INSTRUCTION FOR USING texHash.sty

LAURENCE R. TAYLOR

This package allows you to write macros for which the inputs are named. For example if you are using the semi-direct product a lot, you might want a macro

```
\def\semidirect#1#2#3{{#2}\rtimes_{#3}{#1}}
```

so that `\semidirect{G}{H}{\psi}` gave you $H \rtimes_\psi G$. But twenty pages later you might not remember which variable is the group and which is the subgroup. Using texHash.sty you can set things up so you can write
`\semidirect{'group'=>"G",'subgroup'=>"H",'morphism'=>"\psi"}`.

Moreover, values persist so the next time you need $H \rtimes_\psi G$ you can just write `\semidirect{}` unless you changed some fields. You can also use the individual fields in your text.

1. The basics

The main command, `\newhashcommand`, takes four inputs. The first and last are optional. An invocation of `\newhashcommand` creates a new command with a single input. That input is a Perl-type hash list of key-value pairs.

(1) The first required input passed to `\newhashcommand` is the name of the command you wish to create.
(2) The second required input is the code to produce the output.
(3) The first optional output is the name-space for the key-value pairs. By default the name-space is the name of the macro you are defining, but it can be set to any legal TEX name.
(4) The second optional input is a list of default values.

A new-hash-command takes only one variable but it is in the form of a list of key-value pairs. The list resembles a Perl hash list (if that helps you) but the formatting requirements are more rigid. The macros in this package generate a sequence of appropriate TEX macros you can use in the definition part of the command.

2. First examples

Suppose you wish to create a new command \foo with two hash keys, 'base' and 'exponent' so that the output is the base value raised to the power of the exponent value. The command to create this is

```
\newhashcommand{\foo} {\foobase^{\fooexponent}}
```

and it is used for example as

```
$$\foo{'base'=>"x",'exponent'=>"w",}$$
```

which will produce x^w.

Notice you did not include the optional variables so the two TEX macros you create via an invocation of \foo are \foobase and \fooexponent. You can write

```
\newhashcommand[NS]{\foo} {\NSbase^{\NSexponent}}
```

and this will still produce the same output but now the created TEX macros are \NSbase and \NSexponent. Since the TEX macros are created automatically, the only time you need to worry about the actual macro names are when the code is written in the \newhashcommand. After that you only need the key names, base and exponent in the example.

You can also refer to the current values of the keys elsewhere in your code. To get the current value of the base for example just type

```
$\fieldValue{NS}{base}$
```

and you get the current value of the base. \fieldValue has two inputs, the name-space followed by the key for the field whose value you want.

As usual with hashes, the data can be in any order, so

```
$$\foo{'base'=>"x",'exponent'=>"w",}$$ and
$$\foo{'exponent'=>"w",'base'=>"x",}$$
```

yield exactly the same input. Hence you only need to remember the keys.

The key-value list has rather strict formatting requirements. The key is offset with the ' deliminator (the single quote) and the value is offset with the " (the double quote - not two single quotes) deliminator. The => which ties them together is just the equal sign followed immediately by the > inequality. There can be no spaces in the keys, although spaces are permissible in the values. There must be a comma immediately after the closing " for each key-value pair and the next

key-value pair starts immediately after the comma with no space. Indeed, inside the { ... } there should be no spaces anywhere except inside a "-delineated value. The most common error I make is to forget the final comma in the list. This results in a "Paragraph ended before \next was complete." error.

The key-value pair creates a TEX macro whose name is
*name-space*key
where *key* is the actual key and *name-space* is either the first optional variable when you defined the hash-command or by default is the name of the hash-command. In the example above, the `base` key is turned into the TEX macro `\foobase`: the value of `\foobase` is `x`. Once a TEX macro like `\foobase` is created, it retains its value (it was defined via `\gdef`) so having created a complicated `\newhashcommand` named `\foo` you can continue to use `\foo{}` and you will get exactly what you got the last time. You can also change some fields without changing others.

Using name-spaces, you can hook hash-commands together.

For example, define

```
\newhashcommand[NS]{\foo} {\NSbase^{\NSexponent}}
```

and

```
\newhashcommand[NS]{\cofoo} {\NSexponent^{\NSbase}}
```

Then

```
$$\foo{'base'=>"x",'exponent'=>"w",}$$
```

```
$$\cofoo{}$$
```

produces

$$x^w$$

$$w^x$$

3. The second optional input for \newhashcommand

The second optional input is a default list of values. For example

```
\newhashcommand[NS]{\foo} {\NSbase^{\NSexponent}}
['base'=>"f",'exponent'=>"l",]
```

sets the default values for `base` and `exponent`.

When the hash-command is invoked as `$\foo{}$` one gets f^l unless one has changed one or both values. For example
`$\foo{'base'=>"2",}$` yields 2^l.

Default values go with the name-space and can be set/reset at any time. The command `\resetNameSpace{NS}` resets the current values of the TEX macros to the values previously defined as the default values. Continuing with our example,
`\foo{'exponent'=>"3",} \resetNameSpace{NS} \foo{}`
results in $2^3 \quad f^l$.

The collection of default values can also be modified. The command `\defaultNameSpace{`*`Name-Space`*`}{`*`values`*`}` sets the default TEX macros for the name-space in the first variable to the key-value list in the second variable.

As an example, after `\defaultNameSpace{NS}{'base'=>"r"}`, the `\foo`-command yields r^l. The new default `base` is r while the default value for `exponent` has not changed and so it is still l.

Because TEX is not particularly good at text processing, it is currently impossible to nest hash-commands. The input mechanism chokes on the interior =>'s. You can define a macro using the interior hash-command you wanted and pass that, as for example

```
\newhashcommand{\betterE}{\betterEbase _{\betterEsubscript}}
['base'=>"4",'subscript'=>"6",]
\betterE{}
```

results in 4_6. To get this construction as the exponent in the `\foo` macro continue as follows.

```
\def\xx{\betterE{'subscript'=>"8",}}
$\foo{'exponent'=>"\xx"}$
```

to get r^{4_8}

As is often the case with TEX, time of evaluation is important in understanding the output. Meditate on the following two commands:
`\betterE{'base'=>"3",'subscript'=>"5",}` which yields 3_5 as you probably expected. But if you follow it with

$\foo{'exponent'=>"\xx",}$ you get r^{3s}.

4. A whole paragraph

Having several macros using the same name-space is handy to make sure that all the pieces you think are the same remain the same as you "improve" your notation. Using \fieldValue to get the current values of the fields when you refer to them helps insure uniformity.

Here is a paragraph tied together nicely.

```
\newhashcommand[HOM]{\homology}{H_{\HOMdimension}\HOMleft
\HOMspace ; \HOMcoeff\HOMright}
['dimension'=>"\ast",'left'=>"(",'right'=>")",'space'=>"X",
'coeff'=>"\mathbb Z",]

Given any space $\fieldValue{HOM}{space}$
there is an associated homology group
$\homology{}$.
Given a continuous function
$f\colon \fieldValue{HOM}{space} \to Y$
there is induced a group homomorphism
$$f_{\fieldValue{HOM}{dimension}}\colon
\homology{}\to \homology{'space'=>"Y",}$$

\resetNameSpace{HOM}

Given a subspace
$\fieldValue{HOM}{subspace}\subset\fieldValue{HOM}{space}$
there is a relative homology group, $\homologyrel{}$
and a long exact sequence
$$\cdots\to\homologyrel{'basedim'=>"r",
'dimension'=>"\fieldValue{HOM}{basedim}+1",}
\to
\homologysub{'dimension'=>"\fieldValue{HOM}{basedim}",} \to
\homology{} \to \homologyrel{}
\to \cdots$$
This typesets as follows.
```

Given any space X there is an associated homology group $H_*(X;\mathbb{Z})$. Given a continuous function $f\colon X \to Y$ there is induced a group homomorphism

$$f_*\colon H_*(X;\mathbb{Z}) \to H_*(Y;\mathbb{Z})$$

Given a subspace $A \subset X$ there is a relative homology group, $H_*(X, A;\mathbb{Z})$ and a long exact sequence

$$\cdots \to H_{r+1}(X, A;\mathbb{Z}) \to H_r(A;\mathbb{Z}) \to H_r(X;\mathbb{Z}) \to H_r(X, A;\mathbb{Z}) \to \cdots$$

In texHash.sty keys are evaluated from left to right in order so "trickery" like the above can be worked. We defined a new key, `basedim` and used it to compute the two fields we actually used, so for example if we later decide we'd like to use s instead of r as our base subscript, one change does it.

On the other hand, it does mean we lied slightly when we said the order of the key-value pairs in the list made no difference. Most of the time it doesn't, but the list is always evaluated left to right.

5. `\renewhashcommand`

Finally there is a `\renewhashcommand` which is identical to the `\newhashcommand` except that it renews the hash-command in question. The usual LaTeX mechanism to prevent redefining a command is in effect and so if you want to redefine a hash-command, you need the `\renewhashcommand` command.

Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556

E-mail address: taylor.2@nd.edu