bitelist.sty

“Splitting” a List at a List Inside
in TeX’s Mouth

Uwe Lück†

March 29, 2012

Abstract

bitelist.sty provides commands for “splitting” a token list at the first occurrence of a contained token list (i.e., for given $\sigma$, $\tau$, return $\beta$ and shortest $\alpha$ s.t. $\tau = \alpha\sigma\beta$). As opposed to other packages providing similar features, (i) the method uses TeX’s mechanism of reading delimited macro parameters; (ii) the splitting macros work by pure expansion, without assignments, provided the macro doing the search has been defined before processing (e.g., a file); (iii) instead of using one macro for a “substring” test and another one to replace the “substring”—which includes extracting corresponding prefix and suffix—, the same macro that detects the occurrence returns the split; (iv) \textit{e}-TeX is not required. (And \textit{I}\textit{H}e\TeX is not required.)

This improves the author’s \texttt{fifthdo.sty} (v0.51—and may once be used there). An elaborated approach (additionally to a simpler one) is provided that does not loose outer braces of prefix/suffix.

“Substring” detection and “string” replacement are (implicitly) included with respect to certain representations of characters by tokens. Counting occurrences and “global” replacement could be achieved by applying the operation to earlier results, etc.—so this approach seems to be “fundamental” for a certain larger set of list analysis tasks.

The documentation aims to prove the correctness of the methods with mathematical rigour.

\textbf{Related packages:}  \texttt{datatool, stringstrings, ted, texapi, xstring}

\textbf{Keywords:}  macro programming, text filtering, substrings

---

*This document describes version \texttt{v0.1} of bitelist.sty as of 2012/03/29.
†\url{http://contact-ednotes.sty.de.vu}
1 Task, Background Reasoning, and Usage

1.1 The Task Quite Precisely
Perhaps I should not have written “splitting” before, see Section 7 why I did so though. Actually:
At first we are dealing with token lists \( \tau \) and \( \sigma \) without braces (unless their category code has been changed appropriately) that can be stored as macros without parameter or in token list registers. We want to find out whether \( \tau \) contains \( \sigma \) (“as a subword”) in the sense that there are such token lists \( \alpha \) and \( \beta \) that \( \tau \) is composed as \( \alpha \sigma \beta \), i.e.,
\[
\tau = \alpha \sigma \beta
\]
and in this case we want to get \( \alpha \) and \( \beta \) of this kind with \( \alpha \) being the shortest possible. I.e., if there are such \( \gamma \) and \( \delta \) that \( \tau \) is composed as \( \gamma \sigma \delta \), \( \alpha \) must be
contained as a “prefix” in $\gamma$, i.e., $\gamma$ is composed as $\alpha \eta$ for some token list $\eta$. The token lists $\alpha, \beta, \gamma, \delta, \eta, \sigma, \text{and } \tau$ are allowed to be empty throughout.

The task will be extended for some braces in Section 4.

### 1.2 Idea of Solution

$\text{\TeX}$’s mechanism of expanding macros ($\text{\TeXbook}$ Chapter 20) at least has a built-in mechanism to return such $\alpha$ and $\beta$ provided $\tau$ contains $\sigma$. Define

\[
\texttt{\def\langle cmd\rangle#1\#2\theta\{\langle replace-def\rangle\}}
\]

where $\theta$ must be a token list (maybe of a single token) that won’t occur in $\tau$.\(^{\text{1}}\)

This is a limitation of the approach: It works for sets of such $\tau$ only that do not contain any of a small set of tokens or combinations of them. (\texttt{bitelist} will use $\texttt{\BiteSep}, \texttt{\BiteStop}, \texttt{\BiteCrit}$, or any other three that can be chosen.)

On the other hand, $\text{\TeX}$’s category codes ($\text{\TeXbook}$ Chapter 7) can ensure this quite well. E.g., we may assume that input “letters” always have category code 11 (or 12, or one of them), and for $\theta$ we can choose letters with different category codes such as 3. Without such tricks, you may often assume that nobody will input certain “silly” commands such as $\texttt{\BiteStop}$. (But it may become difficult when you use a package for replacement macros for generating its own documentation …)

With a $\langle cmd\rangle$ as defined above, $\text{\TeX}$ will

expand $\langle cmd\rangle\tau\theta$ to $\langle replace\rangle$,

where $\langle replace\rangle$ will be the result of replacing (a) all occurrences of #1 in $\langle replace-def\rangle$ by $\alpha$ as wanted and (b) all occurrences of #2 in $\langle replace-def\rangle$ by $\beta$ as wanted. I.e., $\langle cmd\rangle$ returns $\alpha$ as its first argument and $\beta$ as its second argument. The reason is that $\langle cmd\rangle$’s first parameter is delimited by $\sigma$ and the second one by $\theta$ in the sense of The $\text{\TeXbook}$ p. 203. Our requirement to get the shortest $\alpha$ for the composition of $\tau$ as $\alpha\sigma\beta$ is met because $\text{\TeX}$ indeed looks for the first occurrence of $\sigma$ at the right of $\langle cmd\rangle$.

### 1.3 When We Don’t Know …

When $\sigma$ does not occur in $\tau$ and we present $\tau\theta$ to $\langle cmd\rangle$ as before, $\text{\TeX}$ will throw an error saying “Use of $\langle cmd\rangle$ doesn’t match its definition.” When the purpose is “substring detection” only, without returning $\beta$, many packages have solved the problem by issuing something like

$\langle cmd\rangle\tau\sigma\theta$

---

\(^{\text{1}}\)I am still following others in confusing source code and tokens. I have better ideas, but must expand on them elsewhere. Writing $\texttt{\def}$ rather indicates that it is source code, then $\sigma$ etc. should be replaced by strings that are converted into tokens $\sigma$ etc. $\langle cmd\rangle$ sometimes is a \texttt{string} starting with an escape character, or it is an active character; but sometimes it rather is an “active” \texttt{token} converted from such an escape string or an active character.
Then (still provided $\theta$ does not occur in $\tau$) $\langle \text{cmd} \rangle$’s second argument is empty exactly if $\sigma$ occurs in $\tau$. This method has, e.g., been employed in \LaTeX’s internal \texttt{in@} mechanism (e.g., for dealing with package options) and by the \texttt{substr} package. \texttt{datatool} has used the latter’s substring test (for $\sigma$) before calling a macro for replacing ($\sigma$ by another token list, perhaps thinking of character tokens).

This way you get the wanted $\alpha$ as the first macro argument immediately indeed. An obstacle for getting $\beta$ is that $\langle \text{cmd} \rangle$’s second argument now contains an occurrence of $\sigma$ that is not an occurrence in $\tau$. In \texttt{fifinddo.sty} I didn’t have a better idea than using another macro to remove the “dummy text” from the second argument. I considered it an advantage as compared with \texttt{datatool} that one macro could do this for \textit{all} replacement jobs, while \texttt{datatool} uses \textit{two} macros with $\sigma$ as a delimiter for each $\sigma$ to be replaced.

But still, \texttt{fifinddo} has used \textit{two} macros for each replacement, the extra one being for presenting $\tau$ to $\langle \text{cmd} \rangle$, using a job identifier. This could be improved within \texttt{fifinddo}, but I could never afford to take the time for this.

### 1.4 The Trick

The solution presented here is not very ingenious, many students would have found it in an exercise for a math course. My personal approach was looking at \texttt{\GetFileInfo} from \LaTeX’s \texttt{doc} package. There they try to get \textit{two} occurrences of a space token this way:\footnote{We are undoubling the hash marks inside the definition text of \texttt{\GetFileInfo}.}

\begin{verbatim}
def\@tempb#1␣#2␣#3\relax#4\relax{
and \@tempb is called as
\@tempb\relax?␣\relax
or with $\tau = \langle \text{list} \rangle$
\@tempb(\texttt{list})\relax?␣\relax
The final \texttt{\relax} may not be removed, but for \texttt{doc} it doesn’t harm. It harms for me when I don’t want to have a \texttt{\relax} in a \texttt{.log} file list. \texttt{\empty} would be better, however ...

The idea is to use a \textit{three}-parameter macro for that \textit{single} occurrence of $\sigma$. We introduce a “dummy separator” $\zeta$ (or (\texttt{sep}), \texttt{\BiteSep}) between $\tau$ and the “dummy text” and a “criterion” $\rho$ (= (\texttt{crit}), \texttt{\BiteCrit}) for determining occurrence of $\sigma$ (= (\texttt{find})) in $\tau$ (= (\texttt{list})). Neither $\zeta$ nor $\rho$ must occur in $\tau$. We will have definitions about as
\begin{verbatim}
def\langle \text{cmd} \rangle #1\sigma#2\zeta#3\theta\{(\text{replace-def})\}
or
\begin{verbatim}
def\langle \text{cmd} \rangle #1\langle \text{find} \rangle #2\langle \text{sep} \rangle #3\text{\texttt{stop}}\{(\text{replace-def})\}
\end{verbatim}
\end{verbatim}
2 IMPLEMENTATION PART I

and τ will be presented with context

⟨cmd⟩τζρζθ or ⟨cmd⟩⟨list⟩⟨sep⟩⟨find⟩⟨crit⟩⟨sep⟩⟨stop⟩

This ensures that ⟨cmd⟩ finds its parameter delimiters σ, ζ, and θ, in this order. σ occurs in τ exactly if the second argument of ⟨cmd⟩ is ρ, and in this case the first occurrence of the second parameter delimiter ζ delimits τ. Then ⟨cmd⟩’s first argument is α, and the second one is β, as wanted. ⟨cmd⟩’s third parameter is delimited by the final θ (\BiteStop). When σ occurs in τ, ⟨cmd⟩’s third argument starts after the first of the two ζ, so it is σρζ. It is just ignored, this way ⟨cmd⟩ removes all the “dummy” material after τ. When σ does not occur in τ, we ignore all of its arguments, and the macro that invoked ⟨cmd⟩ must decide what to do next, e.g., keeping τ elsewhere for presenting it to another parsing macro resembling ⟨cmd⟩.

1.5 Installing and Calling

The file bitelist.sty is provided ready, installation only requires putting it somewhere where \TeX finds it (which may need updating the filename database).\footnote{http://www.tex.ac.uk/cgi-bin/texfaq2html?label=inst-wlcf}

Below the \documentclass line(s) and above \begin{document}, you load bitelist.sty (as usually) by

\usepackage{bitelist}

between the \documentclass line and \begin{document}; or by

\RequirePackage{bitelist}

within a package file, or above or without the \documentclass line. Moreover, the package should work without \LaTeX and may be loaded by

\input bitelist.sty

Actually, using the package for macro programming requires understanding of pp. 20ff. of The \TeX book. On the other hand, the package may be loaded (without the user noticing it) automatically by a different package that uses programming tools from the present package.

2 Implementation Part I

2.1 Package File Header (Legalize)

\begin{verbatim}
1 \def\filename{bitelist} \def\filedate{2012/03/29}
2 \def\fileversion{v0.1} \def\fileinfo{split lists in \TeX’s mouth (UL)}
3 \%% Copyright (C) 2012 Uwe Lueck,
4 \%% http://www.contact-ednotes.sty.de.vu
5 \%% -- author-maintained in the sense of LPPL below --
\end{verbatim}
2 IMPLEMENTATION PART I

2.2 Proceeding without L\TeX

Some tricks from Bernd Raichle's \german.sty—I need L\TeX's \ProvidesPackage for fileinfo, my package version tools. With \readprov.sty, it issues \endinput, close conditional before:

\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname ProvidesPackage\endcsname\relax \else
  \edef\fileinfo{\noexpand\ProvidesPackage{\filename} [%\filedate space \fileversion space \fileinfo]}
\expandafter\fileinfo
\fi
\chardef\atcode=\catcode'@
\catcode'@=11 % \makeatletter

Providing L\TeX's \@firstoftwo and \@secondoftwo:

\long\def\@firstoftwo #1#2{#1}
\long\def\@secondoftwo#1#2{#2}

2.3 Basic Parsing (No Braces)

\BiteMake{⟨def⟩}{⟨cmd⟩}{⟨find⟩} provides the parameter text (\TeXbook p. 203) for defining (by ⟨def⟩) a macro ⟨cmd⟩ that will search for ⟨find⟩:

\def\BiteMake#1#2#3{#1#2##1#3##2\BiteSep##3\BiteStop}

With \BiteFindByIn{⟨find⟩}{⟨cmd⟩}{⟨list⟩} you can use a ⟨cmd⟩ (perhaps defined by \BiteMake) in order to search ⟨find⟩ in ⟨list⟩. This is expandable as promised:

\def\BiteFindByIn#1#2#3{ %
  #2#3\BiteSep#1\BiteCrit\BiteSep\BiteStop}

Preparing a possible \edef as ⟨def⟩:

\let\BiteSep\relax \let\BiteStop\relax
And this is important in any case for correct testing of occurrence:\footnote{The idea for the “funny \texttt{Q}” is from the \texttt{fmtarg} package.}

\begin{verbatim}
\catcode'\Q=7 \let\BiteCrit=Q \catcode'\Q=11
\end{verbatim}

Perhaps you could increase safety of tests by using something similar to the funny \texttt{Q} for \texttt{\BiteSep} and \texttt{\BiteStop}. However, this would additionally require reimplementations of the macros for keeping braces (Section 4) using \texttt{\edef}.

### 2.4 Simple Conditionals

By \texttt{\BiteMakeIfOnly{⟨def⟩}{⟨cmd⟩}{⟨find⟩}}, you can make a command \langle\cmd\rangle that with

\begin{verbatim}
\BiteFindByIn{⟨find⟩}{⟨cmd⟩}{⟨list⟩}{⟨yes⟩}{⟨no⟩}
\end{verbatim}

chooses \langle yes\rangle if \langle find\rangle occurs in \langle list\rangle and \langle no\rangle otherwise.

\begin{verbatim}
def\BiteMakeIfOnly#1#2#3\BiteMake{#1}{#2}{#3}{\BiteIfCrit{##2}}
\end{verbatim}

\begin{verbatim}
\BiteIfCrit{⟨suffix⟩}{⟨yes⟩}{⟨no⟩}
\end{verbatim}
is the basic test for occurrence of \langle find\rangle in \langle list\rangle:

\begin{verbatim}
def\BiteIfCrit#1\ifx\BiteCrit#1\expandafter\@secondoftwo
\BiteIfCrit{⟨suffix⟩}{⟨yes⟩}{⟨no⟩}
\fi
\end{verbatim}

If \langle cmd\rangle’s second argument—same as \texttt{\BiteIfCrit}’s first argument—is empty, \texttt{\BiteCrit} is compared with \texttt{\expandafter}, so \langle yes\rangle is chosen. That is correct, it happens when \langle find\rangle is a suffix of \langle list\rangle.

\begin{verbatim}
\else \expandafter\@firstoftwo\fi
\end{verbatim}

### 2.5 Passing Results Completely—No Braces

So the previous \texttt{\BiteMakeIfOnly} generates pure tests on occurrence, giving away information about prefix and suffix. It may be considered a didactical step fostering understanding of the following. When, by contrast

\begin{verbatim}
\BiteMakeIf{⟨def⟩}{⟨cmd⟩}{⟨find⟩}
\end{verbatim}

has been issued, a later

\begin{verbatim}
\BiteFindByIn{⟨find⟩}{⟨cmd⟩}{⟨list⟩}{⟨list⟩}{⟨yes⟩}{⟨no⟩}
\end{verbatim}

will expand to

\begin{verbatim}
⟨yes⟩{(prefix){⟨find⟩}{⟨suffix⟩}
\end{verbatim}

if \langle list\rangle is composed as \langle prefix\rangle\langle find\rangle\langle suffix\rangle and \langle prefix\rangle is the shortest \(α\) such that there is some \(β\) with \langle list\rangle = \(α\langle find\rangleβ\). Otherwise, \(∗\) will expand to

\begin{verbatim}
⟨no⟩{⟨list⟩}
\end{verbatim}

This gives all the information available. For actual applications, it may be too much, and the macro programmer may do something in between of \texttt{\BiteMakeIfOnly} and \texttt{\BiteMakeIf}:
3 Example Applications

3.1 Splitting at Space

This work actually arose from modifying \GetFileInfo as provided by \LaTeX{}’s \pkg{doc} package so that it would deal reasonably with “incomplete” file info—for the \pkg{nicefilelist} package. \GetFileInfo works best when the file info contains at least two blank spaces. But how many are there indeed?—And I wanted to do it expandably: while \GetFileInfo issues definitions of \texttt{filedate}, \texttt{fileversion}, and \texttt{fileinfo}, date, version, and info should be passed as macro arguments.

\def\BiteIfSpace\% 
\BiteMake\def\BiteIfSpace\{ }\BiteIfCrit{#2}{#5#6{#1}{#2}}{#6#4{#4}}\%

The difference to the \texttt{\BiteMakeIf} construction is that we do not pass \texttt{(find)}, the space—it’s not essential. (\texttt{TODO} names may change ...)

Now

\BiteFindByIn\{ \BiteIfSpace \} \{ \{ \texttt{yes} \} \{ \texttt{no} \} \}

will pass prefix/suffix to \texttt{(yes)} or \texttt{(list)} to \texttt{(no)}. If this is needed frequently, here is a shorthand \texttt{\BiteGetNextWord}\{ \texttt{list} \} \{ \texttt{yes} \} \{ \texttt{no} \}:

\def\BiteGetNextWord\{ \BiteFindByIn\{ \} \BiteIfSpace \}

See a test in \texttt{bitedemo.tex} (Section 6).
3.2 Splitting at Comma

... left as an exercise to the reader ...

4 Keeping Braces: Reasoning

Now we want to generalize task (Section 1.1) and solution (Section 1.4) for the case that $\tau = \langle \text{list} \rangle$ has (balanced) braces (with category codes for argument delimiters), while $\sigma = \langle \text{find} \rangle$ still has not (does not work with our method). So with $\tau = \alpha\sigma\beta$, $\alpha$ ("prefix") or $\beta$ ("suffix") or both may contain braces. But we consider another restriction: braces must be balanced in $\alpha$ and in $\beta$, we don’t try parsing inside braces (as opposed to the search for asterisks in Appendix D of The \TeXbook).

According to \TeXbook p. 204, when a macro $\langle \text{cmd} \rangle$ finds an argument formed as $\{ \langle \text{tokens} \rangle \}$, in $\langle \text{cmd} \rangle$’s replacement text only $\langle \text{tokens} \rangle$ is used, i.e., outer braces are removed. So when $\alpha = \{ \langle \text{tokens} \rangle \}$, a parser $\langle \text{cmd} \rangle$ as defined by our methods above will return $\langle \text{tokens} \rangle$ instead of $\{ \langle \text{tokens} \rangle \}$—likewise for $\beta$. We are now trying to keep outer braces in prefix/suffix by a more elaborate method.

The idea is to present $\tau = \langle \text{list} \rangle$ with context\footnote{Perhaps I am confusing $\empty$ and the token list containing just $\empty$ here?}

\[
\langle \text{cmd} \rangle\empty\langle \text{list} \rangle\langle \text{stop} \rangle\langle \text{sep} \rangle\langle \text{find} \rangle\langle \text{crit} \rangle\langle \text{sep} \rangle\langle \text{stop} \rangle
\]

or in the notation of Section 1.4

\[
\langle \text{cmd} \rangle\empty\tau\zeta\sigma\rho\zeta\theta
\]

Then, if $\langle \text{find} \rangle$ occurs in $\langle \text{list} \rangle$, we must remove the $\empty$ from the prefix that we get with the earlier method (easy) and $\langle \text{stop} \rangle$ from the suffix (tricky, similar problem recurs). Using old $\theta$ for a new purpose works here because $\langle \text{cmd} \rangle$ will look for $\theta$ only when it has found $\zeta$ before.

Mere testing for occurrence is not affected.

\\texttt{\textbackslash BiteMakeIfOnly} and \texttt{\textbackslash BiteFindByIn}

still can be used. We provide an improved version of

\\texttt{\textbackslash BiteMakeIf (\texttt{\textbackslash BiteMakeIfBraces})}

and of

\\texttt{\textbackslash BiteFindInIn (\texttt{\textbackslash BiteFindInBraces}).}
5 Implementation Part II

5.1 Keeping Braces

\BiteFindByInBraces{(\textit{find})}{(\textit{cmd})}{(\textit{list})}{(\textit{yes})}{(\textit{no})}

varies \BiteFindByInIn according to the previous:

\begin{verbatim}
\def\BiteFindByInBraces#1#2#3{\% #2\empty#3\BiteStop\BiteSep\BiteCrit\BiteSep\BiteStop{#3}}
\end{verbatim}

Such a \textit{cmd} can be made by \BiteMakeIfBraces{(\textit{def})}{(\textit{cmd})}{(\textit{find})}:

\begin{verbatim}
\def\BiteMakeIfBraces#1#2#3{\% \BiteMake{#1}{#2}{#3}##4##5##6{\% \BiteIfCrit{##2}{\% \textit{no} works as before. For \textit{yes}, first the \empty in the prefix is expanded for vanishing. \BiteTidyI and \BiteTidyII continue tidying. \% \BiteStop in suffix by \BiteTideII removes outer braces:
\% if #3 in ##4 \BiteTidyI{#1}\% \BiteTidyII\empty##2}\% suffix
\BiteTidyII##3}\% find
\BiteTidyII##5}\% yes
\BiteTidyII##6}\% otherwise
}
\end{verbatim}

Another \empty avoids that removal of \BiteStop in suffix by \BiteTideII in \textit{yes}.
removes outer braces:

\begin{verbatim}
\def\BiteTidyI#1#2{\% \BiteTidyII{##1}}
\def\BiteTidyII#1\BiteStop{#1}
\def\BiteTidied#1#2#3#4{#4{#2}{#3}{#1}}
\end{verbatim}

5.2 Leaving the Package File

\begin{verbatim}
\catcode\@=\atcode
\endinput
\end{verbatim}
5.3 VERSION HISTORY

v0.1 2012/03/26 started
2012/03/27 continued, restructured
2012/03/28 continued, separate sections for "Mere Occurrence" vs. ...; keeping braces, \BiteIfCrit
2012/03/29 proceeding without \LaTeX corrected, restructured

6 Examples/Tests

You should find a separate file bitedemo.tex with examples. It may be run separately with \text (Plain \LaTeX)—demonstrating that bitelist is "\texttt{generic}", then finish by entering \texttt{\bye}. With \texttt{\LaTeX\_bitedemo.tex}, end the job by entering \texttt{\stop}. \texttt{Expandability} is demonstrated by the \texttt{\BiteFind} commands running with \texttt{\typeout}.

\begin{verbatim}
def\filename{bitedemo.tex} \def\filedate{2012/03/29} \def\fileinfo{demonstrating/testing bitelist.sty (UL)} \expandafter\ifx\csname ProvidesPackage\endcsname\relax \else \edef\bitedemolatexstart{\noexpand\ProvidesFile{\filename}\[\filedate\space\fileinfo\]\noexpand\RequirePackage{bitelist}} \expandafter\bitedemolatexstart \fi \ifx\BiteMakeIf\undefined \input bitelist.sty \fi \ifx\typeout\undefined \def\typeout{\immediate\write17} \newlinechar'^^J \fi \def\splitted #1#2#3{>>#1|#2|#3<<} \def\unsplitted#1{>>#1<<} \def\spacetocomma#1#2{>>#1,#2<<} \BiteMakeIfOnly \{\def}{\occursyesno}{no} \BiteMakeIf \{\def}{\noshowsplit}{no} \BiteMakeIfBraces \{\def}{\noShowSplit}{no} \typeout{'^J \BiteFindByIn \{\def}{\occursyesno} \{bonobo\}{YES!}{NO!} \BiteFindByIn \{\def}{\noshowsplit} \{bonobo\}{bonobo}\{\splitted}\{\unsplitted} \BiteFindByInIn\{\def}{\noShowSplit}\{no} \{bonobo\}{\splitted}\{\unsplitted\} \typeout{'^J \BiteFindByIn \{no\}{\occursyesno} \end{verbatim}
7 The Package’s Name

This package deals with TeX’s expansion mechanism. In Knuth’s metaphor, this is TeX’s mouth. I am not entirely sure, I have never understood it, or I have understood it only for a few days or hours. However, the package deals with “Lists in TeX’s Mouth” as described in Alan Jeffrey’s 1990 TUGboat paper (Volume 11, No. 2, pp. 237–245)\[6\]

“Splitting” in title and abstract is an attempt to describe the package briefly without speaking Mathematalese. It roughly refers to certain string functions in various programming languages\[7\] with “split” in their name. However, there strings are split at separators such as commas. I am thinking here that a comma is a certain string “,” and this can be generalized to “splitting” at any substring. With TeX, the analogues are (a) the token with the character code of the comma and category code 12, or the token list consisting of this single token,—and (b) other lists of tokens …

Anyway, calling a triple \((\alpha,\sigma,\beta)\) of token lists such that \(\tau = \alpha\sigma\beta\) a “split” of \(\tau\) is not necessarily a bad idea. Moreover, the blank space example (Section 3.1) is very close to the original idea of splitting at separators, a blank space is about as common as a separator as the comma is.

Finally, according to en.wiktionary.org, the Proto-Indo-European origin of “to bite” just means “to split.”\[8\] So in TeX’s mouth, splitting and biting is the same.

---

\[6\] tug.org/TUGboat/tb11-2/tb28jeffrey.pdf
\[7\] en.wikipedia.org/wiki/String_functions#split
\[8\] en.wiktionary.org/wiki/bite#Etymology