

[Cayley/MAGMA Index] [back] [next]

Cayley/MAGMA: some similarities and differences

Subsections

- Case
- Assignment operators
- Function names and Keywords
- Syntax
- Structures
- Membership coercion
- Reading in a file
- Reading in a file at startup
- Log files
- Setting aliases

Case

Cayley *does not* recognise case -- A and a represent the same character as far as Cayley is concerned; MAGMA *does* recognise case.

Assignment operators

In Cayley there are essentially three assignment operators: =, := and :; in MAGMA there is one assignment operator: :=.

Function names and Keywords

Built-in functions of Cayley have names consisting of words separated by spaces; generally the same functions exist in MAGMA -- the names being created from the Cayley ones by capitalising the first letter of each word and deleting the spaces, e.g.

`composition series (Cayley)`

becomes

`CompositionSeries (MAGMA)`

Boolean-valued functions in MAGMA however are prefixed by `Is`, e.g.

`conjugate (Cayley)`

becomes

`IsConjugate (MAGMA)`

Keywords in MAGMA, remain in lower case, e.g.

if, then, while, do, end, procedure, print

Syntax

Generally there are only cosmetic differences in syntax. Constructions like `if`, `while`, `procedure` terminate with `end` in Cayley; and terminate with `end if`, `end while` etc. (two words) in MAGMA.

The `for each` and `for` constructions of Cayley have been replaced with `for` in MAGMA, and is modelled on the `for each` construction of Cayley.

A string in Cayley is enclosed in a pair of right quotes; a string in MAGMA is enclosed in a pair of double quotes, e.g.

`'abcde'` (Cayley)

becomes

`"abcde"` (MAGMA)

A set in Cayley is enclosed by square brackets; a set in MAGMA is enclosed by curly braces, e.g.

`[a,b,c]` (Cayley)

becomes

`{a,b,c}` (MAGMA)

A sequence in Cayley is enclosed by `seq(...)`; a sequence in MAGMA is enclosed by square brackets, e.g.

`seq(a,b,c)` (Cayley)

becomes

`[a,b,c]` (MAGMA)

Structures

Structures like groups, rings, fields are defined quite differently in MAGMA. Bourbaki defined a magma as a set with one or more laws of composition. Thus, groups, rings, fields etc. are magmas and each of these have submagmas and quotient magmas. The recognition of the commonalities of these structures give rise to the name of MAGMA. To get at a submagma there is the `sub<... | ...>` constructor. To get at a quotient magma there is the `quo<... | ...>` constructor. As an example, to define the group,

$G = \langle (1\ 3), (1\ 2\ 3\ 4) \rangle$

in Cayley requires:

```
G : perm(4);
G.generators : (1,3), (1,2,3,4);
```

whereas in MAGMA the same thing can be achieved by:

```
G := sub< Sym(4) | (1,3), (1,2,3,4) >;
```

Membership coercion

To specify an element is from a given structure in Cayley one uses `of`; in MAGMA one uses the coercion (or bang) operator: `!`, e.g.

```
(1,3) of G (Cayley)
```

becomes

```
G!(1,3) (MAGMA)
```

Reading in a file

To read in a file `file` of Cayley commands while running Cayley, one types

```
library file;
```

Further, `file` must begin with

```
library file;
```

and end with

```
finish;
```

To read in a file `file` of MAGMA commands while running MAGMA, one types

```
load "file";
```

and no special beginning or ending lines are required in `file`.

Reading in a file at startup

To read in a file `file` of Cayley commands when starting up Cayley, one types

```
cayley -c "library file;"
```

To read in a file `file` of MAGMA commands when starting up MAGMA, one types

```
magma -s file
```

Log files

To cause Cayley to copy all the screen output to file `caylog` one starts up Cayley with

```
cayley -l
```

To achieve the same thing dynamically (i.e. after starting up a Cayley session) one can also type

```
set logging = true;
```

In MAGMA one adds the line

```
SetLogFile("mylogfile");
```

to the startup file `file`, to cause screen output of MAGMA to be copied to the file `mylogfile` (i.e. the name of the logfile is chosen by the user) and then when starting up MAGMA, one types

```
magma -s file
```

as above.

To achieve the same thing dynamically one simply types

```
SetLogFile("mylogfile");
```

Setting aliases

To create aliases for long or hard-to-remember built-in functions in Cayley one has to build a mapping with `define` or a procedure with `procedure --` neither of these, copes in a general way with a function that can take a variable number of arguments, and when using `procedure` the syntax of the original function is altered. In MAGMA one just assigns the identifier corresponding to the function to another identifier. For example,

```
define stab(G,n)=stabilizer(G,n); (Cayley)
```

becomes

```
stab := Stabilizer; (MAGMA)
```

This is the sort of thing one might like to put in a start-up file.

