.

Still I hope to have made clear that the preparation module for a document retrieval system should output documents as nested lists to make its sublists accessable by mathematical operations in the retrieval module. I also hope to have stimulated the TeX community to think about TeX's hidden qualities in the field of document retrieval.

References

[1] Martin Bryan: *SGML An author's guide to the Standard Generalised Markup Language*, Addison-Wesley Publishing Company, 1988.

[2] Harry Bunt: *Mass Terms and Model-theoretic Semantics*, Cambridge: Cambridge University Press, 1985.

[3] Jeff Conklin: *Hypertext; An introduction and survey*, in: Computer, september 1987.

[4] Victor Eijkhout & Nico Poppelier: *Wat is TeX?*, in: NTG Report 4, november 1989.

[5] Joop van Gent: *A formal language for describing document structures*, forthcoming, 1990.

[6] Donald E. Knuth: *Computers & Typesetting*, Addison-Wesley Publishing Company, 1986.

[7] Leslie Lamport: *LaTeX user's guide & reference manual*, Addison-Wesley Publishing Company, 1986.

BIJLAGE O

Towards LaTeX 3.0

Frank Mittelbach and Rainer Schöpf

Abstract

LaTeX is a very valuable tool for document composition. As a TeX macro package, it is unique in its concept of logical commands, at the same time retaining enough flexibility with visually oriented commands to allow the user a relatively easy correction of an automatically chosen layout. This fact makes it far superior to the plain and AMS-TeX macro packages when it comes to professional applications.

Therefore the LaTeX re-implementation project is certainly one of the most important efforts to "expand TeX's horizon". This is the place to link TeX with the modern developments like SGML. This paper describes the current status of the re-implementation of LaTeX.

Objectives of the LaTeX project

LaTeX [6] was developed to serve the specific needs and designs found in documents of natural science, whereas the needs of other fields are more or less neglected. Many layouts and concepts cannot be realized in the present LaTeX, and even those that can be realized are often falsely flagged as "impossible in LaTeX".

One of the main objectives of the LaTeX project is, therefore, to redesign the style file interface by incorporating a broader spectrum of possibilities. At the same time, we try to structure this interface in such a way that it satisfies the needs of a designer. This means that desired layouts should be specifyable preferably through parameters and generic functions that allow a wide range of varieties.

An ensuing objective is the proper documentation of the new interface. It should guide the designer in the evaluation of a new layout, allowing him to use the full power of the interface within a short period of time.

As a third objective, we feel it necessary to re-evaluate LaTeX's internal concepts and reverse those that have been proven inadequate.

In our talks at last year's conferences [8, 9, 10, 11], we presented a concept for a re-implementation of LaTeX. After discussing this topic, Leslie Lamport and one of the authors (FMi) agreed on a two-step procedure, first redesigning the style interface, and then enhancing the user interface.[1]

Further discussion throughout this year has shown that this plan is not feasible as the internal style file interface is affected too greatly by enhancements in the user interface, and vice versa. Therefore, we abandoned this idea and decided to merge both steps, at least temporarily.[2] As a result, the discussion then focused on three different major topics, the enhancements and changes to the user interface, the revision of the style file interface, and the re-evaluation of internal concepts.

In the following sections, we will discuss the topics which we have been concerned with since last year's conference and the state of their realisation.

The User Interface

Any change to the user interface of an existing program always opens the question of compatibility. While we judge this question as very important, we feel that it is not justified, for the sake of upward-compatibility, to leave all existing features untouched, even when they have proven to be inadequate. This means that we try to keep this sort of changes small, and devise a possibility to emulate LaTeX 2.09 in the new LaTeX to allow processing of older documents with no changes or only a few.

Attribute concept

It has been generally agreed in the ongoing discussion that an attribute concept as supported by DCF GML [4, 5] and SGML [3] would be a great improvement. Since this is a major change, it was discussed whether such a concept would render the optional arguments obsolete.

Reprinted from TUGboat 12 (1991), No. 1 – TeX90 Conference Proceedings © 1990, TeX Users Group; reprinted with permission.

[1] This was published as the update section in [8] and [7].

[2] We feel that it is still sensible to defer certain parts of the revision to a later stage. However, this will only concern internals of the implementation and not induce any changes to the user or the style-designer interface.

But as the number of LaTeX users is far greater than most people think, such an incompatible change in the syntax is not feasible. In additon, the old optional arguments and star forms provide convenient short forms for the most important attributes. While there have been several proposals for an attribute syntax as well as one prototype implementation, a final decision has not been made as yet. But it seems probable that this concept will only be available for environments, not for ordinary commands. We think that this will be sufficient since it is planned to provide environment forms of all text producing commands (cf.).

Robust and fragile commands

The distinction between robust and fragile commands will no longer be present. Instead, all text in moving arguments will be automatically protected against expansion.[3]

To allow the style file writer the specification of text that has to be expanded, there will be a command that removes or partly removes this protection.

We do not consider it necessary to give this feature to the user. Hence, this is not available in the user interface.[4]

Font selection

The new font selection scheme is already being distributed as beta test version for LaTeX 2.09 and seems to be working very well. It is also the basis for the `amsfonts` style option that makes the AMSFONTS collection available for LaTeX and is part of the AMS-LaTeX distribution [1].

Nevertheless, the current implementation should not yet be regarded as the final product. Time and users' demands will show whether it has to be improved.

Front matter

The specification of preliminary material is one of the parts which are not handled properly in LaTeX 2.09. Although this topic has not been discussed in depth so far, this might be one of the places where the syntax of the new LaTeX might differ in an incompatible way.

Tables

The extensions of the `array` and `tabular` environments by Frank Mittelbach [12] seem to be widely accepted. Several further extensions are conceivable, but this needs careful evaluation. There is also a new implementation of these environments by Denys Duchier that includes the extended syntax. This implementation looks very promising and can probably serve a basis for table handling in the new LaTeX.

We will provide a command to specify notes to tables working similar to the `\footnote` command inside a `minipage` environment.[5]

Math

The `amstex` style option for LaTeX 2.09 implements most of the features of AMS-TeX in LaTeX syntax (such as `\begin{align}...\end{align}` instead of `\align...\endalign`). As this is now being distributed by the AMS, users are able to typeset complicated math formulas in LaTeX without falling back to plain TeX's idiosyncrasies [1].

However, the implementation still has some loopholes that need to be eliminated in future versions.[6]

Text producing arguments

All commands with arguments in which the user specifies text to be typeset (e.g., `\fbox`) will also be available in an environment form to allow their use in user-defined environments.

Verbatim input

The use of the `\verb` command will be possible in all circumstances.[7] A similar extension of the `verbatim` environment is not possible.[8] But this restriction is lessened by the possibility to use the environment form of the respective command in which the `verbatim` environment may be used.

Float positioning

LaTeX 2.09 was designed for documents containing only relatively few floats.[9] The new implementation will improve the float position algorithm and the user's control.

[3] The approach of LaTeX 2.09 to expand everything by default is counter-intuitive and a common source of nasty errors.

[4] Expansion is normally necessary to cope with problems presented by the asynchronous output routine mechanism together with macros that change their contents, so that it is essential to write the expansion and not the macro name to a file, etc.

[5] Using `\footnotemark` and `\footnotetext` commands inside a table that (additionally) has to be put inside a `minipage` environment is another counter-intuitive concept of LaTeX 2.09.

[6] Some features do not work correctly in boundary cases. This is partly due to limitations in the current LaTeX, e.g., the primitive handling of `\begin...\end` (see below), etc.

[7] However, due to limitations of the TeX program itself, the use of multiple blanks in one `\verb` command will not be supported in all cases.

[8] In LaTeX 2.09 neither the `\verb` command nor the `verbatim` environment may be used in arguments.

[9] If, for example, the space for floats is larger than the surrounding main text, as is often the case in appendices of manuals, etc, LaTeX 2.09 is seldom able to compile the document without running out of memory space, even if the float parameters are given full flexibility.

This might include some sort of an 'h' option that really means "here".[10] There have also been proposals to prohibit the use of multiple captions within one float, thus allowing the document style to position the caption according to its own rules. But these topics have not yet been discussed thoroughly enough to present a final concept.

Bibliographies

The handling of citations and bibliographies will probably change to support several conventions. Since this topic depends on the development of the new BIBTeX to some extent, it is not yet clear what is actually implementable.

Specific problems concerning citations and the interaction with BIBTeX are discussed in [14] and in [13].

Omitting environment end tags

The implementation of a prototype for error recovery in case of unmatched `\end` tags has shown that it is possible to implement the concept of implied `\end` tags. This feature will help in writing SGML parsers that produce LaTeX output. Whether the detection of implied `\begin` tags, e.g., the omission of the first `\item` in a list environment, can be easily implemented requires further testing.

The Style-Designer Interface

As we stated above, the main goal for the design of the style file interface is making it easily applicable. Therefore, the new interface will contain a lot more generic commands allowing for the specification of a wide range of layouts with a minimum of effort.

International language support

Support for more than one language (US English) was one of the key issues that triggered this LaTeX reimplementation project. In the new implementation, all textual representations in style files will be settable. While this is certainly not sufficient to support the different typographic conventions of different countries, it allows for typesetting foreign texts within the usual typographic conventions.

Hooks

For the implementation of certain layouts, it is often necessary to carry out specific actions at well-defined points, e.g., the footnote placement algorithm of this article has to be initialized at `\begin{document}`. For this type of applications, many of the internal commands will contain hooks that allow the style file writer to add code to these commands without overwriting the original definitions.[11]

Generic section headers

One goal for the style file interface is to provide the designer with a generic heading macro which implements a broad range of layouts by varying certain parameters. It is clear that a proper balance has to be found between the internal complexity of such a command and the number of different layouts which are specifyable through it. Several different syntax proposals have been discussed so far, but the discussion hasn't reached a satisfactuary conclusion, as yet.

The new mechanism will probably provide a specification for headings, in which the designer has complete control over heading layout, e.g., positioning supplied text[12], ornaments[13] and the like. It is planned to support the following general types of headings:

- Vertically oriented headings, where the heading is separated by white space from preceeding and following text.[14] There will be parameters to allow the heading to extend into one or both margins.
- Horizontally oriented headings, where the heading is partially or fully placed into one of the margins beside the following text.[15] A critical problem with this sort of layout is to guarantee that enough space is available and that any necessary hanging indention (for headings placed only partly into the margin) is applied up to the necessary point.
- Run-in headings, where the text following continues on the same line as the heading.[16]

The designer will be given tools for specifying the heading layout in an easy manner, so that it is possible to vary the layout depending on things like the length of the heading text, the presence or absence of a heading number, etc.

It is planned to allow for the specification of a minimal amount of text that has to follow a heading.[17]

[10] We are aware of the problem of handling previously deferred floats of the same kind.

[11] In LaTeX 2.09 a lot of style options are incompatible with each other, simply because they redefine the same internal macro.

[12] For example, 'Chapter' in the `\chapter` command.

[13] Rules and dingbats, etc.

[14] This layout has already been realized to a certain extent in the `\@startsection` command of LaTeX 2.09. But this generic macro is not able to specify the layout of the heading (except setting the used font), so that it can not even be used for standard headings like the `\chapter` command.

[15] This sort of layout is not supported in the current version.

[16] Again, this layout is provided in LaTeX 2.09 but does not allow specifying the layout of the heading, e.g., punctuation marks at the end, underlining, etc.

[17] In LaTeX 2.09 this is the fixed amount of two lines of text. It might be possible that a more general implementation will fail

Generic table-of-contents entries

What has been said in the last subsection about generic section headers applies to the formatting of entries in the table of contents to as well. This has not yet been discussed in detail, but we hope that a proper implementation of generic section headers can be used as a starting point for generic toc entry formatting. The same mechanism can then be used for other types of tables as well (`lof`, `lot`, etc.).

Generic lists

Necessary extensions and corrections of the generic list environment have been already discussed in [8, 11]. There have been a few proposals concerning lists, but this topic needs further attention, as it affects a major part of most style files.

Parameter tables

There has been a proposal to group certain parameters into tables to make their structure and dependence visible. One item that will probably change on the style designer level in this way is the specification of default parameters for different levels of lists. But it is not yet clear, whether such a concept is implementable in an efficient manner.

Paragraph design

The specification of paragraph layout (such as different forms of ragged right typesetting) will be improved. The designer will be given the possibility to specify such layouts not only for the main text, but also for footnotes, floats, etc.[18]

Toc levels

There are book designs which require several tables of contents, e.g., one for the whole book and those refering to each chapter. This touches on the topic of auxiliary file handling (cf.) since the current mechanism does not allow more than one table per document. There will be support to select only certain entries of a table of contents for printing. A prototype implementation for this feature was written by Nico Poppelier.

Documentation

The interface between the style files and the LaTeX kernel will be documented properly. We will provide a complete description as well as a number of examples.

Internals

Error recovery

In the current LaTeX, an omitted or misspelled environment name usually produced a lot of error messages and often supressed any further compilation of the document. In the new implementation, certain classes of environment errors are detected and corrected without damaging the output. For this complex, a prototype implementation is currently undergoing alpha testing. It implements the following features:

Incorrect `\begin` tag If the user misspells the name of the environment desired inside the `\begin` tag, an error message is generated and the offending `\begin` tag is ignored.[19] However, the user is allowed to insert the correct environment name by specifying `i\begin{`⟨*envir*⟩`}` in response to the error message.[20]

Incorrect `\end` tag When an incorrect `\end` tag is encountered, e.g., `\begin{bar}` ... `\end{foo}`, the new LaTeX tries the following recovery:

1. If `\end{foo}` is unknown, we assume that the user misspelled the name and recover by replacing `\end{foo}` with `\end{`⟨*curenvir*⟩`}`. This will produce an error message but will allow safe continuation of the compilation afterwards.
2. If `\end{foo}` is a legitimate `\end` tag, i.e., if the corresponding internal environment start command is defined, we check to see whether there are any unresolved `\begin{foo}` environments.
   a. If so, the currently open environment is closed by inserting an `\end{`⟨*curenvir*⟩`}` tag.[21] Afterwards `\end{foo}` is tried again. This mechanism will close all open enviroments until the correct one is found. Depending on the status of an internal variable, this will either produce an error message, or a warning message, or will be executed silently to allow for the implementation of implied `\end` tags.
   b. If there is no open `\begin{foo}` we simply ignore the `\end` tag after issuing an appropriate error message. The underlying idea is that this `\end` tag is probably left over after moving some text in the source around.

in special cases due to TeX limitations.

[18] In LaTeX 2.09, this is not possible without redefining several internal macros, because the paragraph shape parameters are reset at several points to fixed defaults.

[19] Of course, this mechanism will be triggered only if the user did not exchange one environment name for another.

[20] In LaTeX 2.09, this response would result in TeX error messages at the end of the compilation.

[21] The implementation of this part of the recovery is not as straight forward as it may seem. The term ⟨*curenvir*⟩ does not necessarily refer to the innermost open environment because it may be possible that a user defined enviroment calls other enviroments in its body. In such a case, the calling environment has to be closed first, because the inner environments are resolved in its `\end` tag.

Auxiliary file handling

We will implement a two-step approach for `.aux` files where all information is written to an intermediate file which is then copied to the 'real' `.aux` file at the end of the run. As a result, there will be only one `.aux` file instead of the many files produced by LaTeX 2.09 when the `\include` command is used. The new scheme will make it possible to preserve cross-reference information if a compile run ended prematurely.[22] In addition, it will be possible to detect whether `\include` files have to be re-compiled because of changes in other parts of the document, or not.

References

The use of symbolic references will be extended to include textual references, if desired.[23] Whether this will result in some changes concerning the user interfaceor not has not been discussed so far. If it is feasible, we will also provide the possibility for hierachical references.[24]

Page selection

To provide easy access to the different parts of a document at the printing level, we plan to record certain document structure information in the `\count` registers 0–9. Besides the usual page counter in `\count` 0, this might include the physical page number, the current chapter, section, or subsection, . . . number, to allow the printing of, e.g., Chapter 6 or "all preliminary pages"[25] by giving a simple page selection pattern to the printer driver.[26]

Plain TeX compatibility

We tend to build up the new LaTeX from scratch, i.e, not to read in the plain TeX format (or a nearly identical variant) as a basis when building a format file.[27] This does not mean the useful functions, `\mathchar` definitions, etc. of `plain.tex` will be discarded, but concepts which are obsolete in the LaTeX environment or macros that can be implemented in a better way will be replaced or removed.

Beta testing

It took LaTeX about three years to develop into a stable system. To avoid needing a similar period of time when switching to the new version, we plan to run the new version throughout the development at a few selected sites for beta testing. So, if you are a maintainer of a TeX installation and think that you can persuade your users to play willing (or unwilling) guinea pigs, please, contact one of the authors. Your installation should have the following characteristics:

- A running TeX 3.0 preferably (but not necessary) with drivers supporting the virtual fonts introduced with TeX 3.0.
- A fair amount of LaTeX processing in a fully supported LaTeX 2.09 environment (including a customized Local Guide) and at least one user with some experience in style writing.
- A working eMail connection.
- A maintainer (you) who is willing to
  - provide backups and fast user support in case something goes wrong
  - send in bug reports, if necessary
  - compile new formats when updates arrive.

The work that has to be carried out by the beta testers should not be underestimated. It is probably a time-consuming commitment since a lot of organizing is usually involved. Nevertheless, we hope that there will be enough people around willing to help us in this stage of development, so that we can finally return a product to the user that will have the same success as the current LaTeX had.

Throughout the beta testing phase, LaTeX will have to show its abilities in real life situations. As TeX and LaTeX are currently finding their way into new areas, we hope that people from these fields will take the opportunity to test whether the new LaTeX matches their needs by participating in the testing phase. This is the time when it is still possible to correct errors before they become established as unfortunate facts.

References

[1] American Mathematical Society, Providence. *AMS-LaTeX Version 1.0 User's Guide*, July 1990.

[2] Brüggemann-Klein, Anne, editor. *1989 EuroTeX Conference Proceedings*, 1990. To appear.

[22] The current implementation writes directly to the `.aux` file, so that its contents are damaged or lost when an error occurs.

[23] In the current LaTeX, references can only be made to counter values, or more exactly, to a combination of counter values. This will lead to problems if document styles decide to use unnumbered headings, for example.

[24] In LaTeX 2.09, this has been realized for the `enumerate` environment, to some extent.

[25] In LaTeX 2.09, it is often not possible to print a specific set of pages that have the same TeX page numbers as other pages that come earlier in the document.

[26] With most printer drivers, the first page to print can be selected by specifying a pattern to be matched against TeX's `\count` registers 0–9. We propose that drivers also allow for the specification of a pattern for the pages to be included.

[27] Currently, LaTeX is built on top of `lplain.tex` which differs from `plain.tex` in only a few places. This means that even the funny keyboard support (for SAIL terminals) is included that will give you in certain situations a '$\oplus$' character if you key in `$^^M$`.

[3] Bryan, Martin. *SGML: an author's guide to the standard generalized markup language.* Addison-Wesley, Woking, England; Reading Massachusetts, second edition, 1988.

[4] *DocumentComposition Facility: Generalized Markup Language Starter Set User's Guide.* New York, fifth edition, May 1987.

[5] *Composing Documents with the Generalized Markup Language.* International Business Machines Corporation, New York, second edition, March 1988.

[6] Lamport, Leslie. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1986.

[7] Mittelbach, Frank and Rainer Schöpf . "A new font selection scheme for TeX macro packages—the basic macros." *TUGboat*, 10(2):222–238, July 1989.

[8] Mittelbach, Frank and Rainer Schöpf . "With LaTeX into the nineties." In Thiele, Christina, editor, *1989 Conference Proceedings*, volume 10#4 of TUGboat, pages 681–690. TeX Users Group, December 1989.

[9] Mittelbach, Frank and Rainer Schöpf . "LaTeX dans les années 90." *Cahiers GUTenberg*, (6):2–14, July 1990. French translation of [11] and of some parts of [10].

[10] Mittelbach, Frank and Rainer Schöpf . "LaTeX limitations and how to get around them." In *1989 EuroTeX Conference Proceedings*, Anne Brüggemann-Klein [2].

[11] Mittelbach, Frank and Rainer Schöpf . "With LaTeX into the nineties." In *1989 EuroTeX Conference Proceedings*, Anne Brüggemann-Klein [2].

[12] Mittelbach , Frank. "A new implementation of the array–and tabular–environments." *TUGboat*, 9(3):298–314, December 1988.

[13] Read, David. "Towards BIBTeX style-files that implement principal standards." *TeXline*, (10):2–8, May 1900.

[14] Wonneberger, Reinhard and Frank Mittelbach . "BIBTeX reconsidered." In Guenther, Mary, editor, *TeX 90 Conference Proceedings*, page ?, ? 1990. Cork September 1990, to appear.