Occam's Razor And Macro Management

Laurent Siebenmann

Université de Paris-Sud
Matématiques, Bâtiment 425
F-91405, Orsay, France
Email: `lcs@topo.matups.fr`

Abstract

The philosophical principle known as Occam's Razor asserts that entities should not be multiplied beyond necessity. The TEX utility *Occam* is a tool to eliminate from a collection of supporting TEX macros (composite commands) those that are unnecessary in a given typescript. Hopefully, it will serve to (i) let Plain TEX users produce typescripts which can be electronically posted in a compact form that is nevertheless autonomous and perfectly archival, and (ii) to simplify a macro package before making modifications for a special purpose.

The *Occam* utility will ultimately be programmed entirely in TEX language to assure that it is universally available. Today it is just an evolving prototype implemented with a bit of help from an editor (on Macintosh) that has a programmable control language based on *grep*.

To achieve reasonably *automatic* functioning of *Occam*, not requiring surveillance by a TEX programmer, it is necessary to maintain a carefully structured master version of each macro package involved; this ".occ" version can double as the documented source version of the package.

Key words: Occam's Razor, macro management, Plain TeX, ".tex" typescripts, electronic publication, docstrip, LATEX.

1 The Aphorism

At this European congress, let me remind you that the English philosopher William of Occam (or Ockham) was, like Abelard or Erasmus, a consumate European; he worked successively in Cambridge, Avignon, and Münich.

Occam's Razor is

entia non sunt multiplicanda praeter necessitudinem
entities should not be multiplied beyond necessity
William of Occam 1285-1349(?)

Experts believe that Occam did not formulate it in exactly these famous words, but rather as

> *What can be done with fewer assumptions*
> *is done in vain with more.*

or

> *Plurality is not to be assumed without necessity.*

The Razor is sometimes called the 'Principle of Parsimony'.

2 Introduction

Have you ever felt guilty about burdening a friend with macros that are not really necessary for composing your typescript? I certainly have; and would ideally like to follow Knuth's example of using macro files that define exactly what is necessary for a document and nothing more.

However, pruning a macro file that has served for other purposes is a pain. Most of us respond to this pain by adopting a rather messy maximalist approach in which all the macros that have a genealogy related to the necessary macros are transmitted.

But there is another approach! One can seek efficient mechanisms to ease the task of weeding out unnecessary macros.

One such mechanism is auditor.tex, which makes a list of names of those macros of macro file that turn out to be unnecessary in a given typescript.

A complementary tool is *DefStrip*. This utility exploits a specially arranged *Occam* version of the macro file to be cleaned up, in conjunction with the list of unused macros provided by auditor.tex in order to delete the macros listed (and some annexed material).

Ultimately, *DefStrip* will hopefully be a ".tex" program "defstrip.tex" resembling the "docstrip.cmd" utility of LaTeX fame. Today, there exists only a QUED/M command script called "DefStrip-QUEDCmds". (QUED/M is an inexpensive editor with convenient composite command capabilities, its own 'macros'. It is available on Macintosh computers, distributed by Nisus Software Inc. of Solano Beach Calif.) Methods suitable for programming "defstrip.tex" are described in [2].

Auditor and *DefStrip* together make up the system called *Occam*.

Let us consider two classes of situation where the *Occam* system will be useful.

(A) Weeding one's personal macro files

Many TEX users build up a cumulative personal macro file through composing many articles with TEX. A time inevitably comes when it is embarrassing, cumbersome, or confusing to submit (or post electronically) the whole macro file along with the article. The *Occam* system makes the pruning of the macro file painless. It is necessary to tidy up the total macro file and maintain it with '*Occam* structure', which is usually distinguished by the extension ".occ"; this structure will be described in §4. Then, and only then, will "auditor.tex" and *DefStrip* collaborate to *automatically* produce a minimal version of the macro file suitable for the article at hand.

(B) Preparing autonomous and archival ".tex" postings

Suppose that one proposes to post in electronic ".tex" form an article prepared using a remarkable but not really standard package such as the "harvmac.tex" macro package of Paul Ginsparg for Plain TEX.[1] Such a macro package is not immune to alteration with time, and unfortunately the principles of upward compatibility are just pious hopes, not laws. Consequently, one is well advised to post, along with the ".tex" version, the macros necessary to compile it — especially if modifications to the macros have been used. Unfortunately, the "harvmac.tex" macros are as voluminous as a 10 page article. This is an unfortunate obstacle to electronic posting of shorter".tex" typescripts.

The solution proposed requires "harvmac.occ", which is "harvmac.tex" macros set out in a form designed for use with *Occam*. Then the necessary macros for a given article can be extracted by *Occam*. This is illustrated (in the *Occam* distribution) for a famous article by Edward Witten posted electronically in November 1994. The resulting archival posting (Plain

[1]The alternative ".dvi" form is undeniably convenient, but also less flexible. For instance, the ".tex" version can be redimensioned and then retypeset to be read in comfort on any computer screen whereas a ".dvi" version (or anything derived from it) often has lines too long for the viewer. Other output formats like ".ps", being derived from the ".dvi", are similarly inflexible.

based) requires only 6 Ko of macros rather than the original 20 Ko. The 68 Ko body of Witten's article is unmodified.

Recently "harvmac.tex" has been enhanced by inputting "hyperbasics.tex", the hyper-reference macros of Tanmoy Bhattacharya, (and the new name is "lanlmac.tex").

The archival nature of the TEX postings that have been minimized using *Occam* still depends on Knuth's Plain format being archival. Plain TEX will probably forever remain unchanged, or at least be upwardly compatible in the best sense. If this does not seem a sure bet to you, your posting could be made archival on the scale of the life of TEX by subjecting the Plain macros to standard *Occam* discipline to produce "plain.occ". An article might require half the macros of "plain.occ" (or 20 Ko) to have its own format built from *initex*. This may seem needlessly radical to an English speaking user; however, for longer works in the many other languages that in any case require a special compilation from *initex*, I consider bootstrapping from *initex* the best approach to fully archival ".tex" postings.

There are many other macro packages that might benefit from *Occam*'s ".occ" structuring. The 'picture' macro package embedded in LATEX is an example; interestingly these macros run on Plain. The "amssym.tex" math symbol definition package for $\mathcal{AMS}$ fonts is another; it defines hundreds of control sequences, of which precious few are used in any given article. Both will be included in ".occ" form in the *Occam* distribution.

Would it be reasonable to convert $\mathcal{AMS}$-TEX into a Plain macro package with ".occ" structuring, much as it has been converted by the $\mathcal{AMS}$ into a documented LATEX package? This would be a move toward abandoning $\mathcal{AMS}$-TEX's status as a full-fledged format. In particular the $\mathcal{AMS}$-Plain package would have no influence outside of math mode.

In the long term, the structuring of a macro package for use with *Occam* will be the responsibility of the author of the package. Clearly such structuring will catch on only if *Occam* performs well as a *fully* portable TEX utility.

3 How Auditor operates

The first utility, *Auditor*, of the *Occam* package is already a perfectly portable TEX program "auditor.tex". On a small scale, *Auditor* is independently useful, so it should help you learn about *Occam* interactively.

Suppose you have a Plain TEX typescript "x.tex" that inputs a macro file "x.sty" whose size and history lead you to suspect it to be full of unused macros. *Auditor* serves to provide a list of macros defined in "x.sty" that are *not* used in "x.tex".

The basic idea used by *Auditor* is easy to grasp if described in a simplified form as follows. Auditor changes the definition of many a new (top level) macro `\mymacro` in "x.sty" so that its expansion includes an 'auditing' device able to report whether `\mymacro` has been been used in "x.tex". If the expansion of `\mymacro` is originally `blablabla` then during use of *Auditor* it becomes:

```
  \global\let\_mymacro_\@Used  blablabla
```

Here `\@Used` is a macro with some arbitrary expansion such as `@@Used`. With a bit of luck, a use of `\mymacro` with this new definition will cause an artificial macro whose name string is `_mymacro_` to become defined and have expansion `@@Used` — and do so without disturbing the normal functioning of `\mymacro`.[2]

The artificial macro `\_mymacro_` can clearly be polled after typesetting "x.tex"; that will record whether `\mymacro` has been used. Of course, *Auditor* must somehow find out which macros `\_mymacro_` should be polled. That is easy, and is done as follows, along with the above redefinition of `\mymacro`. Before the audit, `\def\mymacro` is replaced by `\Def\mymacro`; this `\Def` first stores `\_mymacro_` in a token sequence for later polling, and then makes the modification of the expansion of `\mymacro` we have exhibited above.

We have seen the simple idea behind "auditor.tex". Programmers will have also noticed that the idea can fail to work in unfortunate cases; we shall return to that.

3.1 How to make "audit.tex" list unused macros.

Here now is a user-oriented recipe to get a list of unused macros — they can often be quickly eliminated by hand. Recall that the typescript is "x.tex" and its macro file is "x.sty".

[2]Variants are possible. To identify macros that are little used, one could count how many times `\mymacro` is used.

— make a copy "x.occ" of "x.sty".

— temporarily have "x.tex" `\input` "x.occ" in place of 'x.sty".

— at the top of "x.occ" add: `\input auditor.tex`

— wherever an 'outer' (top level) definition `\def\mymacro...` occurs in "x.occ", replace `\def` by `\Def`. The latter is a special auditing version of `\def`, which is defined in auditor.tex.

— typeset "x.tex"; this will produce a file "audit.lst" containing a list of all the macros defined using `\Def`; in it the macros that are unused by "x.tex" are specially marked.

Using "audit.lst" to eliminate from "x.sty" the unused macros is sometimes tiresome to do by hand; the chief role of the *DefStrip* utility described in the next section §4 is to automate this.

Auditor is a bit more general than indicated so far: `\Def` has a number of cohorts that behave similarly:

```
\Def  (variant of \def)
\gDef  (variant of \gdef or \global\def)
\Let  (variant of \let)
\gLet  (variant of \global\let)
\Mathchardef  (variant of \mathchardef)
\Newsymbol  (variant of \newsymbol)
```

These replace corresponding uncapitalized control sequences in "x.occ". Here, `\newsymbol` (from "amssym.def") is a macro used copiously for declaration of symbols from the $\mathcal{AMS}$ math fonts msam∗ and msbm∗, as in "amssym.tex". Note that `\mathchardef` and `\newsymbol` define a 'mathchar' not a macro; but the capitalized versions define macros.

As has been mentioned, "auditor.tex" is not bullet-proof. Any change whatever in the expansion of a macro can *in principle* alter its behavior. For example, TEX can use `\ifx` and many other means to examine the expansion of a macro; a 'perverse' "x.tex" can always be constructed that stops compilation if there is any tampering with definitions.

However, if one exercises prudence this is *unlikely.* Here is some cautionary advice:

- Use `\Let` only with macros; for example `\Let\mymacro\thymacro` is allowable only when the control sequence `\thymacro` is a macro; the test command `\show\thymacro` will tell you if it really is one.
- Do not modify `\def`'s etc. within other definitions. This would often be pointless and perhaps dangerous. (But see section §5.)
- Avoid definitions involving "`\outer`" and "`\long`" macros; (But perhaps `\outerDef` and `\longDef` will be introduced to handle them.)

If there is trouble in using "x.occ", then opt (by dialog) to compose *without* an audit. There should be no change from the original behavior of "x.tex". Correct any misbehavior — often simply arising from a typing error in constructing "x.occ". Sometimes, here and there in "x.occ", one has to change `\Def` back to `\def` etc; (in that case the macro in question is clearly used!).

4 Occam structuring for macro packages, and the action of DefStrip

As described in the last section §3 above, the *Auditor* half of *Occam* uses both the macros "x.sty" and the typescript "x.tex" to establish a list "audit.lst" indicating macros unused in "x.tex". *Occam*'s second half, *DefStrip*, serves to delete them from "x.sty" in a quite automatic way. The file "x.tex" is not further used but the specially structured version "x.occ" of "x.sty" is required. In addition, more structuring must be added to "x.occ" than was necessary for *Auditor* — the fully structured macro file is said to be *Occam* structured.

The goal of this section is to specify the syntax of the more basic *Occam* structuring, and indicate how *DefStrip* should interpret it.

This structuring is less simple than the easy description of the action of *Auditor* in the last section §3 above might lead one to expect. It is true that *DefStrip* ultimately merely serves to delete selected lines of the file "x.sty". However, the result would be rather messy and less than minimal if only the lines occupied by the unused definitions were deleted. For example, one wants to delete comments attached to deleted macros, and possibly some auxiliary commands like `\newif...` not mentioned in "audit.lst".

We now make this more precise through describing the syntax by which these blocks are unambiguously specified, and the rules for block deletion.

4.1 Main specifications of the *Occam* syntax

Two composite symbols `%^` and `%_` are employed in conjunction with `\Def` etc. to delimit possible deletions; the percentage sign `%` makes them invisible to TEX. On its line, `%^` is always preceded by spaces only (zero or more); similarly `%_` is always followed by spaces only.

(A) Unconditionally deleted material

```
%%^_ <delete me>
```

The block of lines from `%%^_` to the end of file is then deleted. To delete just a segment use

```
%^ <delete me> %_
```

The block of ≥ 1 lines deleted must include no blank line. Note that it may well contain `\Def` etc. but not `%^`, `%_`.

The unconditional deletions will occur as if done in the order described, and before conditional deletions (described below) are considered.

(B) Conditionally deleted material

```
\Def \somemacro ...
                  ...%_
```

may cause deletion of the block of lines beginning with `\Def` etc. and ending with `%_`. This material is really deleted, precisely if the macro `\somemacro` is marked for deletion in the the file “audit.lst”.

The material `<maybe delete me>` must contain no blank line nor `%^`, `%_`, `\Def` etc; but it is otherwise arbitrary; in particular, macro arguments, comments, and auxiliary definitions are permissible.

Along with this material some additional preceding material is deleted, namely contiguous preceding lines (if any) that (a) are nonempty and (b) contain no `%_`. Typically, such preceding material might be comments or commands ‘owned’ by the macro being deleted. For example the whole block

```
%_
\ifx\undefined\eightpoint
  \Def\eightpoint{}
\fi  %_
```

will be deleted, precisely in case `\eightpoint` is marked as unused in "audit.lst". (The first `%_` could be replaced by a blank line.)

Note that `%_` is not really a closing delimiter since it can exist in arbitrary numbers without belonging to a matching pair. For another example, consider:

```
\Def\amacro ...%_
\newtoks\btoks %_
\Def\cmacro ...%_
```

Here, the the first two `%_` prevent `\newtoks\btoks` being deleted — in all circumstances.

The example

```
\Def\amacro ...
\Def\bmacro ...%_
```

is incorrect because the block beginning with `\Def\amacro ...` contains `\Def\bmacro`.

Sentinels

There is a second type of conditional deletion. Suppose `\amacro` is not used and is so designated in "audit.lst". It often occurs that several *disjoint* blocks of lines should be deleted along with the block containing the definition of `\amacro`. These blocks should each be designated as follows:

```
%/^\amacro
   <stuff>
%/_
```

`\amacro` is called the *sentinel (watchman)* for the block. The sentinel's line `%/^...` must contain nothing more than `%/^\amacro` and blank space. The initial and terminal lines will vanish along with `<stuff>`.

Summary of primary deletions by DefStrip

Any block `%^...%_` is unconditionally deleted, while a block signalled by `\Def`, `\gDef`, etc. with the help of `%_` and/or blank lines is deleted or not according as the macro following `\Def` etc. is or is not marked for deletion in "audit.lst". Similarly for blocks with sentinel macro. None of these blocks for conditional or unconditional deletion is allowed to contain an empty line nor any extraneous `%^`, `%_`, `%/^`, `%/_`, `%%^_`, `\Def`, `\gDef`, etc. The blocks introduced by `\Def`, `\gDef`, etc. include material extending backward as far as (but not including) a preceding line that is blank or terminated by one of `%_`, `%/_`. No such extension for blocks introduced by `%^`, `%/^` is allowed — nor would it be helpful.

Secondary cleanup by DefStrip

Beyond these primary deletions, the utility *DefStrip* performs a few auxiliary tasks:

- All remaining `\Def`, `\gDef`, etc. are converted to `\def`, `\global\def`, etc. Also, if a remaining `%_` is alone on its line (spaces ignored), the whole line disappears. And each remaining `%_` *not* alone on its line becomes `%` (this is the only deletion that can affect a line that survives, and TEX is unaffected).
- Any empty line sequence (usually created by the deletion of blocks of lines) is reduced to a single empty line.
- Residual appearances in "x.occ" of macros marked for deletion in audit.lst will be marked by `%%[VESTIGE]` (on a new following line).

Such an occurence of `%%[VESTIGE]` should be considered an error warning concerning the *Occam* structuring of "x.occ". Users may find such vestiges hard to deal with. Thus the programmer should enquire whether (for example) the vestige could be automatically deleted using the sentinel mechanism. For their part, users should report vestiges to the programmers along with the involved "audit.lst" file from "auditor.tex".

In most cases, anyone who programs TEX macros at an intermediate level will find it an easy task to provide *Occam* structuring for any simple macro file. However, as soon as the macro file has interdependent macros, and definitions of control sequences other than macros[3], attention from a programmer will be needed — plus testing.

The author of the macro package himself is the most appropriate person to introduce Occam structuring:

— The author has the oportunity to add extra documentation to the ".occ" version and make of it a fully documented master copy of the package. Using conditional deletions, one has flexible control of which documentation is passed on by *DefStrip*to the user.

— The author can often remodel the macro package to allow *Occam* to do a better job more simply.

— The author protects his work against the erosion of time; it becomes unimportant that large parts of a package become antiquated; only the parts that remain viable will be exported by *Occam* into users typescripts.

— Once committed to *Occam* (or similar means of macro distillation) the author can flout the usual rules of upward compatibility, provided it is made clear from the outset that the package is to be used exclusively to produce autonomous typescripts.

5 Nested macros

Unused macros whose definitions are nested within those of other macros that *are* used can often be be eliminated, although that would be very difficult on the basis of features of *Occam* described thus far.

We now describe suitable additional syntax. It was was implemented in 1995 through modifying both "auditor.tex" and *DefStrip*. This is probably more subject to change than features in earlier sections.

Where *DefStrip* is concerned, the new syntax is currently implemented quite trivially by making several passes, and the example below is conveniently explained in this way. However, the TEX version "defstrip.tex" will almost certainly reduce this to a single pass; the "audit.tex" utility already acts in a single pass.

Here is a generic 'example'. The original macro file contains:

[3]For example, unused fonts are hard to eliminate.

```
\def\MACRO{<stuff1>%
   <stuff2>
   \def\macro{<stuff3>}%
   <stuff4>%
   <stuff5>}
```

An *Occam* structured version is:

```
\Def\MACRO{<stuff1>%#_
  <stuff2>
  \DDef\macro{<stuff3>}%
  <stuff4>%#_
  <stuff5>}%_
```

The macro `\DDef`, defined in audit.tex, behaves much like `\Def` except that the associated sign distinguishing *unused* macros in “audit.lst” is `*#` in place of `*`.

Note that if `\MACRO` is unused then the whole block vanishes.

We are interested in the case where `\MACRO` is *used.* Then, on first pass of the macro file through *DefStrip*, `\DDef` is converted to `\Def`; and the marks `%#_` therein are converted to `%_`. At the same time, the file “audit.lst” undergoes one wave of changes `*#\` $\Rightarrow$ `#*\` and `*\` $\Rightarrow$ `#\`, i.e. asterisks move right or die on backslash. On the second pass through *DefStrip*, one is treating:

```
\def\MACRO{<stuff1>%_
  <stuff2>
  \Def\macro{<stuff3>}%
  <stuff4>%_
  <stuff5>}%
```

and, in response to an entry `#*\macro` in the current “audit.lst”, *DefStrip* will delete the block

```
<stuff2>
\Def\macro{<stuff3>}%
<stuff4>%_
```

i.e. this block is deleted precisely if `\macro` is *unused.* (Only a programmer can guess whether this elimination is safe!)

Nested Sentinels

The macros in such nested definitions are allowed to be *sentinels* for blocks, as follows. Often, one wants to delete other material along with the block surrounding `\macro`. The syntax for a block to be eliminated along with `\macro` is:

```
%#/^\macro
<stuff6>
%#/_
```

It is permissible to use `_` in place of `/_` on the above syntax. But not `^` in place of `/^` since that would give an unconditional deletion.

Nested unconditional deletions

There is also a notion of nested *un*conditional deletion. The syntax is:

```
%#^
<stuff6>
%#_
```

The developments of this section first proved desirable in "harvmac.occ", which is a 'worked example' in the *Occam* package.

Until the TEX program "defstrip.tex" is built, I would particularly welcome suggestions for improvements of *Occam* structuring.

6 Afterthoughts

6.1 Is Occam's Razor now dull?

Occam's Razor was one of the guiding principles of scientific thought for several hundred years before the coming of age of computers. However, I suspect the philosophy of Aristotle or Descartes is far more likely to appeal to computer scientists. One might go so far as to say that programmers have discarded Occam's Razor. Indeed, in object oriented programming they consciously cultivate the art of multiplication of entities, and even LATEX users do this sort of thing with commands such as `\newheading`. What can the minimalism of Occam's Razor offer TEX users at this late date? Probably just a few things.

(a) *Friendliness to human beings.* Unnecessary entities that cost a microprocessor only microseconds can cost the human mind a significant amount of time.

(b) *Extra storage space and computing power.* Both are in a period of exponential growth. But so is the TEX related software we use. Thus performance in a fixed task can occasionally decline. When this happens, the old-fashioned minimalism of Occam's Razor can prove essential to derive pleasure and profit from progress.

6.2 Plain versus LATEX

The goals of *Occam* do not make much sense in the LATEX world. The LATEX group is building official LATEX macro modules that cover more and more territory, and are universally available. If upward compatibility is fully maintained, there will be little reason to use *Occam* in the everyday LATEX world since all macros used will be standard. This is LATEX's greatest strength. It assures LATEX an important and possibly dominant role in electronic scientific publication based on ".tex" typescript format. The role *Occam* seeks to play in this sort of electronic publication lies mostly within the realm of Plain TEX and *initex*.

However, I have been alarmed by the growing difficulties facing an individual when programming LATEX beyond its current capabilities — I mean more than routine reparametering and renaming. It is not just LATEX's exponentially growing internal complexity that is alarming, nor the fact that LATEX runs slower and slower relative to Plain. The most debilitating problem is that internal macros should probably not be reprogrammed since (unlike the user macros) they are open to change. Thus the results of such individual programming efforts risk being electronically 'unpublishable' even if the macros involved are posted along with the article they produce — indeed the version of LATEX's internal macros on which they were based may vanish from the next LATEX update, and that may well invalidate the posting. A related fundamental difficulty is that TEX primitives are not guaranteed to be/remain accessible from within LATEX. The logical conclusion would seem to be that all programming of LATEX sufficiently deep to modify LATEX internals should be left to the official LATEX group.

Don Knuth seems to have had similar premonitions and takes a possibly more doubtful attitude. I repeat what he said on the occasion of the 10th anniversary celebration of TEX 82.

> *Suppose you were allowed to rewrite all the world's literature; should you try to put it all into the same format? I doubt it. I tend to think such unification is a dream that's not going to work.* [TUGboat, vol 13 (1992), page 424]

I (and Knuth?) may be unduly alarmed. *Occam* is nevertheless to some extent my attempt to stop the decline of Plain. (There is nothing significant I know how to do for this real or imaginary malady of LATEX.)[4]

LATEX continues to perform well for an expanding palette of routine tasks, and I am happily using it to prepare this article! The most favorable outcome would be for both approaches to work well. If this comes about, I expect the "look and feel" of Plain and LATEX to nevertheless steadily diverge.

In summary, the TEX utility *Occam* offers a novel response to the most debilitating problem of Plain TEX, namely the confusion and incoherence that come from continual and uncoordinated accretion of macros. Naturally, this weakness of Plain is clearly perceived by Leslie Lamport who stated (on the net in 1994):

> *Because Plain TEX is fixed, it seems likely that the Plain TEX community will fragment into numerous small islands in a sea of incompatibility.*

The LATEX programming group deals with the accretion of macros by constantly improving infrastructure while selectively enlarging LATEX, whereas my partial answer for Plain TEX is to use *Occam* to cull out inessential macros, restoring simplicity in the macro set actually used in each document, and thereby making new infrastructure and standardization largely unnecessary. *Occam*'s action has a parallel in classical programming, namely the use of a compiler — whereas the LATEX approach is parallel to the use of a big and constantly evolving interpreter.

My hope is that, with *Occam*'s help, Plain TEX will regain vigour and prove as viable as LATEX, and indeed complementary to it.

[4]An author who absolutely needs many of the basic talents of LATEX *and* new performance, might consider using a frozen but hopefully permanent format like LAMS-TEX or 'classic' LATEX 2.09 *and* macros distilled by *Occam* from personal of public packages whose permanence is in doubt.

References

[1] L. Siebenmann, *Occam*, a TEX utility, 1993-5; the electronic master posting in 1995 is on `ftp://matups.math.u-psud.fr://pub/TeX/Occam.dir`; see also the CTAN archives.

[2] L. Siebenmann, *Elementary text processing and parsing in TEX — the appreciation of tokens*, pp 62-73, TUGboat 13, April 1992.