LaTeX

To reset or not to reset

Johannes Braams

Abstract

This article describes two possible implementations of a `\@removefromreset` macro that can be used to remove a counter from the reset list of another counter.

1 Introduction

When writing a document class it is sometimes necessary to instruct LaTeX that a certain counter has to be reset when another counter gets a new value. This is the case when one wants to number equations within sections. For this purpose LaTeX has the internal command `\@addtoreset`.

Lately people have requested to do the opposite; when they use a document class that has set equation numbering to be within sections they want to be able to number the equations consecutively throughout the document. For this one would need the command `\@removefromreset`, but that command is *not* available in LaTeX.

2 The reset list

When a LaTeX counter is defined using the command `\newcounter` a number of data structures are set up that belong to that counter. Say we allocate a counter foo with the command

```
\newcounter{foo}
```

Then among other things the command `\thefoo` is defined which is used to represent the value of the counter in printed text. One of the other things that are set up is the 'reset list'. This reset list is a list of counters that are to be reset when the counter foo receives a new value with one of the commands `\stepcounter` or `\refstepcounter`. The reset list for the counter foo is stored in the macro `\cl@foo`.

Before we can start to think about the implementation of `\@removefromreset`, we have to know what kind of data structure is used to store a reset list. When we look up the definition of `\@addtoreset` to find out how it works we find the following piece of code:

```
\def\@addtoreset#1#2{%
  \expandafter\@cons
    \csname cl@#2\endcsname {{#1}}}
```

This tells us that `\@addtoreset` is a command that has two arguments, the first of which is the name of a counter to be added to the reset list of the second argument. This is done using the command `\@cons`, so to find out more about the data structure we have to keep digging. Notice that the name of the counter to add to the reset list is passed to `\@cons` inside an extra pair of braces!.

Searching for the definition of `\@cons` reveals:

```
\def\@cons#1#2{%
  \begingroup
    \let\@elt\relax
    \xdef#1{#1\@elt #2}
  \endgroup}
```

This shows us that the reset list is a list of counter names, separated by the command `\@elt`. So the expansion of `\cl@foo` could look like:

```
\cl@foo -> \@elt {bar}\@elt {baz}\@elt {cnt}
```

So, when the command `\stepcounter{foo}` is executed the counters bar, baz and cnt are all reset (get the value 0).

3 Removing an element from the reset list, the idea

Now that we know what the data structure looks like we can start to think about how to remove an element from the list. The essential piece of information we have learned from our search is that *each* counter name in the reset list is preceded by the command `\@elt`.

So the way to the solution to our problem is obvious. We have to give the command `\@elt` a new definition. What should it do? The first thing that comes to mind is that it should compare the name following it with some other name. When those two names are the same we have found the name of the counter to be removed from the list. But how to do that? A solution for this is to build up a new reset list while processing the existing list. If we do that we simply do not include the counter to be removed in the new reset list.

4 Removing an element from the reset list, the implementation

Now that we know the basic idea of how to solve the problem we can start the implementation. I will show two possible implementations.

4.1 First implementation

We are going to define the command `\@removefromreset`. It will have two arguments. The first argument is the name of the counter to remove; the second argument is the name of the counter whose reset list has to be changed.

```
\def\@removefromreset#1#2{%
```

The first thing to do is to start a group and store the name of the counter to remove from the reset list in a command.

```
  \begingroup
  \def\toremove{#1}%
```

Then we save a copy of the current reset list. We do this because we shouldn't overwrite it while rebuilding a new version.

```
  \expandafter\let\expandafter\old@cl
    \csname cl@#2\endcsname
```

In order to rebuild the reset list from scratch, we empty it.

```
  \expandafter\let\csname cl@#2\endcsname
                       \empty
```

Now we are set up to process the elements of the reset list, except for the proper definition of `\@elt`. Remember that `\@elt` will be defined by the execution of `\@removefromreset` so we have to double the `#` marks for the argument of `\@elt`.

```
  \def\@elt##1{%
```

First we store the argument of `\@elt` in a command in order to be able to use `\ifx` later on for the comparison.

```
    \def\found{##1}%
```

Now we can compare the name of the counter to remove from the list with the name we have just found.

```
    \ifx\toremove\found
```

If they are the same we do nothing, thereby effectively removing it from the list. If they are different we use `\@addtoreset` to build up the new reset list.

```
    \else
      \@addtoreset{##1}{#2}%
    \fi}%
```

Now we have defined `\@elt` so we can simply execute the reset list. This will execute all the occurrences of `\@elt` that are in the list.

```
  \old@cl
```

All that is left to do now is to close the group so that TEX forgets about any temporary definitions we made. Notice that the new reset list was built using `\@addtoreset` which uses global definitions inside.

```
  \endgroup}
```

4.2 Second implementation

A slightly different approach is taken in the following implementation of `\@removefromreset`.

```
\def\@removefromreset#1#2{%
  \begingroup
```

This time we use a token register to temporarily store the new reset list that is to be built up.

```
    \toksdef\newlist8\newlist{}
```

Again we store the first argument in a control sequence for future use in the `\ifx` test.

```
    \def\toremove{#1}%
```

Again we use `\@elt` to check whether the following list-element is the one we are looking for.

```
    \def\@elt##1{%
```

Store the list element in a control sequence

```
      \def\found{##1}
```

and compare it with the one to remove.

```
      \ifx\found\toremove
      \else
```

If it was not the one we are looking for, add the current list element to the new copy of the list, stored in token register `\newlist`.

```
        \expandafter\newlist\expandafter{%
          \the\newlist\@elt{##1}}
      \fi}
```

Now we can simply execute the reset list which will execute all the occurrences of `\@elt` that are in the list.

```
    \csname cl@#2\endcsname
```

Finally, we have to remember to copy the contents of `\newlist` to the original reset list.

```
    \expandafter\xdef\csname cl@#2\endcsname
                    {\the\newlist}
  \endgroup}
```

◇ Johannes Braams
PTT Research
St. Paulusstraat 4
2264 XZ Leidschendam
The Netherlands