The unisugar Package

Yossi Gil*
Department of Computer Science
The Technion—Israel Institute of Technology
Technion City, Haifa 32000, Israel

February 21, 2011

Abstract

This package provides syntactic sugar for LaTeX commands, using selected Unicode characters: Certain Unicode characters can be used as shorthand for certain LaTeX commands. The package also makes it possible to use the familiar command key symbol, ⌘ as a prefix of TeX's macros (the backlash character, \, can still be used). And it allows the use of visual space, ␣, within the names of macros, thus making it easier to write macros whose names are composed of more than one word.

In this document I describe these syntactical extensions, and explain why they make it easier to compose right-to-left documents.

Contents

1 Introduction

You do not really *need* this package. As its name implies, all it offers is what people call "syntactic sugar"—the ability to use a selected number of Unicode characters instead of and within the most common LaTeX commands.

*mailto:yogi@cs.Technion.ac.IL

Some may appreciate the fact that using this package, the text you type—the input to LaTeX—will look a bit more neat and infinitesimally closer to the output. Others will argue against it, mentioning that the input document will be less portable, and less "LaTeX-like" (whatever this term means).

Still, using this package, you will find yourself typing a bit less, provided you can configure your text editor or keyboard driver to generate the handful of Unicode characters defined by this package.

More importantly, if your input files include right-to-left text, you are likely to find this package indispensable. If your document is indeed going to include right-to-left text, please pay special attention to Section 2.2, which describes how `unisugar` should help you with your document divisioning directives, and to Section 2.5, which explains how `unisugar` makes it easier to intermix LaTeXcommands with your text. If however you are not likely to include right-to-left text in your documents, you do not need to read these sections.

2 User Guide

Simply apply a `\usepackage` directive to use this package.

```
\usepackage{unisugar}
```

There are no package options at this time.

You would then have to use a Unicode based version of LaTeX, that is XeLaTeX or LuaLaTeX to process your input file, e.g., for processing the document you are now reading, I typed at my shell command prompt

```
xelatex unisugar.tex
```

2.1 Document Divisioning Commands

Two of Unicode's typographical characters, ¶ (code point B6, the pilcrow sign) and § (code point A7, the section sign) are employed in `unisugar` to make shorthands for LaTeX traditional document divisionining commands: `\section`, `\subsection`, `\subsubsection`, and the lesser units, `\paragraph`, and `\subparagraph` commands.

Thus, instead of writing

```
\section{User Guide}
```

at the beginning of this section, I typed just

```
§ User Guide
```

Similarly, instead of

```
\subsection{Document Divisioning Commands}
```

I typed

```
§§ Document Divisioning Commands
```

to generate the header of this subsection.

Observe that I did not need to type nor an opening curly bracket, `{`, neither a closing curly bracket, `}`. The division's title extends from the `§` character (in case of a section), or the `§§` characters pair (in case of a subsection) until the end of the line.

Table 1 summarizes the shorthands for the document divisioning commands. The original versions are always there, in case you need to use the starred version of the divisioning command, or pass an optional argument to it.

Division	LaTeX	`unisugar`
part	`\part`	—
chapter	`\chapter`	—
section	`\section`	`§`
sub-section	`\section`	`§§`
sub-sub-section	`\subsection`	`§§§`
paragraph	`\paragraph`	`¶`
sub-paragraph	`\subparagraph`	`¶¶`

Table 1: Shorthands for divisioning commands

2.2 Right-to-left Text Editing

There is a great advantage of using these sugared replacements in composing left-to-right documents. Think of a a Hebrew document including a section named מבוא (Which means "Introduction" in Hebrew). Then, with plain XeLaTeX, the document would start with a directive

```
\section{מבוא}
```

Even if your text editor can manage well mixed directionality text, you will find editing the above line a bit confusing. The reason is that the character following the opening curly brackets is not א but rather מ.

As the cursor moves forward beginning at the first character in the line, it hits the opening curly brackets, and then you may expect it to proceed to the adjacent letter א. This so called *visual* flow is in fact incorrect. The more correct *logical*

flow prescribes that the cursor should instead "jump" to the to the letter מ, which is the first letter of the word מבוא.

Some editors adhere to the visual flow, others, more modern editors, to the logical flow. But experience shows that both are quite confusing.

Our sugared version of the divisioning directives offer a more sane alternative. You can write instead

```
§ מבוא
```

To understand why the latter is so much better than the former, you need to know a bit about the manner in which UNICODE deals with text directionality. Broadly speaking, characters come in three major varieties:

1. Left-to-right directed characters, including e.g., Latin characters.
2. Right-to-left directed characters, including e.g., characters of the Hebrew alphabet.
3. Undirected characters, including the digits 0-9, punctuation characters, and characters such as §, ¶, ␣, and ⌘ which are not part of specific writing script.

UNICODE assigns a direction to each line according to the first strongly directed character of that line, and then proceeds to assigning directionality to subsequences of characters occurring in that line.[1]

Most text editors follow UNICODE's algorithm. The line

```
\section{מבוא}
```

will thus be classified as left-to-right, with the מבוא portion classified right-to-left, leading to the visual vs. logical confusion and to irritating cursor jumps.

Further, the fact that the entire line is left-to-right will lead text editors such as `gedit` to place the first character of that line at the left most position in the window. This might be confusing even further since the section body will, most likely, be classified as right-to-left, and hence will be laid out on the screen with the first character at the right-most position.

Figure 1 depicting the use of `gedit` to compose a Hebrew LaTeX document may help in visualizing the difficulty. Line 18 containing the section title is laid out left-to-right, despite the fact that the title is written Hebrew, in visual discrepancy with lines 19 and on, containing the section body, which are laid out right-to-left.

In contrast, the sugared version of our document divisioning command

[1] The full algorithm is fairly complicated, taking into account "weak" directionality of some of the undirected characters, explicit directionality markers and nested directionality levels; details can be found here `http://unicode.org/reports/tr9`.

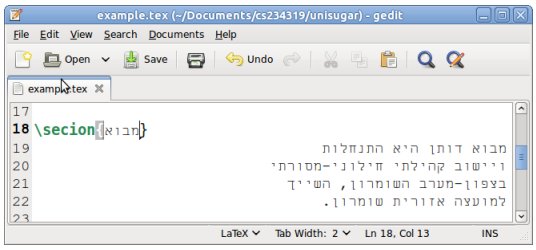

Figure 1: Using traditional sectioning directive with right-to-left text

```
§ מבוא
```

has no left-to-right characters, and hence will be classified as being entirely right-to-left: The cursor will not jump as it moves across that line.

Compare Figure 1 with Figure 2 in which the sugared version of the `\section` directive is used. We see that in Figure 2 *all* lines are laid out right-to-left.

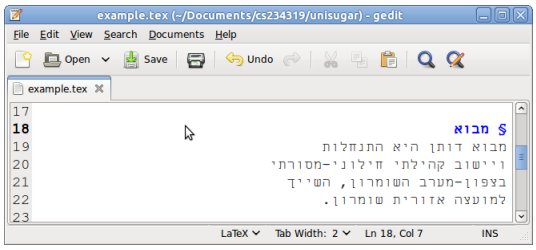

Figure 2: Using sugared sectioning directive with right-to-left text

2.3 Other Useful Unicode Characters

The above sugar replacement for divisioning commands does not scale well. With the exception of mathematical symbols, it is difficult to find reasonable substitutes for UNICODE replacements for the majority of LaTeX commands.

Still, four more UNICODE characters are used by `unisugar` as aliases for common LaTeX commands:

1. Code point 2022, the bullet, rendered as •, is yet another name for the `\item` command. In fact, to obtain this item, I typed

```
• Code point 2022, the bullet, rendered as \•,
is yet another name for the \verb+\item+ command.
In fact, to obtain this item, I typed:
```

2. Code point 23CE, the return symbol, rendered as ⏎ is an alias for \\. The \author directive of this manuscript was typed out as

```
\author{
  Yossi Gil
    {\url{mailto:yogi@cs.technion.ac.il}}⏎
   Department of Computer Science⏎
   The Technion---Israel Institute
                 of Technology⏎
   Technion City, Haifa 32000, Israel
}
```

3. Code point 2316 (position indicator), rendered as ⌖ is an alias for LaTeX's \label command. To generate a label for Table 1, I wrote:

```
⌖{Table:divisions}
```

4. Code point 261D (white up pointing index) rendered as ☝ is an alias LaTeX's \ref command. To reference Table 1, I wrote

```
To reference Table ☝{Table:divisions}, I wrote
```

2.4 Extended Syntax for Commands

It is futile to try to introduce an pictorial, easy to remember, symbol for each of the LaTeX commands in ordinary use, or even for a substantial portion of these. As large as it is, the Unicode character repertoire simply does not include icons that associated visually with notions such as \verb+, \begin{description}, etc. And even if it was, such a large set would be difficult to memorize. Worse, methods for producing so many characters would be cumbersome.

Thus, you would have to type LaTeX commands every so often. This package offers a slightly better syntax for writing these.

First, Unicode's code point 2318, rendered as ⌘, is used in many computing systems to denote the command key. With unisugar,the ⌘ character can be used as a control sequence prefix, So, instead of writing at the beginning of this document

```
\begin{document}
\maketitle
\begin{abstract}
This package provides syntactic sugar…
```

I wrote

```
⌘ begin{document}
⌘ maketitle
⌘ begin{abstract}
This package provides syntactic sugar…
```

Second, `unisugar`, extends to the usual set of 52 letters (a–z and A–Z) UNICODE character 2423, the open box, which looks like like visible space in its rendering ␣. This character can thus participate in control sequences. The intention is that it will serve for separating words in the case that your control sequence is composed of several words.

The names of large number of LaTeX commands are made from two words There are even a dozen or so control sequences whose name consists of three words, e.g., `\enlargethispage` and `\addcontentsline`. Although `unisugar` does not provide aliases for any of these multi-word commands, you can do so yourself. For example, at the preamble of this document, right after `\usepackage{unisugar}`, I wrote

```
⌘let⌘use␣package=⌘usepackage
⌘use␣package{xspace}
⌘let⌘new␣command=⌘newcommand
⌘new␣command⌘unisugar{⌘texttt{unisugar}⌘xspace}
```

2.5 Intermixing Commands with Right-to-Left Text

You may not appreciate so much the advantage of typing UNICODE's ⌘ instead of plain ASCII's \. Granted, on most keyboards, typing \ would be easier.

However, the nice property of ⌘ is that it directionally neutral. You would have to think about a sentence involving at least one LaTeX control sequence and/or a slash character to understand what I mean. You would not have to think so hard, since, this last sentence, namely,

> "*You would have to think about a sentence involving at least one LaTeX control sequence and/or a slash character to understand what I mean.*"

is a perfect example, since it is a sentence which involves a LaTeX control sequence, since the LaTeX logo is printed out using the "`\LaTeX\`" control sequence. Further, this sentence includes the slash character.

Typing this sentence in English is fairly straightforward.

```
You would have to think about a sentence involving at least one
\LaTeX\ control sequence and/or a slash character to understand
what I mean.
```

Let me translate this sentence into Hebrew for you.

יהיה עליך לחשוב על משפט המכיל לפחות פקודת בקרה אחת של LaTeX ו/או לוכסן בכדי להבין למה אני מתכוון.

What input does LaTeX require in order to produce the above? Using backslashes and traditional LaTeX notation I would have written

יהיה עליך לחשוב על משפט המכיל לפחות פקודת בקרה אחת
של \LaTeX\ ו/או לוכסן בכדי להבין למה אני מתכוון.

Figure 3 shows what this might look on an actual text editor. This may not seem too complicated, but a closer look would reveal that the production and inspection of this LaTeX is hindered by two or three annoying hidden issues.

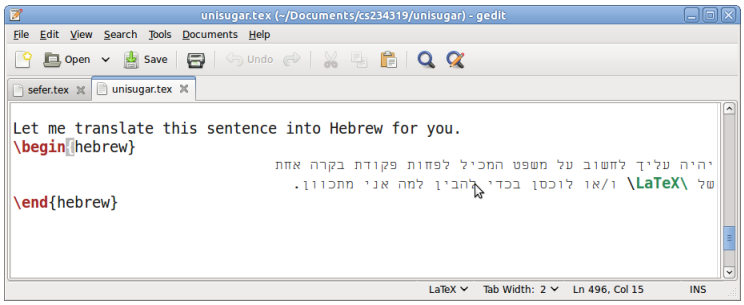

Figure 3: A Hebrew sentence containing a control sequence (without `unisugar`).

First, the two backslash characters in the figure are not really *back*slashes. The reason is that Hebrew is written right-to-left, and the \ character leans in the text direction, and should therefore be considered a *forward* slash in this context.

The second annoyance is the distinction between the two occurrences of this character: the first is at the beginning of the control sequence `\LaTeX`, denoting that the control sequence starts at that point. The second occurrence is at the end of the same control sequence, with the purpose of escaping the space that follows. This escape is required to prevent the control sequence from consuming the spaces that follow.

The distinction between the first and which is the last "backslash" is clear in the case that the enclosing sentence is written left-to-right. But, both humans and text processing devices may be confused when the enclosing sentence is right-to-left.

(There is yet a third difficulty in the above sentence which I will not address here. The English forward slash character is used in Hebrew for separating the day, month and year components of a date. This is natural, since dates involve digits, and these are written, even in Hebrew, left to right. Most Hebrew authors extrapolate this convention to the separation of Hebrew words by a "forward" (left-leaning) slash as in the phrase ו/או in the above. Other authors would right this phrase with the slash leaning in the text direction, i.e., ו\או)

The fact that the ⌘ character does not lean neither left nor right, takes care of the first annoyance.

The remedy for the second is simpler—use a pair of curly brackets to mark the end of the control sequence.

Thus, with `unisugar`, I would write:

```
יהיה עליך לחשוב על משפט המכיל לפחות פקודת בקרה אחת
.של ⌘LaTeX{}ו/או לוכסן בכדי להבין למה אני מתכוון
```

Figure 3 shows this sugared input as it appears in the gedit editor. We can see that the entire control sequence phrase is written left-to-right, with the escape character first, and the pair of curly brackets last.

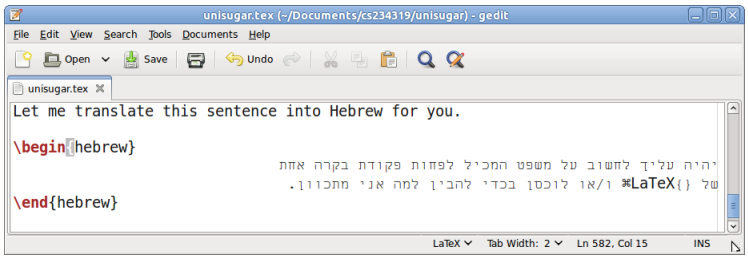

Figure 4: A Hebrew sentence containing a control sequence (with `unisugar`).

Evidently, mixed directionality text is slightly easier and clearer. But we can do even better, if we allow Hebrew characters in control sequences. This is carried out by (a yet to be published) another package, named `sukkar`, which, relying on `unisugar` does precisely this and more. Package `sukkar` also translates many common LaTeX commands to Hebrew, and since juxtaposition of words looks weird in Hebrew, it uses the ␣ Unicode character to separate words. With `sukkar` one can write, e.g., `⌘ תרתכו␣השע` instead of `\make_title`.

3 Acknowledgements

Will Robertson advised gave the advise of using XeLaTeX and `polyglossia` to circumvent a bug of `utf8x`. I pay tribute to Bruno Le Floch and Martin Scharrer who together devised the mechanism that made it possible to define a command which takes the rest of the line as argument. Martin Scharrer and Will Robertson encouraged me to work on this package. Vafa Khalighi devotion to bidirectional text processing with LaTeX was truly inspirational.