Random Notes about SYMSCI Toolbox

Symbolic Math Toolbox for Scilab

16th February 2017

These notes refer to SYMSCI Toolbox 0.3 under SCILAB 5.5.2

Contents

1 Naming of Functions

In general, SYMSCI adopts the lower *and* upper camel case naming convention to indicate, whether the function returns a *symbolic* object, a *native* SCILAB data type or nothing.

upper camel case function returns a *symbolic* object (e.g. `Sym`, `Add`, `GetArgs`, ...)

lower camel case function returns a *native* Scilab data type or nothing (e.g. `isSym`, `typeClass`, `toString`, ...)

2 Wormhole aka Symbolic Object

The connection of native Scilab with the inner guts[1] of SYMSCI is based on `mlist` Scilab objects, which point to *symbolic* objects. Such objects represent a new *symbolic data type*. The predicate function `isSym(obj)` may be used, to check whether an `obj` is symbolic (`%t`) or not (`%f`). Every symbolic object is either a *basic* object (cf. `isScalar`) or a *container* (cf. `isSet`, `isVector`, `isMatrix`) of basic objects. Furthermore, every basic object is classified further (cf. `typeID`, `typeClass`) as `Symbol`, `Integer`, `Rational`, `Add`, `Mul`, `Div`, or otherwise ....

3 Working with Symbolic Variables

Due to the pointer nature of symbolic objects (cf. above), they are marked either as *intermediate* `Value` or *persistent* `Variable`. By default, every created symbolic object is a `Value`, which might be automatically deleted after its first usage or upon request (cf. Memory Management). Such `Value` objects are **not** suitable for assignment to SCILAB variables.

When working with symbolic variables you should always perform the following triple jump:

Creation First mark a symbolic object using the `Variable` (shorthand: `%`) function and then assign it to the preferred SCILAB variable, e.g.

```
x = %(Sym(„x+2"))
y = %(2*x)
```

Usage Use the `Assign` (shorthand: „`:`“) function and **not** „=“ for changing the value of a previously created variable, e.g.

```
x :  2+y
y :  2*x
```

Deletion Use the `free` function to delete a symbolic object, *before* the associated SCILAB variable runs out of scope. Please note, the `Scilab` variable may still exist, but it carries now a dangling pointer. Thus, it is completely useless and should be `clear`ed it right away, e.g.

```
free(x)
free(y)
clear x y
```

[1] SYMENGINE https://github.com/symengine/symengine

4 Memory Management

SYMSCI provides a *very* primitive memory managment based on the distinction between `Value` and `Variable` objects (cf. above). There a two possible modes for garbage collection, they are selected and controlled by the function `gc`.

Manual This is the *default* mode. It is the most convenient mode to carry out interactive calculations in the SCILAB console window. Nothing is deleted automatically, but one may use `gc()` from time to time to delete all `Value` objects manually.

Automatic This mode is useful within user-defined functions. If it is selected by `gc(%t)`, then every `Value` object is deleted automatically after its first usage. Switching back to manual mode is done by `gc(%f)`.

Apart from `gc`, there are the two more utilities regarding memory managment:

heap() Returns the total number of currently allocated symbolic objects.

reset() Deletes *all* symbolic objects.

5 Code Generation

It is possible to generate SCILAB code, or even ready-to-run SCILAB functions from symbolic expressions. Please refer to the help pages of `toSciCode` and `toSciFunction` for further explanations and examples.

6 Epilog

The SYMSCI toolbox is in a work-in-progress state, and thus may contain numerous and severe bugs. Any feedback is appreciated and should be sent to info@kybdr.de