TSET Language Specification

Bruce McCulloch Dr. Richard Beigel

CONTENTS

1 Running TSET

The authors recommend that you build and install TSET on your own system using GNU make. Simply run make in the directory where you have cloned the git repository and TSET should be built very quickly. TSET does not require static or dynamic linking with any other nonstandard libraries. In order to install TSET to your system, move the ELF file to your binary directory or run sudo make install.
Running TSET itself is fairly straightforward, just use the following command structure in your terminal:

```
$ tset <path-to-t-file>.t -o <path-to-outfile>.tex
```

If an outfile is not provided, the resulting LaTeX will simply be written to stdout.

2 Macros

TSET's macro system involves the usage of four commands:

1. **%def** - for macro replacements outside of math mode
2. **%mathdef** - for macro replacements inside of math mode
3. **%mathword** - for typesetting a single word in roman inside math mode
4. **%mathwordlist** - for typesetting multiple words in roman inside math mode A macro consists of one of the four commands, followed by a macro (what TSET will search for), and for a def and mathdef a macro body is included as well (what TSET will replace the macro with). Both macros and macro bodies can be any length of tokens, but they can not include a space unless they are contained by a set of grouping characters (quotes, brackets, etc). It should be noted that macros are case-sensitive.

2.1 Plaintext Macros

A plaintext macro is any macro that takes a defined set of tokens and replaces it with another defined set of tokens. Using a plaintext macro is the same thing as running a find-and-replace command over your document. For example, here is a macro replacement that replaces every instance of the word "red" with the word "blue":

```
%def red blue
```

This would take the following sentence:

```
I drive a red car.
```

and make it into the following sentence:

```
I drive a blue car.
```

2.2 Wildcard Macros

Wildcard macros are a bit different. Instead of replacing a set of tokens with another set of tokens whole-cloth, they can match one or more of the tokens from the text to be replaced, and move it into the new replacement series of tokens. There are three types of wildcard tokens one can place in the macro portion of a definition:

1. **#w** - meaning a word or a single token
2. **#s** - meaning a string or an unbounded number of tokens
3. **#n** - meaning a number, either a digit or a floating point number

Take for example the following definition which replaces any two numbers seperated by a forward slash with the LaTeX fraction command using those two numbers as arguments:

```
%def #n/#n $\frac{#1}{#2}$
```

This would take the following sentence

```
$0.75$ can also be written as 3/4
```

and make it into the following sentence:

```
$0.75$ can also be written as $\frac{3}{4}$
```

Notice also that the ordering of the wildcard tokens in the macro body can be moved in any combination. All of the following defs are valid:

```
%def #n/#n $\frac{#2}{#1}$
%def #n/#n $\frac{#1}{#1}$
%def #n/#n {#1, #1, #1, #2, #1, #2 ...}
```

It should also be noted that a macro of $\#n/\#n$ will match $1/2$, but it will not match A/B, as those would be considered words and must be matched with a macro in the form of $\#w/\#w$.

2.3 mathword and mathwordlist

The mathword and mathword list commands do not require a macro body, and the only replacement they are capable of making is replacing a math mode word token with it's roman typeset equivalant. For example, the following command:

```
%mathdef x
```

Would make the following sentence:

```
{\forall x \in Y | x = 2}
```

into the following sentence:

```
{\forall {\rm x} \in Y | {\rm x} = 2}
```

Of course, this would only apply in math mode. Appearances of the defined mathword outside of math mode would not be replaced unless another $\%def$ were created.
%mathwordlist operates in much the same way, but it allows a user to define more than one mathword in a single command, like in the following example in which we define mathwords foo, bar, and qux.

```
%mathwordlist {foo bar qux}
```

3 Tables

Tables in TSET at the moment do not have any formatting arguments available, and only take files in the CSV format. Be smart about file management, you should include the absolute path for the CSV file or run TSET in the directory containing the file. Do not expect TSET to be able to find the CSV file without a proper directory structure. Creating a table is very simple:

```
%table <path-to-table>.csv
```