# Project Report – Generating Layouts for Re-entry Forms

Thomas Schmidt

January 12, 2020

## 1 Introduction

The Laws of Form[3] describe an interesting logical system that can be used to express a wide variety of mathematical, philosophical, or economic concepts, problems or systems. It is based upon the notion of distinction, and uses a symbol called the "cross" $\overline{\phantom{x}|}$. Up until now, there was no easy way of displaying such forms in a scientific writing environment like LaTeX, especially given more complex forms as, for example the following:

$$f = \overline{\overline{\overline{a}|\,\overline{b}}\,c}$$

In this project we implemented a LaTeX package that facilitates several commands to easily encode and write such formulas. We will first introduce the basic encoding principle by which such forms are required to be expressed. Then explain the concrete implementation and corresponding design decisions. Further, we will go over exact usage of this package, how to properly use the encoding and drawing commands, as well as how to expand this package with further features. After which we will give an outline of which features are presently not implemented, but might be desirable to implemented in the future.

## 2 Encoding

The main question that posed itself was how to encode forms in a command to then be drawn. When looking at simple forms without re-entries you can view the crosses just as parentheses, i.e.

$$\overline{\overline{fh}|\,\overline{gh}\|} \equiv ((fh)(gh)).$$

We are able to encode such forms with nested commands, e.g. `\lof{<content>}`, with the start of the command being the opening, and the end being the closing bracket. These commands by themselves do not need an identifier to determine which "parenthesis" belongs to which, as LaTeX itself ensures a proper nesting of commands and returns an error if there are syntax error or ambiguity. However, if we want to use re-entries, which can originate from any cross, and can be placed anywhere in the form, we need an identifier for each cross, to relate the re-entries to. This identifier will have to be assigned manually as an additional parameter with each `\lof`-command as well as the re-entry-command `\re`.

The form can then be represented in a linear way, for example,

$$f = \overline{\lfloor a \rfloor \lfloor b \rfloor c}$$

can be represented as

```
(3 (2 (1 reentry2 a )1 reentry3 b )2 c )3
```

The numbers behind the brackets and the word "reentry" representing their corresponding indices. Note, that the first bracket starts with an index of 3, as we have a nested form here, with the outermost layer belonging to the $c$ variable.

This means that such a form should be encoded, in principle, by the following simple example,

```
lof{ lof{ lof{ re{2} a }{1} b }{2} c {3}
```

However, in the implementation process, it was found necessary to extend this simple representation to possess a separate parameter for variable or description text, in this case `a`, `b`, and `c`, as well as differentiate the new command `\lofc`, which does have this accommodation for text, from the original command `\lof`, which does not have this parameter. Hence, our example form of $f = \overline{\lfloor a \rfloor \lfloor b \rfloor c}$ would be encoded as follows:

```
f = \lofc{
      \lofc{
        \lofc{\re{2}}{a}{1}
        \re{3}}{b}{2}
      }{c}{3}
```

# 3 Implementation

This concept was implemented using the `amsmath`[5] package for mathematical utilities, the `ifthen`[1] package for conditional commands, the `xstring`[6] package for string manipulation, the `arrayjobx`[2] package for data structures, as well as the `tikz`[4] package for drawing the graphics.

We will first go into implementation problems we encountered before describing the detailed internal work-flow of the package.

## 3.1 Nested Usage of Commands

As mentioned above, we are dealing with nested commands, which LaTeX natively supports. However, there is no facility with which we can differentiate LaTeX-commands from ordinary text, i.e. variables or descriptions. This means, for example, in a command like `\lof{ \lof{b} a}`, even with indices, there is no simple way to know into which bracket, the "`a`" and "`b`" belong respectively. Therefore, a second command `\lofc`, with an additional parameter specifically for text has been introduced. This means, the command now has the following structure: `\lofc{<content>}{<text>}{<index>}`. With the `<content>` parameter being used for further, nested commands. As a result of this, as of right now, the nesting of commands can only take place to the left side of the text. Meaning, nestings like $(a(b))$, are currently not possible, but can be implemented in the future.

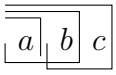
## 3.2 Formatting and the Storing of the Form Structure

The first few implementations used the `\lof` command to directly create `tikz` nodes at the corresponding position in the environment, which then would be drawn by a loop iterating over each node. This approach, however,

can not take into account the relations between the nodes. We would only know the index of a node, from which it was hard to derive precise $x$ and $y$ coordinates that would not overlap or clash with the other nodes. This problem was amplified by the fact that LaTeX-commands themselves do not take up text space on their own, which made it necessary to calculate that required space between nodes on-the-fly, which proved impossible.

Hence, it was necessary to create an intermediary data structure, where we could store the sequential structure of the given form, from which it would then be straightforward to create fitting `tikz` nodes with the proper distances.

Such a data structure can be implemented using the `arrayjobx`[2] package, which allows for simple array operations. We therefore use an array called `orderarray`, in which the structure of the form is stored similar to the example in 2:



$\equiv$ `l3 l2 l1 e2 t1 r1 e3 t2 r2 t3 r3`

With "`l<index>`" standing for a left bracket with the corresponding index, "`r<index>`" for a right bracket, "`e<index>`" for a re-entry, and "`t<index>`" for a piece of text, usually a variable. The content of that field is stored in a further array, `textarray`.

## 3.3 Internal workflow

As the user invokes the `\lofc{<content>}{<text>}{<index>}` command in a corresponding environment, four internal helper commands are executed:

- to insert a "left" marker with the corresponding index in the `orderarray`
- to insert the text in the `textarray`
- to insert a "text" marker with the corresponding index in the `orderarray`
- to insert a "right" marker with the corresponding index in the `orderarray`

Here, recursive execution of nested commands is performed between the first and second step. The resulting data structure is then stored until, either

manually or at the end of a `lof`-environment, the `\formdraw` command is executed. This command contains two consecutive loops.

The first loop, is iterating over the `orderarray`, checks for the type of entry (left, right, re-entry, etc.), and creates a corresponding `tikz`[4] node. Depending on the type of entry, a fitting horizontal spacing is created to avoid overlaps as described in 3.2:

- a left marker requires no horizontal spacing
- a right marker, as well as a re-entry marker requires horizontal spacing behind the created `tikz`[4] node
- a text marker required requires horizontal spacing before as well as behind the created `tikz`[4] node

Further, the text stored in the `textarray` is printed out at the corresponding positions, as we now acquired correct spacing and know exactly where in our structure these variables or descriptions belong.This spacing is set to a constant value of $0.2em$ ($1em$ being the width of an "M," a common unit used in LaTeX), a global variable set at the beginning of the package.

After this process, we have created a structure of `tikz`[4] nodes, which we now need to connect through the second loop.

We now iterate over all `tikz`[4] nodes in order. Starting from `l1`, accounting for vertical shift according to the index number, we draw a horizontal line to `r1`, followed by a vertical line to the bottom of the text line. We then check, via the helper command `\ifnodedefined`, if there exist a re-entry node `e1`. If that is the case, we draw from the end of `r1` horizontally to `e1`, followed by a small vertical line to indicate the re-entry. This is repeated until we encounter a `l<index>` node that doesn't exist, upon which the loop is terminated.

The `\formdraw` command now clears any used data structures and counter that have been used, so it can be used again for further forms.

# 4 Usage

We will now explain the usage of this package for user who want to draw forms using the existing features, as well as how to implement additional features.

## 4.1 Using existing features

The existing features facilitate the following commands to users:

**`\lof{<content>}{<index>}`**

This command creates a new cross, with the corresponding index, at the current location. As mentioned previously, by "content" we mean further, nested forms.

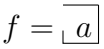
**`\lofc{<content>}{<text>}{<index>}`**

This command creates a new cross, including text, and the corresponding index, at the current location. For example:

```
\begin{lawofforms}
f = \lofc{}{a}{1}
\end{lawofforms}
```

**`\re{<index>}`**

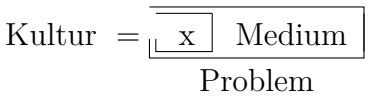
This command creates a re-entry, originating from the given index, re-entering at the current location. For example:



```
\begin{lawofforms}
f = \lofc{\re{1}}{a}{1}
\end{lawofforms}
```

**`\ren{<text>}{<index>}`**

This command creates a labelled re-entry, originating from the given index, re-entering at the current location. For example:



```
\begin{lawofforms}
\text{ Kultur } = \
 lofc{\ren{Problem}{2}
  \lofc{\re{1}}{\text{
  x }}{1}}{\text{
 Medium }}{2}
\end{lawofforms}
```

At the present, this command does not support adjusted spacing for additional re-entries below this re-entry.

`\formdraw`

### The `lawofforms` environment

All these commands should be used in a LaTeX environment, which automatically draw the desired form. The `lawofforms` environment is a `\[...\]` environment, where `\formdraw` is called at the end. This assumes that the users' equation ends with the desired drawing. If that is not the case, you will need to call `\formdraw` manually, directly after the `\lofc` commands, as follows:

$$f = \lofc{\re{1}}{a}{1} = g$$

```
\[
  f = \lofc{\re{1}}{a}{1}
  \formdraw = g
\]
```

### The `lofinline` environment

This environment is used to place form drawing in a text block. Necessary spacing is automatically calculated to align with surrounding text, for example:

This is the start of an example text $h_6 = \lofc{\lofc{\lofc{\lofc{\lofc{\lofc{}{h_8}{1}}{h_6}{2}}{h_1}{3}}{h_8}{4}}{a}{5}}{h_2}{6}$ this is the middle of an example text $f = \lofc{\lofc{\lofc{}{a}{1}\re{2}}{b}{2}}{c}{3}$ this is the end of an example text.

```
This is the start of an example text \begin{lofinline}
h_6 = \lofc{\lofc{\lofc{\lofc{\lofc{\lofc{}{h_8}{1}}{h_6}{2}}{h_1}{3}}{h_8}{4}}{a}{5}}{h_2}{6}
\end{lofinline} this is the middle of an example text \begin{lofinline}
f = \lofc{\lofc{\lofc{}{a}{1}\re{2}}{b}{2}}{c}{3}
\end{lofinline} this is the end of an example text.
```

## 4.2 Extending Features

To extend this package with additional features, there are several key places in the code where certain features can be added or changed.

The first place to start are the definitions of the actual commands that are used to encode the desired forms, i.e. `\lof`, `\lofc`, `\re`, and `\ren`, and their respective helper functions (see below) that enter corresponding elements into the `\orderarray`. To create a fundamental new feature, one will need to first create the command which will be used in the `\lawofform` environments. This command will then have to set the appropriate `\orderarray` entries, with the following commands:

`\enterOAL` : Creates an entry for the start of a cross in the `\orderarray`.
Syntax: `\enterOAL{<index>}`

`\enterOAR` : Creates an entry for the end of a cross in the `\orderarray`.
Syntax: `\enterOAR{<index>}`

`\enterOAT` : Creates an entry for the text under a cross in the `\orderarray`.

Syntax: `\enterOAT{<index>}`

`\enterOAE` : Creates an entry for a re-entry in the `\orderarray`.

Syntax: `\enterOAE{<index>}`

`\enterOAN` : Creates an entry for a re-entry with text in the `\orderarray`.

Syntax: `\enterOAN{<index>}`

It is likely that a new feature will require a new type of entry, and therefore an additional `\enterOA` command, in the `\orderarray`. To then implement the new feature, the `\formdraw` command will need to be extended to check for new types of entries, in its first loop as explained in 3.3.

If the desired behaviour requires a new type of `tikz`[4] node (the types are corresponding to the orderarray entries except for the text), the second `formdraw` loop, in which we draw the crosses, will then have to be extended to account for the new type of node. Additional checks for new types of nodes can be simply implemented with the `\ifnodedefined` helper command. Nodes that have been drawn need to be deleted with the `\aeundefinenode` command at the end of the iteration, so as to free up the name-space for the next use.

If you do not wish to implement a new drawing feature, but simply need to create a new environment in which to use our package, for example a multi-line math environment, you can simply do that by using the `\newenvironment` command, and specifying your desired formatting and environment. Note that it is assumed that our commands are used in a math environment, and that you will have to call the `\formdraw` command at the end of your environment, or whenever you want your form to be drawn. For example in the case of a multi-line math environment, it is likely that you want to call the `\formdraw` command at the end of every line.

# 5 Conclusion and further Features

To conclude, we implemented a fairly simple and concise package to elegantly display formulas using the Laws of Form[3], that can be easily expanded and

extended with further features. The main problems of encoding, internal representation, and storage of data structures have been solved, and thus a robust, flexible framework on which further components can be built upon has been established. Such features may include support for nesting to the right of form text, as described in 3.1, multiple re-entries coming from a single cross, adapted spacing for labelled re-entries - or even automated spacing that recognizes when a segment of the given formula is completed, and therefore resets the vertical spacing accordingly.

# References


[1] David Carlisle. Conditional commands in LaTeX documents. https://ctan.org/pkg/ifthen.

[2] Zhuhan Jiang. Arrayjobx. https://ctan.org/pkg/arrayjobx.

[3] G. Spencer-Brown. *Laws of Form*. 1969.

[4] Till Tantau. Tikz. https://www.ctan.org/pkg/pgf.

[5] The LaTeXTeam. AMS mathematical facilities for LaTeX. https://ctan.org/pkg/amsmath.

[6] Christian Tellechea. String manipulation for LaTeX. https://www.ctan.org/pkg/xstring.